INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N5555

Pattaya, March 2003

Source **JVT**
Title **Draft Text of Final Draft International Standard for Advanced Video Coding (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)**
Status ***Approved for final editing period with delivery by 31 March 2003***

Title page to be provided by ITU-T | ISO/IEC

DRAFT INTERNATIONAL STANDARD
DRAFT ISO/IEC 14496-10 : 2002 (E)
DRAFT ITU-T Rec. H.264 (2002 E)
DRAFT ITU-T RECOMMENDATION

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Foreword

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardisation Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardising telecommunications on a world-wide basis. The World Telecommunication Standardisation Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups that, in turn, produce Recommendations on these topics. The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1. In some areas of information technology that fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

ISO (the International Organisation for Standardisation) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide standardisation. National Bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, 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 JTC1. 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.

This Recommendation | International Standard was prepared jointly by ITU-T SG16 Q.6, also known as VCEG (Video Coding Experts Group), and by ISO/IEC JTC1/SC29/WG11, also known as MPEG (Moving Picture Experts Group). VCEG was formed in 1997 to maintain prior ITU-T video coding standards and develop new video coding standard(s) appropriate for a wide range of conversational and non-conversational services. MPEG was formed in 1988 to establish standards for coding of moving pictures and associated audio for various applications such as digital storage media, distribution and communication.

In this Recommendation | International Standard Annexes A through E contain normative requirements and are an integral part of this Recommendation | International Standard.

0 Introduction

0.1 Prolog

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 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 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 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 parameters 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 and parameters are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream. In order to specify constraints on the syntax (and hence

define a profile), it is thus only necessary to constrain the values of these flags and parameters that specify the presence of later syntactic elements.

0.5 Overview of the syntax

The coded representation specified in the syntax achieves 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. The expected encoding algorithm (not specified in this Recommendation | International Standard) in inter coding first uses block-based inter prediction to exploit temporal statistical dependencies or in intra coding first uses spatial prediction to exploit spatial statistical dependencies in the source signal. 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 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

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 using bi-predictive inter coding may 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 interlaced video

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 time while the two fields of a progressive frame share the same time. Each field may be coded separately or they may be coded together as a frame. Progressive frames are typically coded as a frame.

Each frame of interlaced video consists of two fields that are separated in capture time. This Recommendation | International Standard allows either the representation of complete frames or the representation of individual fields. Frame encoding or field encoding can be adaptively selected on a picture-by-picture basis and also on a more localized basis within a coded frame. Frame encoding is typically preferred when the video scene contains significant detail with limited motion. Field encoding, in which the second field can be predicted from the first, works better when there is fast picture-to-picture motion.

0.5.3 Picture partitioning into macroblocks and smaller partitions

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

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

It is recommended 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.

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

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 Recommendations and International Standards contain provisions that, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardisation Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- ITU-T Recommendation T.35 (2000), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*
- ISO/IEC 11578:1996, Annex A, *Universal Unique Identifier*
- ISO/CIE 10527:1991, *Colorimetric Observers*

3 Definitions

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

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

3.2 **adaptive binary arithmetic decoding**: An entropy decoding process that recovers the values of *bins* from a sequence of bits produced by an *adaptive binary arithmetic encoding* process.

3.3 **adaptive binary arithmetic encoding**: An entropy encoding process, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bin* values by means of a recursive interval subdivision process using the adaptively estimated probabilities of the individual *bins*.

3.4 **arbitrary slice order**: An ordering of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* may be smaller than the *macroblock address* of the first *macroblock* of some other *slice* of the *picture* that precedes it in *decoding order*.

3.5 **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*.

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

3.7 **binarization**: The set of *binary representations* of all possible values of a *syntax element*.

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

3.9 **bin string**: A string of *bins*. A *bit string* is an intermediate binary representation of values of syntax elements.

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

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

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

3.13 **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*.

3.14 **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*.

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

3.16 **byte**: A sequence of 8 bits, ordered from the first and most significant bit on the left to the last and least significant bit on the right.

3.17 **byte-aligned**: A bit in a *bitstream* is *byte-aligned* if its position is a multiple of 8 bits from the first bit in the *bitstream*.

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

3.19 **category**: A number associated with each syntax element that specifies 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.

3.20 **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.

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

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

3.23 **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame* and may be either a *primary coded picture* or a *redundant coded picture*. In cases where a distinction is relevant but is not explicitly stated, the term coded picture shall be interpreted to refer the *primary coded picture*.

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

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

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

3.27 **complementary non-reference field pair**: Two *non-reference fields* that are adjacent in *decoding order* as two *coded fields* where the first field is not already a paired *field*.

3.28 **complementary reference field pair**: Two *reference fields* that are adjacent in *decoding order* as two *coded fields* and share the same value of *frame number*.

3.29 **context variable**: A variable specified for the decoding process of a *symbol* by an equation containing recently *decoded symbols*.

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

3.31 **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*.

3.32 **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.

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

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

3.35 **decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and produces *decoded pictures*.

3.36 **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.

3.37 **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*.

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

3.39 **encoding process**: A process, not specified in this Recommendation | International Standard, that reads a sequence of *fields* and *frames* and produces a conforming *bitstream* as specified in this Recommendation | International Standard.

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

3.41 **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.

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

3.43 **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*.

3.44 **flag**: A variable that can take one of only two possible values.

3.45 **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*.

3.46 **frame macroblock**: A *macroblock* representing samples from 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.

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

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

3.49 **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.

3.50 **I slice**: A *slice* that is decoded using prediction only from decoded samples within the same *slice*.

3.51 **instantaneous decoding refresh (IDR) picture**: A *coded picture* containing only *I* or *SI slice types* that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately before *decoding* the IDR picture, and to indicate that later *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 *video sequence* is an IDR picture.

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

3.53 **inter prediction**: A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*. *Inter prediction* is a collective term for the *prediction process* invoked for P or B macroblock types.

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

3.55 **intra prediction**: A *prediction* derived from the decoded samples of the same *decoded picture*. *Intra prediction* is a collective term for the *prediction process* invoked for I or SI macroblock types.

3.56 **intra slice**: See I slice.

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

3.58 **layer**: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the sequence, *picture*, *slice*, reference picture selection [Ed. Note(GJS): That's not the proper name], *macroblock*, prediction block [Ed. Note(GJS): Is that the right name?], and 4x4 block layers.

3.59 **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 level definitions is used with all *profiles*, but 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*.

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

3.61 **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*).

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

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.

3.63 **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*.

3.64 **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*.

3.65 **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.

3.66 **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 if a macroblock is a *bottom macroblock*.

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

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

3.69 **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.

3.70 **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.

3.71 **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*.

3.72 **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*.

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

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

3.75 **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*.

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

3.77 **output order**: The order in which the *decoded pictures* are intended for output from the decoded picture buffer.

3.78 **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*.

3.79 **parameter**: A syntax element of a *sequence parameter set* or a *picture parameter set*. [Ed. Note(WYK): Not accurate, since “parameter” is used in many other places such as “quantisation parameter”, “HRD parameter”. Is the definition really necessary?]

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

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

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

3.83 **picture order count**: *Picture* position in output order, relative to the previous IDR *picture* in *decoding order*.

3.84 **picture reordering**: The process of reordering the decoded *pictures* when the *decoding order* is different from the *output order*.

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

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

3.87 **predictive slice**: See P slice.

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

3.89 **primary coded picture**: A primary coded representation of a *picture*. The primary coded picture contains all *slices* of the *picture* (and thus contains all *macroblocks* or *macroblock pairs* of the *picture*, as the division of the *picture* into *slice groups* is a *partitioning* and the division of the *slice groups* into *slices* is a *partitioning* for the *primary coded picture*). In cases where a distinction is relevant but is not explicitly stated, the term *coded picture* shall be interpreted to refer the primary coded picture. In particular, all pictures that have a normative effect on the decoding process are primary coded pictures (see *redundant coded pictures*).

3.90 **primary coded slice**: A *slice* belonging to a *primary coded picture*.

3.91 **primary slice data partition**: A *slice data partition* belonging to a *primary coded slice*.

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

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

3.94 **random access**: The act of starting the decoding process for a coded *NAL unit stream* at a point other than the beginning of the stream.

3.95 **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.

3.96 **raw byte sequence payload (RBSP):** A syntactical data structure containing an integer number of bytes encapsulated in a *NAL unit* that is either null 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.

3.97 **raw byte sequence payload 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*.

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

3.99 **redundant coded picture**: A redundant coded representation of a *picture* or a part of a *picture* that shall not be used for decoding unless the *primary coded picture* is missing or corrupted (a condition that is not allowed in a conforming *bitstream*). The *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.

3.100 **redundant coded slice**: A *slice* belonging to a *redundant coded picture*.

3.101 **redundant slice data partition**: A *slice data partition* belonging to a *redundant coded slice*.

3.102 **reference field**: A *reference field* is 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*.

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

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

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

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

3.107 **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*.

3.108 **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*.

3.109 **reference picture marking**: Specifies in the coded data, how the *decoded pictures* are marked for *inter prediction*.

3.110 **reserved**: The term "reserved", when used in the clauses specifying some values of a particular syntax element means that these values may be used in extensions of this Recommendation | International Standard by ITU-T | ISO/IEC, and that these values shall not be used.

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

3.112 **run**: A number of consecutive data elements represented in the decoding process. In one context, the number of zero-valued *transform coefficients* preceding a non-zero *transform coefficient*, in the *zig-zag scan* or *field scan*. In other contexts, run refers to a number of *macroblocks*.

3.113 **sample aspect ratio**: Specifies, for assisting the display process 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).

3.114 **scaling**: The process of *scaling* the *transform coefficient levels* resulting in *transform coefficients*.

3.115 **SI slice**: A *slice* that is coded using prediction only from decoded samples within the same *slice*, encoded such that it can be reconstructed identically to an *SP slice*.

3.116 **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*.

3.117 **slice**: An integer number of *macroblocks* or *macroblock pairs* ordered contiguously in *raster scan order* 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 contiguous in raster scan order within a slice group, these *macroblocks* or *macroblock pairs* are not necessarily contiguous within the picture. The addresses of the *macroblocks* are derived from the address of the first *macroblock* or *macroblock pair* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*.

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

3.119 **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*.

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

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

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

3.123 **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*, encoded such that it can be reconstructed identically to another *SP slice* or an *SI slice*.

3.124 **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*.

3.125 **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.

3.126 **sub-macroblock**: One quarter of the samples of a *macroblock*, i.e., an 8x8 luma block and two 4x4 chroma blocks of which one corner is located at a corner of the *macroblock*.

3.127 **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*.

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

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

3.130 **symbol**: A *syntax element*, or part thereof, to be decoded.

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

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

3.133 **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*.

3.134 **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*.

3.135 **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.

3.136 **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.

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

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

3.139 **video sequence**: A sequence of *pictures* that consists, in syntax order, of an *IDR picture* followed zero or more non-IDR *pictures* including all subsequent *pictures* up to but not including any subsequent *IDR picture*.

3.140 **zig-zag scan**: A specific sequential ordering of *transform coefficients* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficients* 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 **FIFO**: First-In, First-Out

4.7 **HRD**: Hypothetical Reference Decoder

4.8 **IDR**: Instantaneous Decoding Refresh

4.9 **LPS**: Least Probable Symbol

4.10 **LSB**: Least Significant Bit

4.11 **MB**: Macroblock

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

4.13 **MPS**: Most Probable Symbol

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

5.2 Logical operators

The following logical operators are defined as follows

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

5.3 Relational operators

The following relational operators are defined as follows

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

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.
|	Bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value.
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 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 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

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

Ceil(x) rounds x up to the nearest integer greater than or equal to x. (5-2)

Clip1(x) = Clip3(0, 255, x) (5-3)

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

Floor(x) rounds x down to the nearest integer less than or equal to x. (5-5)

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

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

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

Luma4x4BlkScan(x, y) = (x / 2) * 4 + (y / 2) * 8 + RasterScan(x % 2, y % 2, 2) (5-9)

Median(x, y, z) = x + y + z – Min(x, Min(y, z)) – Max(x, Max(y, z)) (5-10)

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

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

RasterScan(x, y, n_x) = x + y * n_x (5-13)

Round(x) = Sign(x) * n where n is an integer and n - 0.5 <= Abs(x) < n + 0.5 (5-14)

RasterTo4x4LumaBlkOffset(x) = (((x % 4) / 2) - ((x / 4) % 2)) * 2 (5-15)

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x \ge 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{5-16}$$

$$\text{Sqrt}(x) = \sqrt{x} \tag{5-17}$$

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

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and 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.

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 a 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 Processes

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

6 Source, coded, decoded, 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 frames and/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. The value of ChromaFormatFactor shall be inferred equal to 1.5.

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

The width and height of the luma arrays are multiples of 16 samples. This Recommendation | International Standard represents colour sequences using 4:2:0 chroma sampling. I.e., the width and height of each of the two chroma samples arrays are half of the width and height of the luma array of the same picture. The width and height of chroma arrays are multiple of 8 samples. The height of a luma array that is coded as two separate fields or in macroblock-based frame field adaptive mode (see below) is a multiple of 32 samples. The height of each chroma array that is coded as two separate fields or in macroblock-based frame field adaptive mode (see below) is a multiple of 16 samples. The width or height of pictures output from the decoding process need not be a 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).

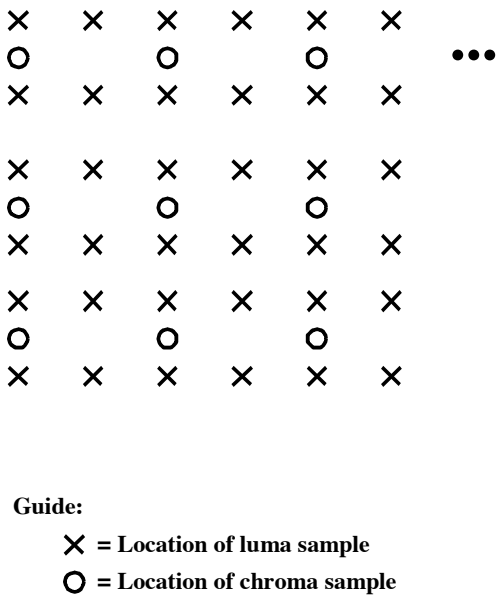


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

A video frame consists of two fields as described below. A coded picture may represent a frame or an individual field. A 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.

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. A top field consists of only the top field rows of a decoded frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even rows (for a top field) or the odd 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. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

The 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.

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.

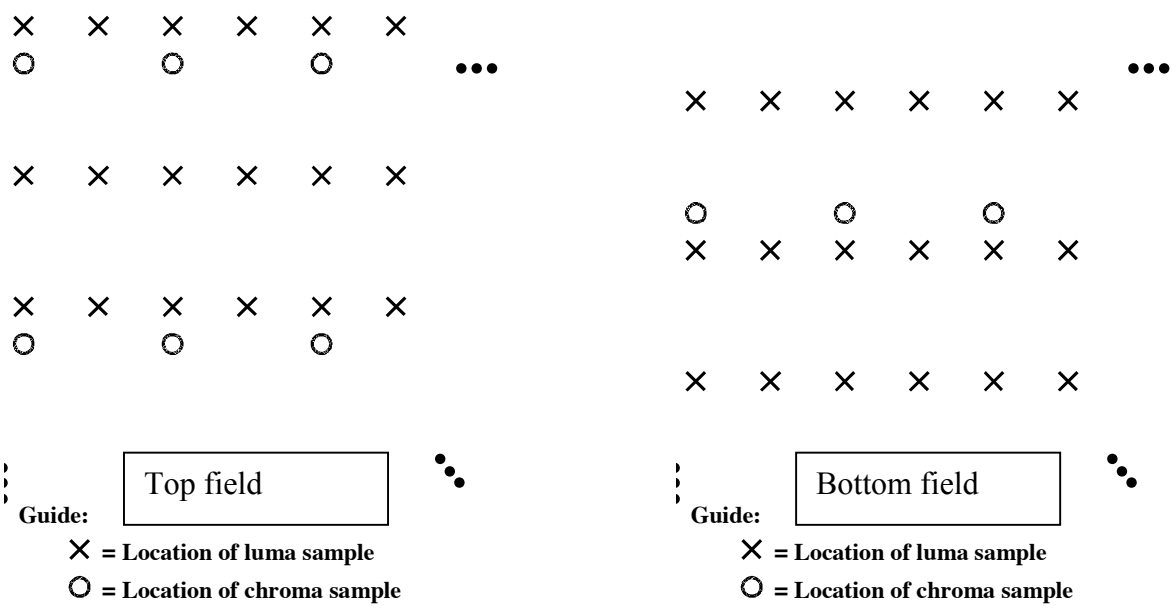


Figure 6-2 – Nominal vertical and horizontal sampling locations of samples top and bottom fields.

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. If the first slice in such a partitioning of a picture with 99 macroblocks contains 48 macroblocks, then the second slice contains 51 macroblocks as shown in Figure 6-3.

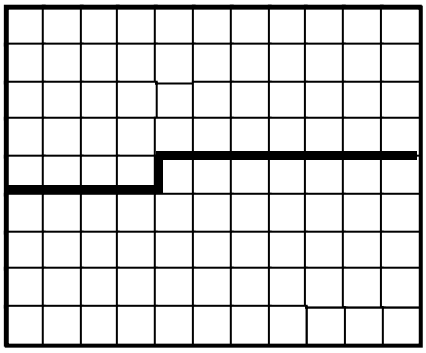


Figure 6-3 – A picture with 11 by 9 macroblocks that is partitioned into two slices

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.

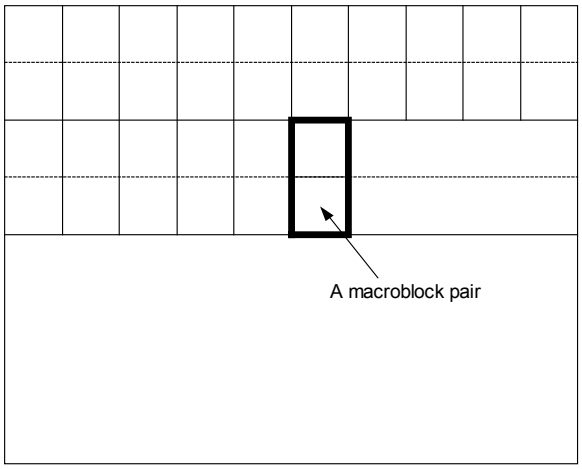


Figure 6-4 – Partitioning of the decoded frame into macroblock pairs.

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,

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

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

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

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

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

  - If the current macroblock is a frame macroblock

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

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

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

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

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

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 mb_part_width( ), mb_part_height( ), sub_mb_part_width( ), sub_mb_part_height( ) 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. mb_part_width( ) and mb_part_height( ) are set to appropriate values for each macroblock, depending on the macroblock type. sub_mb_part_width( ) and sub_mb_part_height( ) are set to appropriate values for each sub-macroblock of a macroblocks with mb_type equal to P_8x8, P_8x8ref0, or B_8x8, depending on the sub-macroblock type.

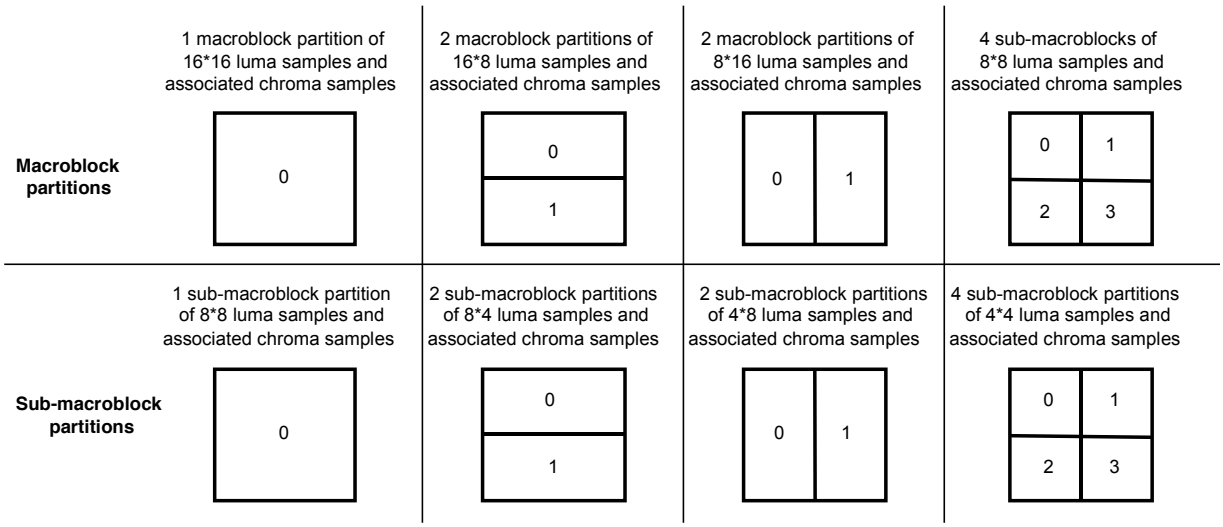


Figure 6-5 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans.

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

$$x = InverseRasterScan(mbPartIdx, mb\_part\_width(mb\_type), mb\_part\_height(mb\_type), 16, 0) \quad (6\text{-}9)$$

$$y = InverseRasterScan(mbPartIdx, mb\_part\_width(mb\_type), mb\_part\_height(mb\_type), 16, 1) \quad (6\text{-}10)$$

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.

subMbPartWidth = sub_mb_part_width(sub_mb_type[mbPartIdx]) (6-11)

subMbPartHeight = sub_mb_part_height(sub_mb_type[mbPartIdx]) (6-12)

The inverse sub-macroblock partition scanning process is specified by

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

y = InverseRasterScan(subMbPartIdx, subMbPartWidth, subMbPartHeight, 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.

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 > PicSizeInMbs – 1
- the macroblock with address mbAddr belongs to a different slice than the current slice
- the macroblock with address mbAddr is at a later position in decoding order than the current macroblock

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.

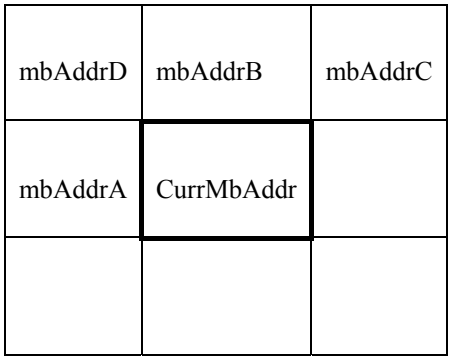


Figure 6-7 – Neighbouring macroblocks for a given macroblock

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 if ((CurrMbAddr % (PicWidthInMbs)) == 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 if (((CurrMbAddr + 1) % (PicWidthInMbs)) == 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 if ((CurrMbAddr % (PicWidthInMbs)) == 0).

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

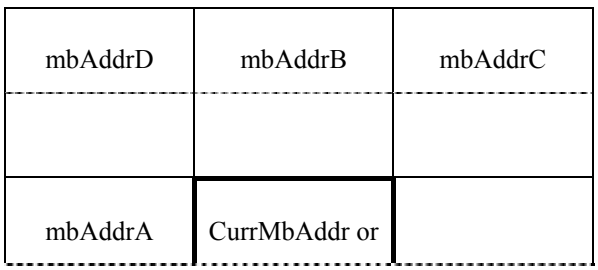
This process can only be invoked when MbaffFrameFlag is equal to 1.

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.



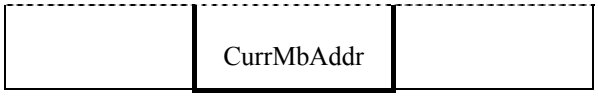


Figure 6-8 – Neighbouring macroblocks for a given macroblock in MBAFF frames

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 if (((CurrMbAddr / 2) % (PicWidthInMbs)) == 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 if (((CurrMbAddr / 2 + 1) % (PicWidthInMbs)) == 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 if (((CurrMbAddr / 2) % (PicWidthInMbs)) == 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 partWidth 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	partWidth	-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, block, or partition E, when E is in frame coding mode

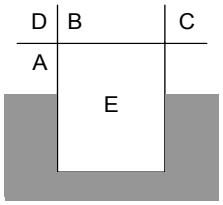


Figure 6-9 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

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 raster scan order.

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
- 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

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

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

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

- 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).
- If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.
- Otherwise, 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
- 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

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

$$xN = x + xD \tag{6-19}$$

$$yN = y + yD \tag{6-20}$$

- 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).
- If mbAddrN is not available, luma4x4BlkIdxN is marked as not available.
- Otherwise, 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
- 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

The derivation process for neighbouring 8x8 luma block is invoked with luma8x8BlkIdx = chroma4x4BlkIdx as the input and mbAddrA, chroma4x4BlkIdxA = luma8x8BlkIdxA, mbAddrB, 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 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.
- When mb_type is equal to P_8x8 or P_8x8ref 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. partWidth in Table 6-2 is specified as partWidth = sub_mb_part_width(sub_mb_type[mbPartIdx]). Otherwise, (xS, yS) are set to 0 and partWidth in Table 6-2 is specified as partWidth = mb_part_width(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

$$xN = x + xS + xD \tag{6-21}$$

$$yN = y + yS + yD \tag{6-22}$$

- 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).
- If mbAddrN is not available, the macroblock or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is marked as not available.
- Otherwise, 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.
  - If 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 if mbPartIdx = 2, subMbPartIdx = 3, xD = 4, yD = -1, i.e., if 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, then

$$maxWH = 16 \tag{6-23}$$

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

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

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

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

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

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:
   - If the macroblock with address CurrMbAddr is a frame macroblock, then currMbFrameFlag is set equal to 1; otherwise (the macroblock with address CurrMbAddr is a field macroblock), currMbFrameFlag is set equal to 0.
   - If the macroblock with address CurrMbAddr is a top macroblock, i.e., if CurrMbAddr % 2 is equal to 0, then mbIsTopMbFlag is set equal to 1; otherwise (CurrMbAddr % 2 is equal to 1), mbIsTopMbFlag is set equal to 0.
2. If mbAddrX is unavailable, mbAddrN is marked as unavailable. 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 following variable
   - If the macroblock mbAddrX is a frame macroblock, then 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 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	unavailable			unavailable	na
		0	1	mbAddrC	1		mbAddrC + 1	2 * yN
					0		mbAddrC	yN
			0	mbAddrC			mbAddrC + 1	yN
> maxWH - 1	0 .. maxWH - 1			unavailable			unavailable	na
	> maxWH - 1			unavailable			unavailable	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( )

- Returns TRUE 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. Otherwise it returns FALSE.

more_data_in_byte_stream( )

- Returns TRUE if more data follows in the byte stream. Otherwise it returns FALSE. Used only in the byte stream NAL unit syntax structure specified in Annex B.

more_rbsp_data( )

- Returns TRUE if there is more data in an RBSP before rbsp_trailing_bits( ). Otherwise it returns 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( )

- Returns TRUE if there is more data in an RBSP. Otherwise it returns 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. If used within the byte stream as specified in Annex B, 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. If n is equal to 0, then 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, and the syntax category for that syntax element or syntax structure is specified in the higher-level syntax structure that includes it.

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

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	C	Descriptor
profile_idc	0	u(8)
level_idc	0	u(8)
more_than_one_slice_group_allowed_flag	0	u(1)
arbitrary_slice_order_allowed_flag	0	u(1)
redundant_pictures_allowed_flag	0	u(1)
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)
required_frame_num_update_behaviour_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)
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_variables_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)
frame_cropping_flag	1	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	1	ue(v)
frame_crop_right_offset	1	ue(v)
frame_crop_top_offset	1	ue(v)
frame_crop_bottom_offset	1	ue(v)
}		
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 Picture delimiter RBSP syntax

pic_delimiter_rbsp( ) {	C	Descriptor
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 == 1 && !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 == 1 && !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_variables_present_flag == 1) {		
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 == 1)		
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	2	u(8)
} else {		
if(mb_part_pred_mode(mb_type, 0) != Intra_4x4 && mb_part_pred_mode(mb_type, 0) != Intra_16x16 && num_mb_part(mb_type) = = 4)		
sub_mb_pred(mb_type)	2	
else		
mb_pred(mb_type)	2	
if(mb_part_pred_mode(mb_type, 0) != Intra_16x16)		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || mb_part_pred_mode(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(mb_part_pred_mode(mb_type, 0) = = Intra_4x4 || mb_part_pred_mode(mb_type, 0) = = Intra_16x16) {		
if(mb_part_pred_mode(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(mb_part_pred_mode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < num_mb_part(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag = = 1) && mb_part_pred_mode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < num_mb_part(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag = = 1) && mb_part_pred_mode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < num_mb_part(mb_type); mbPartIdx++)		
if(mb_part_pred_mode (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 < num_mb_part(mb_type); mbPartIdx++)		
if(mb_part_pred_mode(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 = = 1) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && sub_mb_pred_mode(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 = = 1) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && sub_mb_pred_mode(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 && sub_mb_pred_mode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < num_sub_mb_part(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 && sub_mb_pred_mode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < num_sub_mb_part(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(mb_part_pred_mode(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(mb_part_pred_mode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16ACLevel[i8x8 * 4 + i4x4], 15)	3	
else		
residual_block(LumaCoeffLevel[i8x8 * 4 + i4x4], 16)	3 | 4	
} else {		
if(mb_part_pred_mode(mb_type, 0) = = Intra_16x16)		
for(i = 0; i < 15; i++)		
Intra16x16ACLevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
LumaCoeffLevel[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(CodedBlockPattern & 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(total_coeff(coeff_token) > 0) {		
for(i = 0; i < total_coeff(coeff_token); i++)		
if(i < trailing_ones(coeff_token)) {		
trailing_ones_sign_flag	3 | 4	u(1)
level[i] = 1 – 2 * trailing_ones_sign_flag		
} else {		
coeff_level	3 | 4	ce(v)
level[i] = coeff_level		
}		
if(total_coeff(coeff_token) < maxNumCoeff) {		
total_zeros	3 | 4	ce(v)
zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for(i = 0; i < total_coeff(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[total_coeff(coeff_token) – 1] = zerosLeft		
coeffNum = -1		
for(i = total_coeff(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. Framing of NAL units is necessary to enable inference of NumBytesInNALunit. Such framing is specified in Annex B for the byte stream format, and other methods for framing 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. If 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 be 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, 2, 3, 4, 5, or 12. 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 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	Picture delimiter pic_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	

No decoding process for nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, is specified in this Recommendation | International Standard.

NOTE – NAL unit types 0 and 24..31 may be used as determined by the application.

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

If one slice has nal_unit_type equal to 5, all other slices associated with the same picture shall have nal_unit_type equal to 5. Such pictures are referred to as IDR pictures. The first picture in the bitstream shall be an IDR picture.

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

rbsp[i **]** a raw byte sequence payload is specified as an ordered sequence of bytes.

The RBSP contains an SODB in the following form:

- If the SODB is null, the RBSP is also null.
- Otherwise, the RBSP contains the SODB in the following form:
  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) If 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[i] data bytes. The association of the RBSP syntax structures to the NAL units shall be as specified in Table 7-1.

NOTE - If 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. If an emulation_prevention_three_byte is present in the NAL unit, it shall be discarded.

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 Constraints on NAL unit order

This subclause specifies the requirements for NAL unit syntactical order.

Decoders conforming to this Recommendation | International Standard shall be capable of receiving NAL units in the order specified in this subclause, which is the syntax order and is also the decoding order. Constraints expressed in this Recommendation | International Standard on the decoding order of NAL units shall also constrain the syntax order of the NAL units.

Applications presenting a NAL unit stream conforming to this Recommendation | International Standard to a decoder shall either:

1) Present the NAL unit stream to the decoder in the specified syntactical order, or

2) Provide a means to indicate the specified NAL unit syntactical order for decoders that may be capable of receiving or processing some NAL units in an out-of-order fashion. No such capability is required for decoders conforming to this Recommendation | International Standard.

[Ed. Note (JVT): Change syntax order to decoding order below. Agreed.]A sequence parameter set NAL unit shall precede in syntax order all NAL units that refer to that sequence parameter set, unless the sequence parameter set NAL unit contains a duplicate copy of the previous sequence parameter set RBSP with the same value of seq_parameter_set_id.

A picture parameter set shall precede in syntax order all NAL units that refer to that picture parameter set, unless the picture parameter set NAL unit contains a duplicate copy of the previous picture parameter set RBSP with the same value of pic_parameter_set_id.

The syntax order of coded slices and data partitions of a primary coded picture shall not be interleaved with the syntax order of coded slices and data partitions of other primary coded pictures.

All coded slices and data partitions of a primary coded picture or a corresponding redundant coded picture shall precede in syntax order the coded slices and data partitions of any other coded picture that uses the primary coded picture as a reference picture for inter prediction.

Each coded slice or data partition of a primary coded picture shall precede in syntax order any slices or data partitions of a corresponding redundant coded picture containing coded data for the macroblock locations represented in the slice or data partition of the primary coded picture.

If consecutive non-reference primary coded pictures occur in the bitstream, the syntax order of the non-reference primary coded pictures shall be in ascending order of picture order count.

[Ed. Note (YKW/GJS): Can the syntax order of the slices and data partitions of one redundant coded picture be interleaved with the syntax order of the slices and data partitions of a different redundant coded picture corresponding to the same macroblocks of the same primary coded picture? (I think not.) How about for different macroblocks? (perhaps that is OK.) Can the syntax order of a slice or data partition of a redundant coded picture corresponding to a non-reference primary coded picture follow after the syntax order of the next subsequent primary coded picture with the same frame_num and (if applicable) parity?]

If arbitrary_slice_order_allowed_flag is equal to 1, slices and data partitions of a primary coded picture may follow any syntax order relative to each other. If arbitrary_slice_order_allowed_flag is equal to 0, the syntax order of slices and data partitions of a primary coded picture shall be increasing in the macroblock or macroblock pair address of the first macroblock of each slice, and if slice data partitioning is in use for a slice the data partition A of the coded slice shall precede the data partition B of the coded slice, and the data partition B of the coded slice shall precede the data partition C of the coded slice, and the data partitions A, B, and C of the coded slice shall not be interleaved in syntax order with any data partitions or non-partitioned slice data NAL units of other coded slices of the primary coded picture.

The syntax order of SEI NAL units, if present, shall precede the order of the slices and data partitions of the primary coded picture to which the SEI NAL unit corresponds, and shall be subsequent to the syntax order of all slices and data partitions of primary coded pictures that precede the primary coded picture to which the SEI NAL unit corresponds. If an SEI NAL unit contains data that pertains to more than one primary coded picture (for example, if the SEI NAL unit has a sequence as its scope), it shall be considered to correspond to the first primary coded picture in syntax order to which it pertains.

A picture delimiter, if present, shall precede in syntax order all SEI NAL units, slices and data partitions of the corresponding primary coded picture, and shall follow in syntax order after any associated SEI NAL units, slices, and data partitions of primary coded pictures that precede the corresponding primary coded picture in syntax order.

If an end of sequence NAL unit is present, the first picture that follows the end of sequence NAL unit in syntax order, if any pictures follow, shall be an IDR picture. If such an IDR picture follows, the syntax order of the end of sequence NAL unit shall precede the syntax order of any associated picture delimiter, SEI NAL units, slices, and data partitions of that IDR picture and shall be subsequent to the syntax order of all slices and data partitions of pictures that precede the IDR picture.

If an end of sequence NAL unit is present and no pictures follow the end of sequence NAL unit in syntax order, the syntax order of the end of sequence NAL unit shall be subsequent to the syntax order of all slices and data partitions of all pictures in the bitstream.

If an end of stream NAL unit is present, the syntax order of the end of stream NAL unit shall be subsequent to the syntax order of all end of sequence NAL units, slices and data partitions of all pictures in the bitstream.

The syntax order of NAL units having nal_unit_type equal to 0 or in the range of 12 to 31, inclusive, are not constrained.

7.4.1.2 Association of NAL units to primary coded pictures

Each NAL unit in the bitstream up to and including the end of stream NAL unit, if present, is associated with a primary coded picture. The NAL units associated with any primary coded picture are not interleaved in syntax order with the NAL units associated with any other primary coded picture.

All NAL units in the bitstream that precede the first slice or data partition of the first primary coded picture in the bitstream are associated with the first primary coded picture.

All NAL units in the bitstream from the first slice or data partition of a primary coded picture to the last slice or data partition of the primary coded picture, inclusive, are associated with the primary coded picture.

An end of sequence NAL unit in the bitstream is associated the preceding primary coded picture.

All NAL units in the bitstream that precede an end of stream NAL unit and follow the last slice or data partition of the primary coded picture prior to the end of sequence NAL unit are associated with the preceding primary coded picture.

An end of stream NAL unit in the bitstream is associated with the preceding primary coded picture.

All NAL units in the bitstream that precede an end of stream NAL unit and follow the last slice or data partition of the primary coded picture prior to the end of stream NAL unit are associated with the preceding primary coded picture.

All NAL units in the bitstream prior to the end of stream NAL unit, if present, that are not associated with a primary coded picture by the other association rules specified in this subclause are associated with the next primary coded picture in syntax order.

NAL units in the bitstream that follow after an end of stream NAL unit, if present, are not associated with any primary coded picture.

7.4.1.3 Association of primary coded pictures to video sequences

Each primary coded picture in the bitstream is associated with a video sequence.

The IDR picture at the beginning of the bitstream is the first picture of the first video sequence in the bitstream. Each subsequent IDR picture in the bitstream starts a subsequent video sequence.

All non-IDR pictures in the bitstream are associated with the video sequence containing the preceding IDR picture.

7.4.1.4 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, if the last byte of the RBSP data is equal to 0x00 (which can only occur if 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.

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

A sequence parameter set includes parameters that can be referred to by one or more picture parameter sets.

The sequence parameter set that is referred to in the slices of an IDR picture is called the active sequence parameter set. The content of the active sequence parameter set shall not change in any sequence parameter set NAL unit until after the last NAL unit in syntax order having nal_unit_type ranging from 1 to 6, inclusive, prior to another IDR picture. However, exact duplicate copies of the active sequence parameter set may be present in sequence parameter set NAL units in a manner interleaved with the NAL units of the sequence having nal_unit_type ranging from 1 to 6.

All picture parameter sets that are referred to by all NAL units of the sequence shall refer to the active sequence parameter set.

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.

profile_idc and **level_idc** indicate profile and level as specified in Annex A. The value of profile_idc and level_idc shall be equal to one of the values specified in Annex A for these syntax elements.

more_than_one_slice_group_allowed_flag indicates whether more than one slice group may be used in any picture of the sequence. The constraint indicated by more_than_one_slice_group_allowed_flag equal to 0 is specified in subclause 7.4.2.2.

arbitrary_slice_order_allowed_flag indicates whether arbitrary slice order may be used within pictures of the sequence. The constraint indicated by arbitrary_slice_order_allowed_flag equal to 0 on the order of slices and data partitions in pictures is specified in subclause 7.4.1.1.

redundant_pictures_allowed_flag indicates whether redundant coded pictures may be present in the sequence. The constraint indicated by redundant_pictures_allowed_flag equal to 0 on the order of slices and data partitions in pictures is specified in subclause 7.4.2.2.

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:

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

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 code picture order count (as specified in subclause 8.2.2). 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 sequence that contains two or more consecutive non-reference frames, complementary non-reference field pairs or non-paired non-reference fields in decoding order.

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 8.2.2 as follows:

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

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 always 0 and are not coded. delta_pic_order_always_zero_flag equal to 0 specifies that delta_pic_order_cnt[0] is coded and delta_pic_order_cnt[1] may be coded.

offset_for_non_ref_pic is used to calculate the picture order count of a non-reference picture as specified in 8.2.2. The value of offset_for_non_ref_pic shall be in the range from -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 8.2.2. The value of offset_for_top_to_bottom_field shall be in the range from -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 8.2.2. The value of num_ref_frames_in_pic_order_cnt_cycle shall be in the range from 0 to 255.

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 8.2.2. The value of offset_for_ref_frame[i] shall be in the range from -2^{31} to $2^{31} - 1$, inclusive.

num_ref_frames specifies the maximum total number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields 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 8.2.7.3. The value of num_ref_frames shall be in the range of 0 to 16, inclusive.

required_frame_num_update_behaviour_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 8.2.7.2.

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

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

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

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

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

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

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

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

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

frame_mbs_only_flag equal to 0 specifies that coded pictures in the sequence may either be coded fields or coded frames. frame_mbs_only_flag equal to 1 specifies that every coded picture of the 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.

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

The variables FrameHeightInMbs and FrameSizeInMbs are derived as follows

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

$$FrameSizeInMbs = PicWidthInMbs * FrameHeightInMbs \quad (7\text{-}9)$$

mb_adaptive_frame_field_flag equal to 0 specifies no switching between frame and field macroblocks within a picture, whereas 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. If frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.

vui_parameters_present_flag equal to 0 specifies that default parameter values for the VUI parameters as specified in Annex E shall be applied. vui_parameters_present_flag equal to 1 specifies that VUI parameters are present next in the bitstream. Syntax and semantics of VUI parameters is specified in Annex E.

7.4.2.2 Picture parameter set RBSP semantics

A picture parameter set includes parameters that can be referred to by multiple NAL units of type 1 to 6, inclusive. A coded picture is referred to in the text as all NAL units with types between 1 and 6, inclusive, that are decoded between two detected picture boundaries. The detection process for picture boundaries is specified in subclause 8.2.1.

The picture parameter set that is referred to in the slice of a coded picture is called the active picture parameter set. A change of the active picture parameter set is only possible after the last NAL unit of the picture is decoded. However, it is possible to decode exact copies of the active picture parameter set before or after decoding NAL units of the picture.

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. If entropy_coding_mode_flag is equal to 0 the method specified by the left descriptor is applied (Exp-Golomb coded, see subclause 9.1 or CAVLC, see subclause 9.2), whereas if entropy_coding_mode_flag is equal to 1 the method specified by the right descriptor is applied (CABAC, see subclause 9.3).

pic_order_present_flag specifies the presence of other picture order count related syntax elements in the slice header as specified in subclause 7.3.3.

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

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. slice_group_map_type shall not be equal to 3, 4, or 5 if num_slice_groups_minus1 is not equal to 1.

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 1 or if a coded picture is a field, the slice group map units are units of macroblocks.
- 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 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 a 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 from 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, top_left[i] shall be less than or equal to bottom_right[i] and bottom_right[i] shall be less than PicSizeInMapUnits.

slice_group_change_direction_flag is used with slice_group_map_type to indicate the refined mapping type of the slice_group_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:

SliceGroupChangeRate = slice_group_change_rate_minus1 + 1 (7-10)

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 if 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 replaced by l1 and list 0 replaced by list 1.

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 weighted prediction shall not 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 from -12 to +12, inclusive.

deblocking_filter_variables_present_flag specifies whether a set of variables controlling the characteristics of the deblocking filter is specified in the slice header.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows use of neighbouring inter macroblock residual data and decoded samples for the prediction of intra macroblocks, whereas constrained_intra_pred_flag equal to 1 specifies constrained intra prediction, where intra prediction only uses residual data and decoded samples from I or SI macroblock types.

redundant_pic_cnt_present_flag specifies the presence of the redundant_pic_cnt syntax element 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. If redundant_pictures_allowed_flag is equal to 0, redundant_pic_cnt_present_flag shall be equal to 0.

frame_cropping_flag equal to 1 specifies the presence of frame cropping information. If frame_cropping_flag is equal to 0, then the following default values shall be inferred frame_crop_left_offset = 0, frame_crop_right_offset = 0, frame_crop_top_offset = 0, frame_crop_bottom_offset = 0.

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 containing luma samples with horizontal coordinates from 2 * frame_crop_left_offset to $PicWidthInSamples_L$ - (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.

The value of frame_crop_left_offset shall not exceed 8 * PicWidthInMbs - (frame_crop_right_offset + 1).

The value of frame_crop_top_offset shall not exceed 8 * FrameHeightInMbs - (frame_crop_bottom_offset + 1).

[Ed. Note (GJS): frame_crop_top_offset and frame_crop_bottom_offset should be required or interpreted to be multiples of 2 if frame_mbs_only_flag is equal to 0. Otherwise no clear association of chroma lines to fields is possible. Also, it

should be noted that without field-specific cropping information, smooth movement of cropping rectangles is not really possible for interlaced video.

Move to SPS and interpret for frame_mbs_only_flag = 0 as mult of 4 for top & bottom? Agreed.]

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode 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 and size of the SEI payload and the SEI payload. SEI payload types are specified in Annex D. The SEI payload size is specified in bytes.

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 Picture delimiter RBSP semantics

The picture 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 primary coded pictures. There is no normative decoding process associated with the picture delimiter.

pic_type indicates which slice_type may be present in the picture [Ed. Note (AG): To provide small further clarification here]. Table 7-2 shows the slice_type that may occur in a picture with a given pic_type.

Table 7-2 – Meaning of pic_type

pic_type	slice_type that may be present in 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 picture in the bitstream in decoding order, if any, shall be an IDR picture. The syntax content of the SODB and RBSP for the end of sequence RBSP are null. 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 pictures 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 null. 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 primary or redundant coded picture. If arbitrary_slice_order_allowed_flag is equal to 0, within any primary or redundant coded picture the first coded slice shall have slice_id equal to 0 and the value of slice_id shall be incremented by one for each subsequent coded slice in decoding order.

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

When 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 redundant coded slices and slice data partitions. If redundant_pic_cnt is not present, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall not exceed 127.

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. If 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.

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. If 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 string of two bytes equal to 0x0000. When entropy_coding_mode_flag is equal to 1, the number of bins resulting from decoding the contents of all slice layer NAL units of a picture with nal_unit_type equal to 1, 2, 3, 4, or 5 shall not exceed (32 ÷ 3) * NumBytesInNALunitsTotal + 96 * PicSizeInMbs. NumBytesInNALunitsTotal is set to the sum of NumBytesInNALunit of all NAL units of a picture with nal_unit_type equal to 1, 2, 3, 4, or 5.

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 NumBytesInNALunitsTotal. 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

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 primary and (if present) redundant coded picture.

first_mb_in_slice specifies the address of the first macroblock in the slice. If arbitrary_slice_order_allowed_flag is equal to 0, 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.

When 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.

When 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 specifies, 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.

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

pic_parameter_set_id specifies the picture parameter set in use. All slices belonging to a picture shall have the same value of pic_parameter_set_id. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive. The value of seq_parameter_set_id in the picture parameter set referred to by pic_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referred to by the pic_parameter_set_id in the previous picture in decoding order unless nal_unit_type is equal to 5.

frame_num is used as a unique identifier for each short-term reference frame and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

If the current picture is an IDR picture, let PrecedingRefFrameNum be equal to 0. If the current picture is not an IDR picture and redundant_pic_cnt is equal to 0 for the current picture, let PrecedingRefFrameNum be equal to the value of frame_num for the previous primary reference picture in decoding order. If the current picture is not an IDR picture and redundant_pic_cnt is not equal to 0 for the current picture, let PrecedingRefFrameNum be equal to the value of frame_num for the primary reference picture that immediately precedes the previous primary reference picture in decoding order.

When the current picture is an IDR picture frame_num shall be equal to 0.

When the current picture is not an IDR picture, it shall not have frame_num equal to PrecedingRefFrameNum unless the current picture and the preceding primary reference picture are both reference fields and all of the following three conditions are true.

- the current field and the preceding field have opposite parity

- the preceding primary picture is the preceding primary reference picture
- one of the following conditions is true
  - the preceding primary coded picture is an IDR picture
  - the preceding primary coded picture includes a memory_management_control_operation syntax element equal to 5

    NOTE – If the preceding primary coded picture includes a memory_management_control_operation syntax element equal to 5, PrecedingRefFrameNum is 0.
  - there is a primary picture that precedes the preceding primary reference picture and the primary picture that precedes the preceding primary picture does not have frame_num equal to PrecedingRefFrameNum
  - there is a primary picture that precedes the preceding primary reference picture and the primary picture that precedes the preceding primary picture is not a reference picture

If required_frame_num_update_behaviour_flag is equal to 0 and frame_num is not equal to PrecedingRefFrameNum, frame_num shall be equal to (PrecedingRefFrameNum + 1) % MaxFrameNum.

If the value of frame_num is not equal to PrecedingRefFrameNum, 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 = ( PrecedingRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum  !=  frame_num )                                   (7-11)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

A picture including a memory_management_control_operation equal to 5 (reset all reference pictures) shall have frame_num constraints as described above, but after the decoding of the current picture and the processing of the memory management control operations, shall be inferred to have had frame_num equal to 0 for all subsequent use in the decoding process.

NOTE - The value of frame_num of a redundant coded picture is the same as the value of frame_num in the primary coded picture with which it is associated. 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 associated to a coded field and equal to 0 specifies that the picture is 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.

```
MbaffFrameFlag =  mb_adaptive_frame_field_flag  &&  !field_pic_flag                (7-12)
```

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

```
PicHeightInMbs = FrameHeightInMbs / ( 1 + field_pic_flag )                         (7-13)
```

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

$$PicHeightInSamples_L = PicHeightInMbs * 16 \quad (7\text{-}14)$$

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

$$PicHeightInSamples_C = PicHeightInMbs * 8 \quad (7\text{-}15)$$

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

```
PicSizeInMbs = PicWidthInMbs * PicHeightInMbs                                      (7-16)
```

bottom_field_flag equal to 1 specifies that the slice is associated to a coded bottom field. bottom_field_flag equal to 0 specifies that the picture is a coded top field. If this syntax element is not coded 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. If two consecutive primary coded pictures in decoding order are both IDR pictures, the value of idr_pic_id in the slices

of the first such IDR picture shall differ from the idr_pic_id in the second such IDR picture. 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 coded in modulo MaxPicOrderCntLsb arithmetic for the top field of a coded frame or for a coded field. An IDR picture shall have pic_order_cnt_lsb equal to 0. The size of the pic_order_cnt_lsb variable 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. The value of delta_pic_order_cnt_bottom shall be in the range of -2^{31} to 2^{31} - 1, inclusive. If this syntax element is not coded for the current slice, it is 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 8.2.2. The value of delta_pic_order_cnt[0] shall be in the range of -2^{31} to 2^{31} - 1, inclusive. If this syntax element is not coded for the current slice, it is 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 8.2.2. The value of delta_pic_order_cnt[0] shall be in the range of -2^{31} to 2^{31} - 1, inclusive. If this syntax element is not coded for the current slice, it is 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 redundant_pic_cnt shall be greater than 0 for redundant coded slices and slice data partitions. If redundant_pic_cnt is not present, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall not exceed 127.

NOTE - There should be no noticeable difference between the co-located areas of the decoded primary picture and any decoded redundant pictures.

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. If the value of nal_ref_idc in a primary picture is equal to 0, the nal_ref_idc in corresponding redundant pictures shall be equal to 0. If the value of nal_ref_idc in a primary picture is greater than 0, the nal_ref_idc in corresponding redundant picture(s) shall be greater than 0.

All slices of a redundant picture shall have the same values of field_pic_flag and bottom_field_flag as the corresponding primary picture. The frame_num of a redundant picture shall be the same as the frame_num in the primary picture, or one of the following conditions shall be true for both the redundant picture and the primary picture:

- the nal_unit_type of the picture is equal to 5, or
- the picture shall have a memory management control operation equal to 5.

The derived picture order count of a redundant picture shall be the same as for the corresponding primary picture.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to determine the prediction values of direct prediction. If direct_spatial_mv_pred_flag is equal to 1, then 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. If direct_spatial_mv_pred_flag is equal to 0, then 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.

If 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.

If 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. If field_pic_flag is equal to 0, the allowed range of num_ref_idx_l0_active_minus1 is from 0 to 15, inclusive. If field_pic_flag is equal to 1, the allowed range of num_ref_idx_l0_active_minus1 is from 0 to 31, 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. [Ed. Note: Check against SPS semantics.]

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

cabac_init_idc is only present when entropy_coding_mode_flag is equal to 1 and slice_type is not equal to I or SI. It 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:

$$SliceQP_Y = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta \quad (7\text{-}17)$$

The value of slice_qp_delta shall be limited such that QP_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. 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. If 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:

$$QS_Y = 26 + pic\_init\_qs\_minus26 + slice\_qs\_delta \quad (7\text{-}18)$$

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. If 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.

If disable_deblocking_filter_idc is equal to 0, the deblocking filter shall be applied across all edges controlled by the macroblocks within the current slice.

If disable_deblocking_filter_idc is equal to 1, the deblocking filter shall not be applied across any edges controlled by the macroblocks within the current slice.

If disable_deblocking_filter_idc is equal to 2, the deblocking filter shall be applied across all edges controlled by the macroblocks within the current slice, with the exception of the macroblock edges that are also slice boundaries.

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 C0 deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as:

$$FilterOffsetA = slice\_alpha\_c0\_offset\_div2 << 1 \quad (7\text{-}19)$$

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

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

$$FilterOffsetB = slice\_beta\_offset\_div2 << 1 \quad (7\text{-}20)$$

The value of slice_beta_offset_div2 shall be in the range of -6 to +6, inclusive. If 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 * SliceGroupChangeRate or PicSizeInMapUnits, whichever is smaller, specifies the number of slice group map units in slice group 0. The value of slice_group_change_cycle is represented in the bitstream by the following number of bits [Ed. Note: Get rid of the starting asterisk in the semantics.]

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

The minimum value of slice_group_change_cycle is 0. The maximum value of slice_group_change_cycle is

Ceil(PicSizeInMapUnits ÷ SliceGroupChangeRate). (7-22)

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, while ref_pic_list_reordering_flag_l0 equal to 0 specifies that this syntax element is not present.

If 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.

If RefPicList0[num_ref_idx_l0_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.6.2 is equal to Null, 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 8.2.6.3 is not equal to Null.

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, while ref_pic_list_reordering_flag_l1 equal to 0 specifies that this syntax element is not present.

If 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, if RefPicList0[num_ref_idx_l1_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.6.2 is equal to Null, ref_pic_list_reordering_flag_l1 shall be equal to 1 and reordering_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l1_active_minus1] in the reordered list produced as specified in subclause 8.2.6.3 is not equal to Null.

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. When reordering_of_pic_nums_idc is equal to 0, abs_diff_pic_num_minus1 shall not exceed MaxPicNum/2 - 1. When reordering_of_pic_nums_idc is equal to 1, abs_diff_pic_num_minus1 shall not exceed MaxPicNum/2-2. The allowed values of abs_diff_pic_num_minus1 are further restricted as specified in subclause 8.2.6.3.1. [Ed. remove MaxPicNum since it is local to subclause 8.2.6.3.1]

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 pair 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

[Ed.Note: We should check later subclauses to make sure that it says somewhere that if MbaffFrameFlag is equal to 1, explicit weighted prediction for field macroblocks uses the same weighting for each field of a frame or complementary reference field pair and that implicit weighted prediction uses TopFieldOrderCnt or BottomFieldOrderCnt for field MBs and FrameOrderCnt for frame MBs.]

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, while luma_weight_l0_flag equal to 0 specifies that these weighting factors are not present. If luma_weight_l0_flag is equal to 0, then luma_weight_l0[i] shall be inferred as equal to $2^{\text{luma\_log2\_weight\_denom}}$ and luma_offset_l0[i] shall be inferred as equal to 0 for RefPicList0[i].

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. Allowed range is –128 to 127, inclusive.

luma_offset_l0[i **]** is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList0[i]. Allowed range is –128 to 127, inclusive.

chroma_weight_l0_flag equal to 1 specifies that weighting factors for the Cb and Cr components of list 0 prediction are present, while chroma_weight_l0_flag equal to 0 specifies that these weighting factors are not present. If chroma_weight_l0_flag is equal to 0**,** chroma_weight_l0[i] shall be inferred as equal to $2^{\text{chroma\_log2\_weight\_denom}}$ and chroma_offset_l0[i] shall be inferred as equal to 0 for RefPicList0[i].

chroma_weight_l0[i **][** j **]** is the weighting factor applied to the Cb prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. Allowed range is –128 to 127, inclusive.

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. Allowed range is -128 to 127, inclusive.

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 and List 0 replaced by l1 and List1, respectively.

7.4.3.3 Decoded reference picture marking semantics

All syntax elements of the decoded reference picture marking shall be identical in all slice headers of a primary and (if present) redundant coded picture.

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 reference pictures marking can be modified by marking them as “unused for reference” and by assigning them to long-term frame indices. 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 shall be inferred to be equal to 2 if this syntax is in a slice header and shall be inferred to be equal to 5 if this syntax is in a decoded reference picture marking repetition SEI message as specified in Annex D.

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. [Ed.Note (GJS): Ensure the following sentences are redundant with content of Annex C: "A value of 1 specifies that all previously-decoded pictures that are stored in the decoded picture buffer are removed and not output. A value of 0 specifies that the previously-decoded pictures remain in the decoded picture buffer for output. (MH): “removed” is unclear. This paragraph has to refer to the operation of the “bumping decoder”. (GJS) The operation of this flag needs to be clarified such that in Annex C the same pictures are produced by the timed HRD as for the bumping decoder – i.e., such that the output time marked on any of these non-output pictures shall exceed the decoding time of the IDR picture AND such that the constraint on how close together the time tags are between output times *does not apply* to the non-output pictures.]

long_term_reference_flag equal to 0 specifies that the MaxLongTermFrameIdx variable is set 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 to 0 and that the current IDR picture is marked “used for long-term reference” and is assigned LongTermFrameIdx equal to 0.

adaptive_ref_pic_marking_mode_flag specifies the reference picture marking mode of the currently decoded picture and specifies how the reference picture marking is modified after the current picture is decoded. The values for adaptive_ref_pic_marking_mode_flag are specified in Table 7-5.

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.

[Ed. Note (GJS): Check to ensure that the following sentence is redundant with content of decoding process section: If memory_management_control_operation is 5 (reset all reference pictures), all frames and fields marked as “used for reference” in the decoded picture buffer shall be marked as "unused for reference". Moreover, the MaxLongTermFrameIdx variable shall be set to "no long-term frame indices".]

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" and 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".

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" and 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 5 in a slice header unless no memory_management_control_operation in the range of 1..3 is present in the same decoded reference picture marking syntax structure.

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, if 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.

If, in decoding order, the first field 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 to assign 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". 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. When decoding a frame, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs. When decoding a field, 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". The resulting picture number derived from long_term_pic_num shall be equal to a picture number assigned to one of the reference pictures marked as "used for long-term reference". When decoding a frame, 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. When decoding a field, 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. If the MaxLongTermFrameIdx variable is equal to "no long-term frame indices", long_term_frame_idx shall not be present. Otherwise, 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 allowed range of max_long_term_frame_idx_plus1 is from 0 to num_ref_frames.

7.4.4 Slice data semantics

cabac_alignment_one_bit is a bit equal to 1.

mb_skip_run is present only when entropy_coding_mode_flag is equal to 0. mb_skip_run specifies the number of consecutive macroblocks for which, when decoding a P or SP, slice mb_type is inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, and when decoding a B slice mb_type is 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 is present only when entropy_coding_mode_flag is equal to 1. mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or SP slice mb_type is inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, and when decoding a B slice mb_type is 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 and 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 following applies.

– If there is a neighbouring macroblock pair to the left 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 to the left of the current macroblock pair,

– If there is no neighbouring macroblock pair to the left in the same slice and there is a neighbouring macroblock pair above 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 above the current macroblock pair,

– Otherwise (if there is no neighbouring macroblock pair either on the left or above the current macroblock pair), 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, whereas 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 is specified in subclause 8.2.4.

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 num_mb_part(mb_type) function), the prediction mode of the macroblock (if it is not partitioned) or the first partition (given by the mb_part_pred_mode(mb_type, 0) function) and the prediction mode of the second partition (given by the mb_part_pred_mode(mb_type, 1) function). When a value is not applicable it is designated by “na”. A column of the table including only not applicable ("na") values does not appear in the table. In the text, the value of mb_type may be referred to as the macroblock type and a value X of mb_part_pred_mode 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	mb_part_pred_mode (mb_type, 0)	Intra16x16 PredMode	CodedBlock PatternChroma	CodedBlock PatternLuma
0	I_4x4	Intra_4x4	na	na	na
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 coefficient levels are present. CodedBlockPatternLuma equal to 0 specifies that there are no AC coefficient levels in the luma component of

the macroblock. CodedBlockPatternLuma equal to 15 specifies that at least one AC coefficient level is in the luma component of the macroblock, requiring scanning of AC coefficient values 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 is 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	mb_part_pred_mode (mb_type, 0)	Intra16x16 PredMode	CodedBlock PatternChroma	CodedAC PatternLuma
0	SI	Intra_4x4	na	na	na

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. The mb_type values 0 to 4 are specified in Table 7-10 and the 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	num_mb_part (mb_type)	mb_part_pred_mode (mb_type, 0)	mb_part_pred_mode (mb_type, 1)	mb_part_width (mb_type)	mb_part_height (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

The following semantics are assigned to the macroblock types in Table 7-10:

P_L0_16x16, P_L0_L0_16x8, P_L0_L0_8x16, and P_8x8: the macroblock is predicted from a previous decoded picture with luma block sizes 16x16, 16x8, 8x16, and 8x8, respectively, and the associated chroma blocks. For the macroblock types with NxM = 16x16, 16x8, and 8x16, a motion vector difference is decoded for each NxM luma block and the associated chroma blocks. If N is equal to 8 and M is equal to 8, for each sub-macroblock an additional syntax element is decoded that specifies in which type the corresponding sub-macroblock is decoded (see subclause 7.4.5.2). Depending on N, M and the sub-macroblock types there may be 1 to 16 sets of motion vector difference for a macroblock.

P_8x8ref0: same as P_8x8 but ref_idx_l0 is not present in the bitstream and set equal to 0 for all sub-macroblocks. P_8x8ref0 is only present when entropy_coding_mode_flag is equal to 0.

Pred_L0 specifies the macroblock (partition) prediction mode and 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	num_mb_part (mb_type)	mb_part_pred_mode (mb_type, 0)	mb_part_pred_mode (mb_type, 1)	mb_part_width (mb_type)	mb_part_height (mb_type)
0	B_Direct_16x16	1	Direct	na	16	16
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

The macroblock types in Table 7-11 are collectively referred to as B macroblock types.

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

B_Direct_16x16 type: no motion vector data or reference index is present in the bitstream.

B_x_y_NxM, x,y=L0,L1,Bi: each NxM luma block and the associated chroma blocks of a macroblock are predicted by using decoded motion vector differences and reference pictures. As shown in Table 7-11, for 16x16 macroblock partitions, 3 different macroblock types that differ in their prediction modes can be decoded for a macroblock. For the 16x8 and 8x16 macroblock partitions, 18 different combinations of prediction modes can be decoded for a macroblock. If a macroblock is coded as B_8x8 macroblock type, an additional codeword for each 8x8 macroblock partition, referred to as sub-macroblock, specifies the decomposition of the sub-macroblock (see Table 7-15).

B_8x8: the macroblock is partitioned into sub-macroblocks. The decoding of each sub-macroblock is specified using sub_mb_type.

Direct specifies the macroblock (partition) prediction mode and specifies that no data are present in the bitstream for the prediction process. The variables for the Inter prediction process are derived may be using list 0 or list 1 prediction. Direct is an Inter macroblock prediction mode.

Pred_L1 specifies the macroblock (partition) prediction mode and specifies that the Inter prediction process is invoked using list 1 prediction. Pred_L1 is an Inter macroblock prediction mode.

BiPred specifies the macroblock (partition) prediction mode and 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 is a sample value. pcm_byte shall not be equal to 0. The first 256 pcm_byte values represent luma sample values in raster scan order within the macroblock. The next (256*(ChromaFormatFactor-1))/2 pcm_byte values represent Cb sample values in raster scan order within the macroblock. The last (256*(ChromaFormatFactor-1))/2 pcm_byte values represent Cr sample values in raster scan order within the macroblock.

coded_block_pattern specifies which of the 6 8x8 blocks - luma and chroma - contain non-zero transform coefficients. 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                    (7-23)
```

CodedBlockPatternLuma is a number between 0 and 15 inclusive where bit n (binary representation) is 1 if 8x8 luma block n contains non-zero coefficients; otherwise, bit n is 0. The bit index n refers to the n-th luma 8x8 block in raster scan within the current macroblock.

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

Table 7-12 – Specification of CodedBlockPatternChroma values

CodedBlockPatternChroma	Description
0	All chroma coefficients are 0.
1	One or more chroma DC coefficients are non-zero. All chroma AC coefficients are 0.
2	Zero or more chroma DC coefficients are non-zero. One or more chroma AC coefficients are non-zero.

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 from -26 to +25, inclusive. mb_qp_delta shall be inferred to be equal to 0 if it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = (QP_{Y,PREV} + mb\_qp\_delta + 52) \% 52 \quad (7\text{-}24)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y, of the previous macroblock in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set to $SliceQP_Y$ derived in Equation 7-17 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 **]** indicate 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 whenever any part of the luma macroblock is intra coded. This is 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. If MbaffFrameFlag is equal to 0, the value of ref_idx_l0 shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive. If MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1, the value of ref_idx_l0 shall be in the range of 0 to 2 * num_ref_idx_l0_active_minus1 + 1.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l0, except that it is applied to reference picture list 1 and therefore all l0 or L0 extensions are replaced by l1 or L1, 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. For each motion vector variable MvL0[mbPartIdx][0][CompIdx], a prediction is formed for the horizontal and vertical components MvpL0 that is then added to mvd_l0[mbPartIdx][0][CompIdx] in order to derive the motion vector variable as specified in subclause 8.4.1. The motion vector variable MvL0[mbPartIdx][0][CompIdx] is used in inter prediction for the samples covered by the macroblock partition mbPartIdx as specified in subclause 8.4.2. The range of the components of mvd_l0 is specified by constraints on the motion vector variable values as specified in Annex A.

mvd_l1[mbPartIdx **][** 0 **][** compIdx **]** has the same semantics as mvd_l0, except that it is applied to reference picture list 1 and therefore all l0 or L0 extensions are 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 num_sub_mb_part(sub_mb_type) function) and the prediction mode of the sub-macroblock (given by the sub_mb_pred_mode(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 sub_mb_pred_mode may be referred to by "sub-macroblock prediction mode". [Ed. Note (JVT): Change function names to follow convention.]

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]	num_sub_mb_part (sub_mb_type [mbPartIdx])	sub_mb_pred_mode (sub_mb_type [mbPartIdx])	sub_mb_part_width (sub_mb_type [mbPartIdx])	sub_mb_part_height (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_XxY, X,Y=4,8 the corresponding partition of the sub-macroblock is predicted from a past picture with luma block size 8x8, 8x4, 4x8, and 4x4, respectively, and the associated chroma blocks. A motion vector is present in the bitstream for each NxM=8x8, 8x4, 4x8, and 4x4 block. Depending on N and M, up to 4 motion vector component differences may be decoded for a sub-macroblock, and thus up to 16 motion vector component differences may be decoded for a macroblock.

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]	num_sub_mb_part (sub_mb_type [mbPartIdx])	sub_mb_pred_mode (sub_mb_type [mbPartIdx])	sub_mb_part_width (sub_mb_type [mbPartIdx])	sub_mb_part_height (sub_mb_type [mbPartIdx])
0	B_Direct_8x8	1	Direct	8	8
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 sub-macroblock types in Table 7-15:

B_L0_XxY, X,Y=4,8, have the same semantics as in Table 7-14.

B_Z_X_Y, Z=L1,Bi, X,Y=4,8 [Ed.Note: add description of these mnemonic names]

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. 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

When entropy_coding_mode_flag is equal to 0, residual_block is equal to residual_block_cavlc, which is used for parsing the syntax elements for coefficient levels.

When entropy_coding_mode_flag is equal to 1, residual_block is equal to residual_block_cabac, which is used for parsing the syntax elements for coefficient levels.

Depending on mb_type, luma or chroma, residual_block(coeffLevel, maxNumCoeff) is invoked with the arguments coeffLevel and maxNumCoeff. coeffLevel is an array containing the maxNumCoeff coefficient levels that are parsed.

When mb_part_pred_mode(mb_type, 0) is equal to Intra_16x16, the coefficient levels are parsed into two arrays: Intra16x16DCLevel and Intra16x16ACLevel. Intra16x16DCLevel contains the 16 coefficient levels of the DC coefficient levels for each 4x4 luma block. Intra16x16ACLevel[i8x8 * 4 + i4x4] contains for each 4x4 luma block indexed by i8x8 * 4 + i4x4 the 15 AC coefficients.

When mb_part_pred_mode(mb_type, 0) is not equal to Intra_16x16, the 16 coefficient levels for each 4x4 luma block indexed by i8x8 * 4 + i4x4 are contained in LumaCoeffLevel[i8x8 * 4 + i4x4].

The 4 DC coefficients of all 4x4 chroma blocks per chroma component are parsed into ChromaDCLevel[iCbCr] with iCbCr indexing the chroma component.

The 15 AC coefficients for each 4x4 chroma coefficient that is indexed by i4x4 for each chroma block and iCbCr for each chroma component are contained in ChromaACLevel[iCbCr][i4x4].

7.4.5.3.1 Residual block CAVLC semantics

The function total_coeff(coeff_token) that is used in subclause 7.3.5.3.1 returns the number of coefficients derived from coeff_token.

The function trailing_ones(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 coefficients and the number of trailing one coefficients in a coefficient scan. A trailing one coefficient is one of up to three consecutive non-zero coefficients having an absolute value equal to 1 at the end of a scan of non-zero coefficients. The range of coeff_token is specified in subclause 9.2.1.

trailing_ones_sign_flag specifies the sign of a trailing one coefficient. A trailing_ones_sign_flag equal to 0 specifies that the level of the corresponding transform coefficient is +1. A trailing_ones_sign_flag equal to 1 specifies that the level of the corresponding transform coefficient is -1.

coeff_level specifies the level and sign of a non-zero transform coefficient. The range of coeff_level is specified in subclause 9.2.2.

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

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

coeffLevel contains maxNumCoeff coefficient levels for the current array.

7.4.5.3.2 Residual block CABAC semantics

coded_block_flag specifies whether the block contains non-zero transform coefficients. If coded_block_flag is equal to 0, the block contains no non-zero transform coefficients. If coded_block_flag is equal to 1, the block contains at least one non-zero transform coefficient.

significant_coeff_flag[i **]** specifies whether the transform coefficient at scanning position i is non-zero. If significant_coeff_flag[i] is equal to 0, the transform coefficient at scanning position i is equal to zero; if significant_coeff_flag[i] is equal to 1, the transform coefficient 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 coefficients for subsequent scanning positions i + 1 to maxNumCoeff - 1. If all following transform coefficients (in scanning order) of the block have value equal to zero last_significant_coeff_flag[i] is equal to 1. If last_significant_coeff_flag[i] is equal to 0, there are further non-zero transform coefficients 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 constraint by the limits in subclause 8.5.

coeff_sign_flag[i **]** specifies the sign of a transform coefficient level. When coeff_sign_flag is equal to 0, the corresponding transform coefficient level has a positive value. When coeff_sign_flag is equal to 1, the corresponding transform coefficient level has a negative value.

coeffLevel contains maxNumCoeff coefficient levels for the current array.

8 Decoding process

Outputs of this process are decoded samples of the current picture.

This clause describes the decoding process, given syntax elements and 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.

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, 3, 4 and 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, 3, 4 and 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 through 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 Detection process of coded picture boundaries

Inputs to this process are the slice header of the previous and current slice, as well as TopFieldOrderCnt and/or BottomFieldOrderCnt for the previous and current slice and the nal_ref_idc of the NAL units that contained the current and previously decoded slice. [Ed. Note (JVT): "and/or"? → "or" (for "xor" use "a or b but not both")]

Output of this process is an indication if decoding of a new picture is being started from the current slice.

Decoding of a new picture is being started from the current slice, if the slice is not a redundant coded slice and if any of the following conditions is true:

- The frame_num of the current slice is different from the frame_num of the previous slice.
- The field_pic_flag of the previous slice is different from the field_pic_flag of the current slice.
- The bottom_field_flag is present in the bitstream for both the previous slice and the current slice but is not equal in the two slices.
- The nal_ref_idc of the previous slice is different from the nal_ref_idc of the current slice with one of the nal_ref_idc values being equal to 0.
- The frame_num of the current slice is the same as the frame_num in the previous slice and pic_order_cnt_type is equal to 0, and either pic_order_cnt_lsb is different from the pic_order_cnt_lsb in the previous slice, or delta_pic_order_cnt_bottom is different from the delta_pic_order_cnt_bottom in the previous slice.
- The frame_num of the current slice is the same as the frame_num in the previous slice and pic_order_cnt_type is equal to 1, and either delta_pic_order_cnt[0] is different from the delta_pic_order_cnt[0] in the previous slice, or delta_pic_order_cnt[1] is different from the delta_pic_order_cnt[1] in the previous slice.
- Both the current slice and the previous slice belong to an IDR picture, and the idr_pic_id of the current slice differs from the idr_pic_id of the previous slice.

At least one of the above conditions shall be fulfilled at the start of each primary coded picture in the bitstream.

NOTE – The presence of a picture delimiter NAL unit after the previous slice or data partition is also a sufficient condition for detection of the start of a new primary coded picture. However, a picture delimiter NAL unit has no normative effect on the decoding process and may not be present.

8.2.2 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 8.2.6.2.3 and 8.2.6.2.4), to represent picture order differences between frames or fields for motion vector scaling in temporal direct mode (see subclause 8.4.1.2.3) and for implicit mode weighted prediction in B slices (see subclause 8.4.2.3.2).

Picture order count information is derived for every field, independent of the value of field_pic_flag, as follows:

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

TopFieldOrderCnt and BottomFieldOrderCnt indicate the output order of the corresponding decoded top field or decoded bottom field relative to the first output field of the previous IDR picture in decoding order.

TopFieldOrderCnt and BottomFieldOrderCnt are derived by invoking one of the decoding process for picture order count type 0, 1, and in subclauses 8.2.2.1, 8.2.2.2, and 8.2.2.3, respectively.

[Ed. (TW) consider removing FramePicOrder Cnt and replace its use by PicOrderCnt(CurrPic)]

In addition, for each frame or field pair or non-paired field a FrameOrderCnt is derived such that for a frame or complementary field pair,

$$FrameOrderCnt = \min(TopFieldOrderCnt, BottomFieldOrderCnt) \quad (8\text{-}1)$$

and for a non-paired top field,

$$FrameOrderCnt = TopFieldOrderCnt \quad (8\text{-}2)$$

and for a non-paired bottom field,

```
FrameOrderCnt = BottomFieldOrderCnt                                    (8-3)
```

The bitstream shall not contain data that results in a FrameOrderCnt not equal to 0 for an IDR picture. Thus, at least one of TopFieldOrderCnt and BottomFieldOrderCnt shall be equal to 0 for the fields of a frame whose FrameOrderCnt is equal to zero.

The signed picture order difference from X to Y, DiffPicOrderCnt(X, Y), between two fields or frames is specified as follows (with Z being a variable for X and Y):

```
if ( Z is a frame ) PicOrderCnt( Z ) = FrameOrderCnt of frame Z
else if ( Z is a complementary field pair )
   PicOrderCnt( Z ) = min( TopFieldOrderCnt, BottomFieldOrderCnt ) of field pair Z
else if ( Z is a top field ) PicOrderCnt( Z )                          (8-4)
else if ( Z is a bottom field ) PicOrderCnt( Z ) = BottomFieldOrderCnt of frame containing Z
```

```
DiffPicOrderCnt( X, Y ) = PicOrderCnt( X ) - PicOrderCnt( Y )          (8-5)
```

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

NOTE – Many applications assign PicOrderCnt(X) proportional to the sampling time of the picture X relative to the sampling time of an IDR picture. In this case, if X is the current picture and Y and Z are reference pictures, Y and Z are considered to be in the same temporal direction from X if both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

The bitstream shall not contain data that results in values of FrameOrderCnt, TopFieldOrderCnt, BottomFieldOrderCnt, PicOrderCntMsb, FrameNumOffset used in the decoding process as specified in subclauses 8.2.2.1 to 8.2.2.3 that exceed the range of values from -2^{31} to $2^{31}-1$, inclusive.

8.2.2.1 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.

Outputs of this process are TopFieldOrderCnt and/or BottomFieldOrderCnt.

Let prevPicOrderCntMsb be the PicOrderCntMsb of the previous reference picture in decoding order. If the current picture is an IDR picture, prevPicOrderCntMsb is set to 0.

Let prevPicOrderCntLsb be 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           (8-6)
else if( ( pic_order_cnt_lsb  >  prevPicOrderCntLsb )  &&
          ( ( pic_order_cnt_lsb – prevPicOrderCntLsb )  >  ( MaxPicOrderCntLsb / 2 ) ) )
     PicOrderCntMsb = prevPicOrderCntMsb – MaxPicOrderCntLsb
else
     PicOrderCntMsb = prevPicOrderCntMsb
```

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

```
if( field_pic_flag  = =  0  ||  bottom_field_flag  = = 0 )
   TopFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb               (8-7)
```

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

```
if( field_pic_flag  = =  0 )
   BottomFieldOrderCnt = TopFieldOrderCnt + delta_pic_order_cnt_bottom
else if( bottom_field_flag )                                           (8-8)
   BottomFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb
```

8.2.2.2 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.

Outputs of this process are TopFieldOrderCnt and/or BottomFieldOrderCnt.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are derived relative to the most recent IDR picture as specified in this subclause. Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

If the current picture is an IDR picture, then the variable idrPicFlag is set equal to 1, otherwise idrPicFlag is set equal to 0.

First, let prevFrameNumOffset be the value of FrameNumOffset of the previous picture. FrameNumOffset is derived as follows:

```
if( idrPicFlag )
    FrameNumOffset = 0
else if ( prevFrameNum > frame_num )                                    (8-9)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

Second, let absFrameNum is derived as follows:

```
if( num_ref_frames_in_pic_order_cnt_cycle  !=  0 )
    absFrameNum = FrameNumOffset + frame_num
else                                                                    (8-10)
    absFrameNum = 0
if( nal_ref_idc  = =  0  &&  absFrameNum  >  0 )
    absFrameNum = absFrameNum – 1
```

Third, if 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   (8-11)
}
```

Let expectedDeltaPerPicOrderCntCycle be derived as follows:

```
expectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    expectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]       (8-12)
```

Let expectedPicOrderCnt be 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 )                                               (8-13)
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic
```

TopFieldOrderCnt or BottomFieldOrderCnt of the current picture are derived as follows:

```
if( field_pic_flag  = =  0 ) {
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
    BottomFieldOrderCnt = TopFieldOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 1 ] (8-
14)
} else if ( bottom_field_flag  = =  0 )
```

```
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
else if ( bottom_field_flag  = =  1 )
    BottomFieldOrderCnt = expectedPicOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 0 ]
```

8.2.2.3 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 TopFieldOrderCnt or BottomFieldOrderCnt.

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

If the current picture is an IDR picture, then the variable idrPicFlag is set equal to 1, otherwise idrPicFlag is set equal to 0.

Let prevFrameNumOffset be the value of FrameNumOffset of the previous picture. FrameNumOffset is derived as follows.

```
if( idrPicFlag )
    FrameNumOffset = 0
else if ( prevFrameNum > frame_num )                                    (8-15)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

Let tempPicOrderCnt be derived as follows:

```
if ( idrPicFlag == 1)
    tempPicOrderCnt = 0
else if ( nal_ref_idc == 0 )
    tempPicOrderCnt = 2 * (FrameNumOffset + frame_num) – 1
else
    tempPicOrderCnt = 2 * (FrameNumOffset + frame_num)
```

TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

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

NOTE – Picture order count type 2 cannot be used in a sequence that contains two or more consecutive non-reference frames, complementary non-reference field pairs or non-paired non-reference fields in decoding order.

NOTE – When picture order count type 2 is used the output order is the same as the decoding order.

8.2.3 Decoding process for redundant slices

Input to this process is a redundant coded slice.

Output of this process is a decoded approximation of the area of the primary picture represented in the redundant slice.

There is no required decoding process for a redundant coded slice. If 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 redundant coded slice shall be decodable by the decoding process for a primary coded slice.

NOTE – If some of the samples in the decoded primary picture cannot be correctly decoded due to errors or losses in transmission of the sequence and if the 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. If 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.

8.2.4 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 after a new picture has been detected using the process specified in subclause 8.2.1 and the slice header of the first slice of the new picture has been parsed.

If 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 8.2.4.4-8.2.4.6. In such a case, the number of slice group map units in slice group 0 is equal to

mapUnitsInSliceGroup0 = Min(slice_group_change_cycle * SliceGroupChangeRate, PicSizeInMapUnits) (8-16)

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, this number of slice group map units in the specified growth order is allocated for slice group 0 and the remaining PicSizeInMapUnits – mapUnitsInSliceGroup0 slice group map units of the picture are allocated for slice group 1.

If 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 (8-17)

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 (8-18)

Otherwise,

- If slice_group_map_type is equal to 0, the process specified in subclause 8.2.4.1 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 1, the process specified in subclause 8.2.4.2 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 2, the process specified in subclause 8.2.4.3 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 3, the process specified in subclause 8.2.4.4 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 4, the process specified in subclause 8.2.4.5 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 5, the process specified in subclause 8.2.4.6 is invoked for the generation of mapUnitToSliceGroupMap.
- If slice_group_map_type is equal to 6, the process specified in subclause 8.2.4.7 is invoked for the generation of mapUnitToSliceGroupMap.

After generation of the mapUnitToSliceGroupMap, the process specified in subclause 8.2.4.8 is invoked to convert the map unit to slice group map mapUnitToSliceGroupMap to the macroblock to slice group map MbToSliceGroupMap. [Ed. Note (JVT): Convert to refer to “specification”, not process.]

After generation of the macroblock to slice group map as specified in subclause 8.2.4.8, 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                                                                    (8-19)
```

8.2.4.1 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                                 (8-20)
while( i < PicSizeInMapUnits )
```

8.2.4.2 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 )                  (8-21)
```

8.2.4.3 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                 (8-22)
}
```

After application of the process specified in Equation 8-22, 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.

8.2.4.4 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                              (8-23)
    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 )
}
```

8.2.4.5 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                                                                                 (8-24)
        mapUnitToSliceGroupMap[ i ] = 1 – slice_group_change_direction_flag
```

8.2.4.6 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                                                                             (8-25)
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = 1 – slice_group_change_direction_flag
```

8.2.4.7 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 ]                                    (8-26)
```

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

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

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

– 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:

$$MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i] \quad (8\text{-}27)$$

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

$$MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[i / 2] \quad (8\text{-}28)$$

– Otherwise (if 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:

$$MbToSliceGroupMap[i] = mapUnitToSliceGroupMap[(i / (2 * PicWidthInMbs)) * PicWidthInMbs + (i \% PicWidthInMbs)] \quad (8\text{-}29)$$

8.2.5 Decoding process for slice data partitioning

Inputs to this process are

– a slice data partition A layer RBSP,

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

– if syntax elements of category 4 are present in the slice data, a slice data partition C layer RBSP having the same slice_id and redundant_pic_cnt 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 clause 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, if present, contains a slice data partition B header and all syntax elements of category 3. Partition C, if 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 if 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 and redundant_pic_cnt 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 only the slice_id and redundant_pic_cnt that establishes their association with the slice data partition A of the slice.

8.2.6 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.

[Ed. Note (GJS): Check to ensure that the following deleted sentence is redundant with Annex C content: The decoded picture buffer contains frames. Each frame may contain one field or a pair of complementary fields.] 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 8.2.7. 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 8.2.7.

Subclause 8.2.6.1 specifies

- the assignment of variables FrameNum and FrameNumWrap to each of the short-term reference frames,
- the assignment of variable 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 reference picture list RefPicList1 in addition to RefPicList0 that is independent of RefPicList0.

Let long_term_entry(RefPicListX[i]) for an entry RefPicListX[i] at index i in reference picture list X where X is 0 or 1 be specified as equal to 1 if RefPicListX[i] is associated with a LongTermPicNum (for a long-term reference picture) and be specified as equal to 0 if the entry is associated with a PicNum (for a short-term reference picture).

At the beginning of decoding of each slice, the 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 8.2.6.2.
- The initial reference picture list RefPicList0 and for B slices RefPicList1 are modified as specified in subclause 8.2.6.3.

The total length of the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1+1, and for B slices the total length of 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.

8.2.6.1 Decoding process for picture numbers

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.6.2, the modification process for reference picture lists in subclause 8.2.6.3, and for the decoded reference picture marking process in subclause 8.2.7.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows. First, FrameNum is set 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                    (8-30)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-30 is the frame_num in the slice header(s) for the current picture. [Ed. Note (AG): Make sure that somewhere it is stated that for a decoded field, its two fields inherit the value of FrameNum and FrameNumWrap. And for a complementary reference field pair, the pair inherits the value of FrameNum and FrameNumWrap of its constituent fields.]

To each long-term reference picture the variable LongTermFrameIdx is assigned as specified in subclause 8.2.7.

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.

When decoding a frame,

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

```
PicNum = FrameNumWrap                                        (8-31)
```

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

```
LongTermPicNum = LongTermFrameIdx                            (8-32)
```

NOTE – When decoding a frame the value of MbaffFrameFlag has no influence on the derivations in subclauses 8.2.6.2, 8.2.6.3, and 8.2.7.

When decoding a field,

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

    PicNum = 2 * FrameNumWrap + 1 (8-33)

  - Otherwise

    PicNum = 2 * FrameNumWrap (8-34)

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

    LongTermPicNum = 2 * LongTermFrameIdx + 1 (8-35)

  - Otherwise

    LongTermPicNum = 2 * LongTermFrameIdx (8-36)

8.2.6.2 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 lists RefPicList0 and when decoding a B slice RefPicList1.

RefPicList0 and RefPicList1 have initial entries of the variables PicNum and LongTermPicNum as specified in subclauses 8.2.6.2.1 through 8.2.6.2.5.

If the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.6.2.1 through 8.2.6.2.5 exceeds 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. If the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.6.2.1 through 8.2.6.2.5 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 Null.

8.2.6.2.1 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.

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 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 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, if 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 to PicNum = 303,
- RefPicList0[1] is set to PicNum = 302,

- RefPicList0[2] is set to PicNum = 300,
- RefPicList0[3] is set to LongTermPicNum = 0, and
- RefPicList0[4] is set to LongTermPicNum = 3.

And long_term_entry(RefPicList0[i]) is equal to 0 for i equal to 0, 1, and 2; and is equal to 1 for i equal to 3 and 4.

8.2.6.2.2 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 reference picture list RefPicList0.

When decoding a field, each decoded 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 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 field marked "used for short-term reference" are included in the list of short-term reference frames refFrameList0ShortTerm. If 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 current field is included in the list refFrameList0ShortTerm. refFrameList0ShortTerm is ordered starting with the frame with the highest FrameNumWrap value and proceeding through to the frame with the lowest FrameNumWrap value.
- The LongTermFrameIdx of all frames having one or more field marked "used for long-term reference" are included in the list of long-term reference frames refFrameList0LongTerm. If 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 to the frame with the highest LongTermFrameIdx value.

The process specified in subclause 8.2.6.2.5 is invoked with refFrameList0ShortTerm and refFrameList0LongTerm given as input and the output is assigned to RefPicList0.

8.2.6.2.3 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.

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( ).

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 derived 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 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 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 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 derived 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 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 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 to the long-term reference frame or complementary reference field pair that has the highest LongTermPicNum value.
- If the reference picture list RefPicList1 has more than one entry and it 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.

8.2.6.2.4 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( ).

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

Three ordered lists of reference frames, refFrameList0ShortTerm, refFrameList1ShortTerm and refFrameListXLongTerm, are derived as follows. For purposes of the formation of these lists of frames the term reference frame refers in the following to decoded frames, complementary reference field pairs, or non-paired reference fields.

- refFrameList0ShortTerm is ordered starting with the reference frame frm0 with the largest value of PicOrderCnt(frm0) less than the value of PicOrderCnt(CurrPic) of the current field and proceeding through to the short-term reference frame frm1 that has the smallest value of PicOrderCnt(frm1), and then continuing with the reference frame frm2 with the smallest value of PicOrderCnt(frm2) greater than the value of PicOrderCnt(CurrPic) of the current field and proceeding through to the short-term reference frame frm3 that has the largest value of PicOrderCnt(frm3). When for the current field nal_ref_idc is greater than 0 and the current coded field follows in decoding order a coded field fld1 with which together it forms a complementary reference field pair after decoding, fld1 shall be included into the list refFrameList0ShortTerm using PicOrderCnt(fld1) and the ordering method described in the previous sentence shall be applied.
- refFrameList1ShortTerm is ordered starting with the reference frame frm4 with the smallest value of PicOrderCnt(frm4) greater than the value of PicOrderCnt(CurrPic) of the current field and proceeding through to the short-term reference frame frm5 that has the largest value of PicOrderCnt(frm5), and then continuing with the frame picture frm6 with the largest value of PicOrderCnt(frm6) less than the value of PicOrderCnt(CurrPic) of the current field and proceeding through to the short-term reference frame frm7 that has the smallest value of PicOrderCnt(frm7). When for the current field nal_ref_idc is greater than 0 and the current coded field follows in decoding order a coded field fld2 with which together it forms a complementary reference field pair after decoding, fld2 shall be included into the list refFrameList1ShortTerm using PicOrderCnt(fld2) and the ordering method described in the previous sentence shall be applied.
- refFrameListXLongTerm is ordered starting with the frame having the lowest LongTermFrameIdx value and proceeding to the frame having highest LongTermPicNum value. The current frame (with only the opposite-parity field marked "used for reference") is included into the list refFrameListXLongTerm when the complementary field of the current picture is stored as long-term picture and marked "used for reference". Frames in which only one field is stored as long-term picture and marked as “used for reference” are included into the list refFrameListXLongTerm.

The process specified in subclause 8.2.6.2.5 is invoked with refFrameList0ShortTerm and refFrameListXLongTerm given as input and the output is assigned to RefPicList0.

The process specified in subclause 8.2.6.2.5 is invoked with refFrameList1ShortTerm and refFrameListXLongTerm given as input and the output is assigned to RefPicList1.

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

8.2.6.2.5 Initialisation process for reference picture lists in fields

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

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

8.2.6.3 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.

If 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 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 8.2.6.3.1 is invoked with RefPicList0 and refIdxL0 given as input, and the output is assigned to RefPicList0 and refIdxL0.
  - If reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.6.3.2 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, and the reordering process for reference picture list RefPicList0 is finished.

If 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 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 8.2.6.3.1 is invoked with RefPicList1 and refIdxL1 given as input, and the output is assigned to RefPicList1 and refIdxL1.
  - If reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.6.3.2 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, and the reordering process for reference picture list RefPicList1 is finished.

8.2.6.3.1 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 (with X being 0 or 1) and the incremented index refIdxLX.

If the current picture is a coded frame, let maxPicNum be equal to MaxFrameNum. Otherwise, let maxPicNum be equal to 2*MaxFrameNum.

Let currPicNum be a variable for identifying the current picture. If the current picture is a coded frame, it is set to currPicNum = frame_num. Otherwise, it is set currPicNum = 2 * frame_num.

Let picNumLXNoWrap be an auxiliary variable only used inside this subclause, it is specified 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          (8-37)
else
      picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 )
```

- If 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          (8-38)
else
      picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. If 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 to currPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

Let picNumLX be the variable PicNum of the short-term reference picture to be inserted into the reference picture list RefPicListX at index refIdxLX, it is specified as follows

```
if( picNumLXNoWrap > currPicNum )
      picNumLX = picNumLXNoWrap – maxPicNum                                              (8-39)
else
      picNumLX = picNumLXNoWrap
```

picNumLX shall specify a reference picture that is marked as “used for short-term reference” and shall not specify a short-term reference picture that is marked as "non-existing".

The following procedure shall then 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++ ] = picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1+1; cIdx++ )                   (8-40)
    if( long_term_entry( RefPicListX[ cIdx ] ) || RefPicListX[ cIdx ] != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

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.

8.2.6.3.2 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 (with X being 0 or 1) and the incremented index refIdxLX.

LongTermPicNum equal to long_term_pic_num shall specify a reference picture that is marked as "used for long-term reference".

The following procedure shall then 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++ ] = LongTermPicNum                                    (8-41)
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1+1; cIdx++ )
    if( !long_term_entry( RefPicListX[ cIdx ] ) || RefPicListX[ cIdx ] != LongTermPicNum )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

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.

8.2.7 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

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, if a field, by its parity. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, if a field, by its parity.

Frames or complementary reference 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 or one of its constituent fields is marked as “unused for reference”. Fields 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 8.2.7.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.7.4.

A short-term reference picture is identified for use in the decoding process by 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. Subclause 8.2.6.1 specifies how PicNum and LongTermPicNum are calculated.

8.2.7.1 Sequence of operations for decoded reference picture marking process

The sequence of operations for decoded reference picture marking is as follows.

Step 1:

If frame_num of the current picture is not equal to PrecedingRefFrameNum and is not equal to (PrecedingRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num is performed according to subclause 8.2.7.2.

Step 2:

The current picture is decoded.

Step 3:

- If the current picture is an IDR picture, the following applies.
    - All reference pictures shall be marked as "unused for reference"
    - 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 to “no long-term frame indices”.
    - Otherwise (if 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 to 0, and MaxLongTermFrameIdx shall be set to 0.

- Otherwise, the following applies.
  - If adaptive_ref_pic_marking_mode_flag is equal to 0, the process specified in subclause 8.2.7.3 is invoked.
  - Otherwise (if adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.7.4 is invoked.

Step 4:

If the current picture 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 num_ref_frames.

8.2.7.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrecedingRefFrameNum and is not equal to (PrecedingRefFrameNum + 1) % MaxFrameNum.

NOTE – This process can only be invoked for a conforming bitstream if required_frame_num_update_behaviour_flag is equal to 1. If required_frame_num_update_behaviour_flag is equal to 0 and frame_num is not equal to PrecedingRefFrameNum and is not equal to (PrecedingRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

If this process is invoked, a set of values of frame_num pertaining to “non-existing” pictures is derived as all values taken on by UnusedShortTermFrameNum in Equation 7-11 except the value of frame_num for the current picture.

The decoding process shall mark a frame for each values of frame_num pertaining to “non-existing” pictures, in the order in which the values of UnusedShortTermFrameNum are generated by Equation 7-11, using the “sliding window” picture marking process as specified in subclause 8.2.7.3. The added frames shall also be marked as “non-existing” and “used for short-term reference”. The sample values of the added frames may be set to any value. These added frames 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 8.2.6.3.1), and shall not be referred to in the assignment process of a LongTermFrameIdx to a short-term picture (subclause 8.2.7.4.3). When a frame marked as “non-existing” is marked as “unused for reference” using the “sliding window” buffering process or the “adaptive memory control” mechanism, it shall no longer be marked as “non-existing”.

NOTE - The decoding process should infer an unintentional picture loss if any of these values of frame_num pertaining to “non-existing” pictures is referred to in inter prediction or is referred to in the assignment process for reference indices. The decoding process should not infer an unintentional picture loss if a memory management control operation is applied to a frame marked as “non-existing”.

8.2.7.3 Sliding window decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 0.

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”.
- If (numShortTerm + numLongTerm) is equal to num_ref_frames, numShortTerm shall be greater than 0, and the short-term reference frame, complementary reference field pair or non-paired reference field that was decoded first (it has the smallest value of FrameNumWrap) is marked as “unused for reference”. If it is a frame or a complementary field pair, both of its fields are also marked as “unused for reference”.

8.2.7.4 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 8.2.7.4.1 to 8.2.7.4.6 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.

When decoding a frame, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.

When decoding a field, memory_management_control_operation commands are applied to the individual reference fields specified.

8.2.7.4.1 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 currPicNum be the current picture number. If the current picture is a coded frame, currPicNum = frame_num. Otherwise, currPicNum = 2 * frame_num.

Let picNumX be specified by

$$picNumX = currPicNum - (difference\_of\_pic\_nums\_minus1 + 1). \quad (8\text{-}42)$$

If the current picture is a frame, 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”.

If the current picture is a field, the short-term reference field specified by picNumX is marked as “unused for reference”. If that reference field is part of a reference frame or a complementary reference field pair, the reference frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.7.4.2 Marking process of a long-term picture as “unused for reference”

This process is invoked when memory_management_control_operation is equal to 2.

If the current picture is a frame, 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”.

If the current picture is a field, the long-term reference field specified by LongTermPicNum equal to long_term_pic_num is marked as “unused for reference”. If that reference field is part of a reference frame or a complementary reference field pair, the reference frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.7.4.3 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 8.2.7.4.1. 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".

If 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". If 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”.

If the current picture is a frame, 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.

If the current picture is a field, 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.

8.2.7.4.4 Decoding process for MaxLongTermFrameIdx

This process is invoked when memory_management_control_operation is equal to 4.

If max_long_term_frame_idx_plus1 is smaller than MaxLongTermFrameIdx+1, all pictures marked as "used for long-term reference" and assigned LongTermFrameIdx greater than max_long_term_frame_idx_plus1 – 1 shall be marked as “unused for reference”.

If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx shall be set to “no long-term frame indices”. Otherwise, MaxLongTermFrameIdx shall be set 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.

8.2.7.4.5 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 to “no long-term frame indices”.

8.2.7.4.6 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.

If LongTermFrameIdx 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". If 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. If the current picture is a frame, then both its fields are also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx. If the current picture is a second (in decoding order) field of a complementary reference field pair, then the pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.3 Intra prediction process

This process is invoked for I 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 the Intra prediction samples of components of the macroblock or in case of the Intra_4x4 prediction process for luma samples the output are 4x4 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock.

The decoding processes for Intra prediction modes are described for the luma component if the macroblock prediction mode is equal to Intra_4x4 in subclause 8.3.1 otherwise (Intra_16x16) in subclause 8.3.2.

The decoding processes for Intra prediction modes for the chroma component 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 the 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 the 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 $pred4x4_L(x, y)$ 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).

$$pred_L(xO + x, yO + y) = pred4x4_L(x, y) \quad (8\text{-}43)$$

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 prior (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.

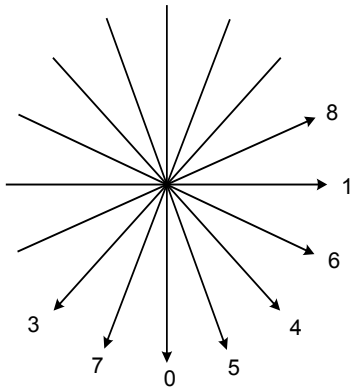


Figure 8-1 – Intra_4x4 prediction mode directions (informative)

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.
- If mbAddrA is not available or mbAddrA is not coded in Intra_4x4 prediction mode, intra4x4PredModeA and intra4x4PredModeB are set to 2 (Intra_4x4_DC prediction mode). Otherwise, intra4x4PredModeA is set to Intra4x4PredMode[luma4x4BlkIdxA], where Intra4x4PredMode is the variable assigned to the macroblock mbAddrA.
- If mbAddrB is not available or mbAddrB is not coded in Intra_4x4 prediction mode, intra4x4PredModeA and intra4x4PredModeB are set to 2 (Intra_4x4_DC prediction mode). Otherwise, intra4x4PredModeB is set to Intra4x4PredMode[luma4x4BlkIdxB], where Intra4x4PredMode is the variable assigned to the macroblock mbAddrB.
- Intra4x4PredMode[luma4x4BlkIdx] is derived by applying the following procedure.

```
predIntra4x4PredMode = Min( intra4x4PredModeA, intra4x4PredModeB )
if ( prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )
    IntraLuma4x4PredMode[ luma4x4BlkIdx ] = predIntra4x4PredMode
else                                                                    (8-44)
    if ( rem_intra4x4_pred_mode[ luma4x4BlkIdx ] < predIntra4x4PredMode )
        IntraLuma4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ]
    else
        IntraLuma4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ] + 1
```

8.3.1.2 Intra_4x4 sample prediction

This process is invoked for each of 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 $pred4x4_L(x, y)$, 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 deblocking, with x = -1, y = -1..3 and x = 0..7, y = -1, are derived as follows.

– The luma location (xN, yN) is specified by

$$xN = xO + x \tag{8-45}$$

$$yN = yO + y \tag{8-46}$$

– 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.

– 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.
- 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 following applies

- 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 if 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 reconstructed 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 if 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 as follows:

$$pred4x4_L(x, y) = p(x, -1), \text{ with } x, y = 0..3 \tag{8-47}$$

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 if p(-1, y) , 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 as follows:

$$pred4x4_L(x, y) = p(-1, y), \text{ with } x,y = 0..3 \tag{8-48}$$

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 as follows:

$$pred4x4_L(x, y) = (p(0, -1) + p(1, -1) + p(2, -1) + p(3, -1) + p(-1, 0) + p(-1, 1) + p(-1, 2) + p(-1, 3) + 4) >> 3 \tag{8-49}$$

- If samples p(x, -1), with x = 0..3 are marked as "not available for Intra_4x4 prediction" 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 as follows:

$$pred4x4_L(x, y) = (p(-1, 0) + p(-1, 1) + p(-1, 2) + p(-1, 3) + 2) >> 2 \tag{8-50}$$

- If samples p(-1, y), with y = 0..3 are marked as "not available for Intra_4x4 prediction" and 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 as follows:

$$pred4x4_L(x, y) = (p(0, -1) + p(1, -1) + p(2, -1) + p(3, -1) + 2) >> 2 \quad (8\text{-}51)$$

- Otherwise, the values of the prediction samples $pred4x4_L$(x, y), with x, y = 0..3 are derived as follows:

$$pred4x4_L(x, y) = 128 \quad (8\text{-}52)$$

NOTE – A 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 if all 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:

For x = 3 and y = 3,

$$pred4x4_L(x, y) = (p(6, -1) + 3 * p(7, -1) + 2) >> 2 \quad (8\text{-}53)$$

Otherwise,

$$pred4x4_L(x, y) = (p(x + y, -1) + 2 * p(x + y + 1, -1) + p(x + y + 2, -1) + 2) >> 2 \quad (8\text{-}54)$$

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 if all 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:

For x > y,

$$pred4x4_L(x, y) = (p(x - y - 2, -1) + 2 * p(x - y - 1, -1) + p(x - y, -1) + 2) >> 2 \quad (8\text{-}55)$$

For x < y,

$$pred4x4_L(x, y) = (p(-1, y - x - 2) + 2 * p(-1, y - x - 1) + p(-1, y - x) + 2) >> 2 \quad (8\text{-}56)$$

Otherwise,

$$pred4x4_L(x, y) = (p(0, -1) + 2 * p(-1, -1) + p(-1, 0) + 2) >> 2 \quad (8\text{-}57)$$

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 if all 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:

For (2 * x – y) = 0, 2, 4, 6,

$$pred4x4_L(x, y) = (p(x - (y >> 1) - 1, -1) + p(x - (y >> 1), -1) + 1) >> 1 \quad (8\text{-}58)$$

For (2 * x – y) = 1, 3, 5,

$$pred4x4_L(x, y) = (p(x - (y >> 1) - 2, -1) + 2 * p(x - (y >> 1) - 1, -1) + p(x - (y >> 1), -1) + 2) >> 2 \quad (8\text{-}59)$$

For (2 * x – y) = -1,

$$pred4x4_L(x, y) = (p(-1, 0) + 2 * p(-1, -1) + p(0, -1) + 2) >> 2 \quad (8\text{-}60)$$

Otherwise

$$pred4x4_L(x, y) = (p(-1, y - 1) + 2 * p(-1, y - 2) + p(-1, y - 3) + 2) >> 2 \quad (8\text{-}61)$$

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 if all 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:

For (2 * y - x) = 0, 2, 4, 6,

$$pred4x4_L(x, y) = (p(-1, y -(x >> 1) - 1) + p(-1, y - (x >> 1)) + 1) >> 1 \quad (8\text{-}62)$$

For (2 * y - x) = 1, 3, 5,

$$pred4x4_L(x, y) = (p(-1, y - (x >> 1) - 2) + 2 * p(-1, y - (x >> 1) - 1) + p(-1, y - (x >> 1)) + 2) >> 2 \quad (8\text{-}63)$$

For (2 * y - x) = -1,

$$pred4x4_L(x, y) = (p(-1, 0) + 2 * p(-1, -1) + p(0, -1) + 2) >> 2 \quad (8\text{-}64)$$

Otherwise

$$pred4x4_L(x, y) = (p(x - 1, -1) + 2 * p(x - 2, -1) + p(x - 3, -1) + 2) >> 2 \quad (8\text{-}65)$$

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 if all 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:

For y = 0 or y = 2,

$$pred4x4_L(x, y) = (p(x + (y >> 1), -1) + p(x + (y >> 1) + 1, -1) + 1) >> 1 \quad (8\text{-}66)$$

Otherwise,

$$pred4x4_L(x, y) = (p(x + (y >> 1), -1) + 2 * p(x + (y >> 1) + 1, -1) + p(x + (y >> 1) + 2, -1) + 2) >> 2 \quad (8\text{-}67)$$

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 if 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 as follows:

For (x + 2 * y) = 0, 2, 4

$$pred4x4_L(x, y) = (p(-1, y + (x >> 1)) + p(-1, y + (x >> 1) + 1) + 1) >> 1 \quad (8\text{-}68)$$

For (x + 2 * y) = 1, 3

$$pred4x4_L(x, y) = (p(-1, y + (x >> 1)) + 2 * p(-1, y + (x >> 1) + 1) + p(-1, y + (x >> 1) + 2) + 2) >> 2 \quad (8\text{-}69)$$

For (x + 2 * y) = 5,

$$pred4x4_L(x, y) = (p(-1, 2) + 3 * p(-1, 3) + 2) >> 2 \quad (8\text{-}70)$$

For (x + 2 * y) > 5,

$$pred4x4_L(x, y) = p(-1, 3) \quad (8\text{-}71)$$

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 $pred_L(x, y)$.

The 33 neighbouring samples p(x, y) that are constructed luma samples prior to deblocking, with x = -1, y = -1..15 and 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.

– 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.

– Otherwise, the sample p(x, y) is marked as “available for Intra_16x16 prediction” and the following applies

  - The luma sample at luma location (xW, yW) inside the macroblock mbAddrN is assigned to p(x, y).

Let $pred_L(x, y)$ with x, y = 0..15 denote the prediction 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 mode shall be used only if all neighbouring samples p(x, -1) , with x = 0..15 are marked as “available for Intra_16x16 prediction”.

$$pred_L(x, y) = p(x, -1), \text{ with } x, y = 0..15 \quad (8\text{-}72)$$

8.3.2.2 Specification of Intra_16x16_Horizontal prediction mode

This mode shall be used only if all neighbouring samples p(-1, y) , y = 0..15 are marked as “available for Intra_16x16 prediction”.

$$pred_L(x, y) = p(-1, y), \text{ with } x, y = 0..15 \quad (8\text{-}73)$$

8.3.2.3 Specification of Intra_16x16_DC prediction mode

If all neighbouring samples p(x', -1) and p(-1, y') used in Equation 8-74 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L(x, y) = (\sum_{x'=0}^{15} p(x',-1) + \sum_{y'=0}^{15} p(-1,y') + 16) >> 5 \text{ with } x, y = 0..15 \quad (8\text{-}74)$$

If the neighbouring samples p(x', -1) are not available and the neighbouring samples p(-1, y') are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L(x, y) = (\sum_{y'=0}^{15} p(-1,y') + 8) >> 4 \text{ with } x, y = 0..15 \quad (8\text{-}75)$$

If the neighbouring samples p(-1, y') are not available and the neighbouring samples p(x', -1) are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L(x, y) = (\sum_{x'=0}^{15} p(x',-1) + 8) >> 4 \text{ with } x, y = 0..15 \quad (8\text{-}76)$$

If none of the neighbouring samples p(x', -1) and p(-1, y') are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock shall be

$$pred_L(x, y) = 128 \text{ with } x, y = 0..15 \quad (8\text{-}77)$$

8.3.2.4 Specification of Intra_16x16_Plane prediction mode

This mode shall be used only if all neighbouring samples p(x, -1) with x = -1..15 and p(-1, y) with y = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L(x, y) = Clip1((a + b * (x - 7) + c * (y - 7) + 16) >> 5), \quad (8\text{-}78)$$

where:

$$a = 16 * (p(-1, 15) + p(15, -1)) \quad (8\text{-}79)$$

$$b = (5 * H + 32) >> 6 \quad (8\text{-}80)$$

$$c = (5 * V + 32) >> 6 \quad (8\text{-}81)$$

and H and V are specified in Equations 8-82 and 8-83.

$$H = \sum_{x'=0}^{7} (x'+1) * (p(8 + x', -1) - p(6 - x', -1)) \quad (8\text{-}82)$$

$$V = \sum_{y'=0}^{7} (y'+1) * (p(-1, 8 + y') - p(-1, 6 - y')) \quad (8\text{-}83)$$

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 $pred_{Cb}(x, y)$ and $pred_{Cr}(x, y)$.

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 subcase C is used as a replacement of the subcases Cb or Cr.

The 17 neighbouring samples p(x, y) that are constructed chroma samples prior to deblocking, with x = -1, y = -1..7 and 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.

– 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.

– Otherwise, the sample p(x, y) is marked as "available for Intra chroma prediction" and the following applies

  - 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

[Ed.(TW) Double-Check clarity of this subclause]

The block of chroma samples is predicted in four separate 4x4 sub-blocks, labelled a-d as shown in Figure 8-2.

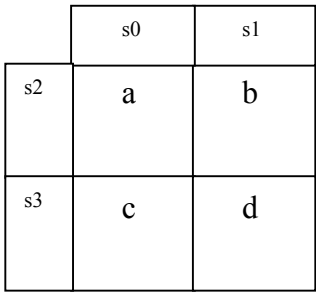


Figure 8-2 – Chroma 4x4 blocks a, b, c, d, and predictors s0, s1, s2, s3.

In Figure 8-2, each of s0, s1, s2, and s3 represents the sum of the four reconstructed samples bordering the chroma block at the location shown. The predictions for all chroma samples within a block (a, b, c, or d) are dependent upon the availability of the samples used to compute s0, s1, s2, and s3, and are given in Table 8-5. s0, s1, s2, and s3 may be unavailable by being outside the picture or slice or because constrained intra prediction is in use and the neighbouring blocks are not intra coded. In Table 8-5 block predictors are given by the left-most formula for which the sums used in their calculation are available. For example, with s0 and s3 available, but s1 and s2 not available, a, b, c, and d are predicted by the formulas in their pred1, pred3, pred0, and pred2 columns, respectively.

Table 8-5 – Specification for Intra_Chroma_DC prediction mode

Block	pred0	pred1	pred2	pred3
a	(s0 + s2 + 4) >> 3	(s0 + 2) >> 2	(s2 + 2) >> 2	128
b	(s1 + 2) >> 2	(s1 + 2) >> 2	(s2 + 2) >> 2	128
c	(s3 + 2) >> 2	(s0 + 2) >> 2	(s3 + 2) >> 2	128
d	(s1 + s3 + 4) >> 3	(s1 + 2) >> 2	(s3 + 2) >> 2	128

8.3.3.2 Specification of Intra_Chroma_Horizontal prediction mode

This mode shall be used only if all neighbouring 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.

$$pred_C(x, y) = p(-1, y), \text{ with } x, y = 0..7 \quad (8\text{-}84)$$

8.3.3.3 Specification of Intra_Chroma_Vertical prediction mode

This mode shall be used only if all neighbouring 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.

$$pred_C(x, y) = p(x, -1), \text{ with } x, y = 0..7 \quad (8\text{-}85)$$

8.3.3.4 Specification of Intra_Chroma_Plane prediction mode

This mode shall be used only if all neighbouring 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.

$$pred_C(x, y) = Clip1((a + b * (x - 3) + c * (y - 3) + 16) >> 5), \text{ with } x, y = 0..7 \quad (8\text{-}86)$$

where:

$$a = 16 * (p(-1, 7) + p(7, -1)) \quad (8\text{-}87)$$

$$b = (17 * H + 16) >> 5 \quad (8\text{-}88)$$

$$c = (17 * V + 16) >> 5 \quad (8\text{-}89)$$

and H and V are specified as follows.

$$H = \sum_{x=0}^{3} (x+1) \cdot [p(4+x,-1) - p(2-x,-1)] \quad (8\text{-}90)$$

[Ed. Change dot to asterisk]

$$V = \sum_{y=0}^{3} (y+1) \cdot [p(-1,4+y) - p(-1,2-y)] \quad (8\text{-}91)$$

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 to 0.

The following steps are specified for each macroblock partition or for each sub-macroblock partition.

The functions mb_part_width( ), mb_part_height( ), sub_mb_part_width( ), sub_mb_part_height( ) describing the width and height of macroblock partitions and sub-macroblock partitions 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 = mb_part_width(mb_type) (8-92)

  partHeight = mb_part_height(mb_type) (8-93)

- Otherwise,

  partWidth = sub_mb_part_width(sub_mb_type[mbPartIdx]) (8-94)

  partHeight = sub_mb_part_height(sub_mb_type[mbPartIdx]). (8-95)

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 (8-96)

partHeight = 4 (8-97)

with mbPartIdx proceeding over values 0..3 and for each sub-macroblock indexed by mbPartIdx, subMbPartIdx proceeds over values 0..3.

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

2. 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 derived later in the decoding process, the following assignments are made:

MvL0[mbPartIdx][subMbPartIdx][compIdx] = mvL0[compIdx] (8-98)

MvL1[mbPartIdx][subMbPartIdx][compIdx] = mvL1[compIdx] (8-99)

RefIdxL0[mbPartIdx] = refIdxL0 (8-100)

RefIdxL1[mbPartIdx] = refIdxL1 (8-101)

[Ed. Note (LoWi): double-check whether the last 4 equations are redundant.]

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 partitition prediction samples in their correct relative positions in the macroblock, as follows.

for x = 0..partWidth – 1, y = 0 .. partHeight – 1

$pred_L(xP + xS + x, yP + yS + y) = predPart_L(x, y)$

for x = 0..partWidth/2 – 1, y = 0 .. partHeight/2 – 1, and C being replaced by either Cb or Cr

$$pred_C(xP/2 + xS/2 + x, yP/2 + yS/2 + y) = predPart_C(x, y)$$

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 and the chroma motion vectors mvCL0 and mvCL1
- reference indices refIdxL0 and refIdxL1
- prediction list utilization flags predFlagL0 and predFlagL1

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 to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set to 0.
- If mb_type is equal to B_Skip, or B_Direct_16x16 or sub_mb_type[subMbPartIdx] is equal to B_Direct_8x8, the following applies. 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, 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, the following is specified. predFlagLX is initially set to 0. When mb_part_pred_mode(mbPartIdx) is equal to Pred_LX or BiPred,

  refIdxLX = ref_idx_lX[mbPartIdx] (8-102)

  predFlagLX = 1 (8-103)

  and the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, and list suffix LX as the input and the output being mvpLX. The luma motion vectors are derived as follows.

  mvLX[0] = mvpLX[0] + mvd_lX[mbPartIdx][subMbPartIdx][0] (8-104)

  mvLX[1] = mvpLX[1] + mvd_lX[mbPartIdx][subMbPartIdx][1] (8-105)

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 set equal to zero:

refIdxL0 = 0. (8-106)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

- The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set to 0, subMbPartIdx set to 0, and list suffix L0 as input and the output is assigned the mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.
- If at least one of the following conditions is true, both components of the motion vector mvL0 are set to zero.

- 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, with mb_part_width(mb_type) = 16 and mb_part_height(mb_type) = 16, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, 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, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The derivation process depends on the value of direct_spatial_mv_pred_flag in the slice header syntax as specified in subclause 7.3.3. 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, it 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.

When direct_spatial_mv_pred_flag is equal to 1, the spatial direct motion vector and reference picture index prediction mode specified in subclause 8.4.1.2.2 is used.

When direct_spatial_mv_pred_flag is equal to 0, the temporal direct motion vector and reference picture index prediction mode specified in subclause 8.4.1.2.3 is used.

8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions

Inputs to this process are mbPartIdx, 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 coded frame or a pair of complementary fields let firstRefPicL1Top and firstRefPicL1Bottom be the top and bottom fields of firstRefPicL1 and the following variables are specified as

topAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Top, CurrPic)) (8-107)

bottomAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Bottom, CurrPic)) (8-108)

colPic is the picture that contains the co-located macroblock. It is specified in Table 8-6.

Table 8-6 – Specification of colPic [Ed. Note (GJS): double-check use of term "pair of complementary fields"]

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

If direct_8x8_inference_flag is equal to 1, the motion vectors associated to the outside macroblock corner 4x4 sub-macroblock partition are used for all 4x4 sub-macroblock partitions of a macroblock partition. In this case, subMbPartIdx is set to

subMbPartIdx = mbPartIdx (8-109)

Let pic_coding_struct(X) be a function with the argument X being either CurrPic or colPic. It is specified in Table 8-7.

Table 8-7 – Specification of pic_coding_struct(X)

X is coded with field_pic_flag equal to …	mb_adaptive_frame_field_flag	pic_coding_struct(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 (x, y) assigned to (xCol, yCol) as the output.

Table 8-8 specifies the co-located macroblock address mbAddrCol, yM, and the variable vertMvScale in two steps:

1. Specification of a macroblock address mbAddrX depending on pic_coding_struct(CurrPic), and pic_coding_struct(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 following variable

   - fieldDecodingFlagX is set to 1 if the macroblock mbAddrX in the picture colPic is coded in field decoding mode, otherwise it is set to 0.

Unspecified values in Table 8-8 indicate that the value of the corresponding variable is not relevant for the current table row.

mbAddrCol is set to CurrMbAddr or to one of the following values.

mbAddrCol1 = 2 * PicWidthInMbs * (CurrMbAddr / PicWidthInMbs) +
(CurrMbAddr % PicWidthInMbs) + PicWidthInMbs * (yCol / 8) (8-110)

mbAddrCol2 = 2 * CurrMbAddr + (yCol / 8) (8-111)

mbAddrCol3 = 2 * CurrMbAddr + bottom_field_flag (8-112)

mbAddrCol4 = PicWidthInMbs * (CurrMbAddr / (2 * PicWidthInMbs)) +
(CurrMbAddr % PicWidthInMbs) (8-113)

mbAddrCol5 = CurrMbAddr / 2 (8-114)

mbAddrCol6 = 2 * (CurrMbAddr / 2) + ((topAbsDiffPOC < bottomAbsDiffPOC) ? 0 : 1) (8-115)

mbAddrCol7 = 2 * (CurrMbAddr / 2) + (yCol / 8) (8-116)

Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale

pic_coding_struct(CurrPic)	pic_coding_struct(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.

Let predFlagL0Col and predFlagL1Col be the prediction utilization flags predFlagL0 and predFlagL1 that are assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol 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 mode or both prediction utilization flags, predFlagL0Col and predFlagL1Col are equal to 0, both components of mvCol are set to 0 and refIdxCol is set to –1.
- Otherwise, the following applies.

- If predFlagL0Col is equal to 1, the motion vector mvL0 and the reference index refIdxL0 of the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic are assigned to mvCol and refIdxCol.
- Otherwise (predFlagL0Col is equal to 0 and predFlagL1Col is equal to 1), the motion vector mvL1 and the reference index refIdxL1 of the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic are assigned to mvCol and refIdxCol.

8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode

This process is invoked if the direct_spatial_mv_pred_flag is equal to 1 and 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, subMbPartIdx.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, 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. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, and list suffix L0 as input and the output is assigned to the motion vectors mvL0N and the reference indices refIdxL0N with N being either A, B, or C.
2. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, and list suffix L1 as input and the output is assigned to the motion vectors mvL1N and the reference indices refIdxL1N with N being either A, B, or C.
3. The reference indices refIdxL0, refIdxL1 and directZeroPredictionFlag are derived by

refIdxL0 = MinPositive(refIdxL0A, MinPositive(refIdxL0B, refIdxL0C)) (8-117)
refIdxL1 = MinPositive(refIdxL1A, MinPositive(refIdxL1B, refIdxL1C)) (8-118)
directZeroPredictionFlag = 0 (8-119)

where

$$\text{MinPositive}(x, y) = \begin{cases} \text{Min}(x,y) & \text{if } x >= 0 \text{ and } y >= 0 \\ \text{Max}(x,y) & \text{otherwise} \end{cases} \quad (8\text{-}120)$$

4. If both reference indices refIdxL0 and refIdxL1 are less than 0, then

refIdxL0 = 0 (8-121)
refIdxL1 = 0 (8-122)
directZeroPredictionFlag = 1 (8-123)

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.

Let colZeroFlag be a flag that is set to 1 if the reference picture in reference index list RefPicList1 referred by refIdxL1 is a short-term reference picture, refIdxCol is equal to 0, and both motion vector components mvCol[0] and mvCol[1] lie in the range of -1 and 1 in quarter-sample units, inclusive. Otherwise, it is set 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 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, refIdx, subMbPartIdx = 0, list suffix LX, and partWidth = partHeight = 16 as the input and the output is assigned to mvLX.

  NOTE – The motion vector mvLX is equal to the motion vector prediction mvpLX that would be derived for the Pred_16x16_LX macroblock mode with refIdxLX given as input.

The prediction utilization flags predFlagL0 and predFlagL1 shall be derived as specified using Table 8-9.

Table 8-9 – Assignment of prediction utilization flags

refIdxL0	refIdxL1	predFlagL0	predFlagL1
>= 0	>= 0	1	1
>= 0	< 0	1	0
< 0	>= 0	0	1

8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode

This process is invoked if the direct_spatial_mv_pred_flag is equal to 0 for a B slice and when mb_type is equal to B_Skip, 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 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.

refIdxL0 = (refIdxCol < 0 ? 0 : MapColToList0(refIdxCol)) (8-124)

refIdxL1 = 0 (8-125)

NOTE - refIdxL0 and refIdxL1 index a list of fields if the current macroblock is a field macroblock (i.e., field_pic_flag equal to 1, or MbaffFrameFlag with mb_field_decoding_flag equal to 1 for the current macroblock pair); otherwise, both indices select from lists of frames.

The function MapColToList0(refIdxCol) is specified as follows. [Ed. Note (LW/GJS): Add a note indicating that a picture used as a short-term picture for reference in the colocated MB can be turned into a long-term picture before using it again for reference.]

- Let refPicCol be a frame, a field, or a pair of complementary fields that was referred by the reference index refIdxCol when decoding the co-located macroblock mbAddrCol inside the picture colPic.
- If vertMvScale is equal to One_To_One, MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference index list RefPicList0 that references refPicCol. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references refPicCol.
- If vertMvScale is equal to Frm_To_Fld, MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference index 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 index list RefPicList0 that references the frame or pair of complementary fields that contains refPicCol. RefPicList0 shall contain a variable PicNum or LongTermPicNum that references the frame or complementary field pair that contains refPicCol.

Depending on the value of vertMvScale the vertical component of mvCol is modified as follows.

- If vertMvScale is equal to Frm_To_Fld

  mvCol[1] = mvCol[1] / 2 (8-126)

- If vertMvScale is equal to Fld_To_Frm

  mvCol[1] = mvCol[1] * 2 (8-127)

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 that 4x4 luma sample blocks. For example, if 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 if DiffPicOrderCnt(pic1, pic0) with pic1 being the picture referred by RefPicList1[refIdxL1] and pic0 being the picture referred by RefPicList0[refIdxL0] is equal to 0, the motion vectors mvL0, mvL1 for the direct mode partition are derived by

$$mvL0 = mvCol \tag{8-128}$$

$$mvL1 = 0 \tag{8-129}$$

– 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 8-3)

$$X = (16384 + Abs(TD_D / 2)) / TD_D \tag{8-130}$$

$$DistScaleFactor = Clip3(-1024, 1023, (TD_B * X + 32) >> 6) \tag{8-131}$$

$$mvL0 = (DistScaleFactor * mvCol + 128) >> 8 \tag{8-132}$$

$$mvL1 = mvL0 – mvCol \tag{8-133}$$

where

$$TD_B = Clip3(-128, 127, DiffPicOrderCnt(CurrPic, RefPicList0[refIdxL0])), \tag{8-134}$$

$$TD_D = Clip3(-128, 127, DiffPicOrderCnt(RefPicList1[refIdxL1], RefPicList0[refIdxL0])) \tag{8-135}$$

NOTE - mvL0 and mvL1 cannot exceed the ranges specified in Annex A.

The prediction utilization flags predFlagL0 and predFlagL1 are both set to 1.

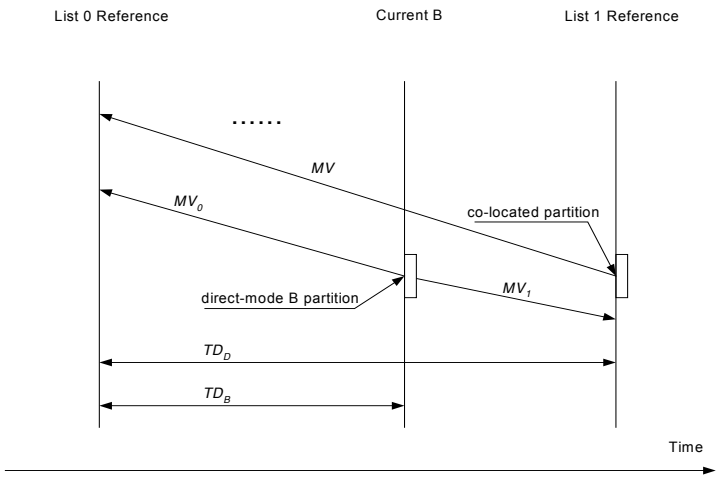


Figure 8-3 A direct-mode B partition has two derived motion vectors (mvL0, mvL1) pointing to two reference pictures referred by refIdxL0, refIdxL1. [Ed. Note(YK): In the figure, MV, MV0, and MV1 should be changed to mvCol, mvL0, and mvL1, respectively.]

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.

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, and list suffix LX as the input and the output being mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being either A, B, or C.

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 either A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

- mb_part_width(mb_type) is equal to 16, mb_part_height(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX, then

  mvpLX = mvLXB (8-136)

- mb_part_width(mb_type) is equal to 16, mb_part_height(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX, then

  mvpLX = mvLXA (8-137)

- mb_part_width(mb_type) is equal to 8, mb_part_height(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX, then

  mvpLX = mvLXA (8-138)

- mb_part_width(mb_type) is equal to 8, mb_part_height(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX, then

  mvpLX = mvLXC (8-139)

Figure 8-4 illustrates the non-median prediction as described above.

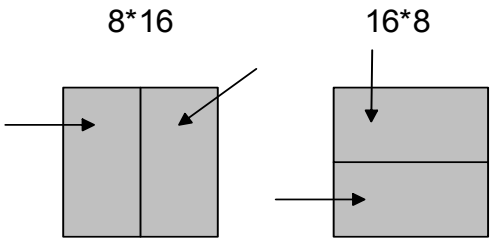


Figure 8-4 – Directional segmentation prediction (informative)

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 either A, B, or C),
- the motion vectors mvLXN (with N being either A, B, or C) of the neighbouring partitions,
- the reference indices refIdxLXN (with N being either 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 following rules are applied in sequential order to determine the motion vector prediction mvpLX:

- If both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available, then

  mvLXB = mvLXA (8-140)

  mvLXC = mvLXA (8-141)

  refIdxLXB = refIdxLXA (8-142)

  refIdxLXC = refIdxLXA (8-143)

- If one and only one of the reference indices refIdxLXA, refIdxLXB and 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, then the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

  mvpLX = mvLXN (8-144)

- 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]) (8-145)

  mvpLX[1] = Median(mvLXA[1], mvLXB[1], mvLXC[1]) (8-146)

8.4.1.3.2 Derivation process for the neighbouring blocks for motion data of neighbouring partitions

Inputs to this process are

- the macroblock partition index mbPartIdx,
- the sub-macroblock partition index subMbPartIdx,
- the list suffix LX

Outputs of this process are (with N being 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 and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, C, or D.
3. If the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies

$$mbAddrC = mbAddrD \tag{8-147}$$

$$mbPartIdxC = mbPartIdxD \tag{8-148}$$

$$subMbPartIdxC = subMbPartIdxD \tag{8-149}$$

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, then both components of mvLXN are set to zero and refIdxLXN is set to –1.
- Otherwise, the following applies.
  - The motion vector mvLXN and reference index refIdxLXN are those that are assigned to the macroblock or sub-macroblock partition mbPartIdxN\subMbPartIdxN inside the macroblock mbAddrN.
  - If the current macroblock a field macroblock and the macroblock mbAddrN is a frame macroblock

$$mvLXN[1] = mvLXN[1] / 2 \tag{8-150}$$

$$refIdxLXN = refIdxLXN * 2 \tag{8-151}$$

  - If the current macroblock a frame macroblock and the macroblock mbAddrN is a field macroblock

$$mvLXN[1] = mvLXN[1] * 2 \tag{8-152}$$

$$refIdxLXN = refIdxLXN / 2 \tag{8-153}$$

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 if the luma vector applies to 8x16 luma samples, the corresponding chroma vector applies to 4x8 chroma samples and if 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 picture is a frame (field_pic_flag equal to 0), and the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector mvCLX are derived by scaling the corresponding components of luma motion vector mvLX by 2, through mapping one-quarter sample mvLX units to one-eighth sample mvCLX units:

mvCLX[0] = mvLX[0] (8-154)
mvCLX[1] = mvLX[1] (8-155)

- If the current picture is a field (field_pic_flag equal to 1) or if the current macroblock is a field macroblock, only the horizontal component of the chroma motion vector mvCLX[0] is derived using Equation 8-154. The vertical component of the chroma motion vector mvCLX[1] is dependent on the parity of the current frame 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 8-10.

Table 8-10 – Calculation of vertical component of 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 $refPicLX_{Cb}$, $refPicLX_{Cr}$ 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 either 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

Inputs to this process are a reference index refIdxLX.

Outputs of this process are a reference picture consisting of an ordered two-dimensional array of luma samples $refPicLX_L$ and two ordered two-dimensional arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

The 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 frame, complementary reference field pairs, or non-paired reference fields that have been marked as "used for reference" as specified in subclause 8.2.7. 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, 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 8.2.6.

The reference pictures consist 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 $refPicLX_{Cr}$.

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.
- Otherwise, the following applies.
  - If the current macroblock is a frame macroblock, the reference frame or complementary field pair referred by PicNum = RefPicListX[refIdxLX] or LongTermPicNum = RefPicListX[refIdxLX] shall be the output.
  - Otherwise, the reference field or field of a reference frame referred by PicNum = RefPicListX[refIdxLX / 2] + 1 – (refIdxLX % 2) or LongTermPicNum = RefPicListX[refIdxLX / 2] + 1 – (refIdxLX % 2) shall be the output. [Ed Note (AG): State that this returns a field with partity conditioned on the parity of the current field macroblock: even for the same parity, odd for the opposite parity.]

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 $predPartLX_{Cb}$, $predPartLX_{Cr}$ 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 $refPicLX_{Cr}$. [Ed. Note (GJS): Actually, no. These variables are now also used in Annex A (the chroma ones are not). I suppose this means we should start these variable names with capital letters.]

For each luma sample location ($0 <= x_L < partWidth$, $0 <= y_L < partHeight$) inside the prediction luma sample array $predLX_L$, the corresponding predicted luma sample value $predLX_L(x_L, y_L)$ is derived as follows:

$$xInt_L = xA_L + (mvLX[0] >> 2) + x_L \quad (8\text{-}156)$$
$$yInt_L = yA_L + (mvLX[1] >> 2) + y_L \quad (8\text{-}157)$$

$$xFrac_L = mvLX[0] \& 3 \quad (8\text{-}158)$$
$$yFrac_L = mvLX[1] \& 3 \quad (8\text{-}159)$$

- The prediction sample value $predLX_L(x_L, y_L)$ 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 ($0 <= x_C < partWidth/2$, $0 <= y_C < partHeight/2$) inside the prediction chroma sample arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$, the corresponding prediction chroma sample values $predPartLX_{Cb}(x_C, y_C)$ and $predPartLX_{Cr}(x_C, y_C)$ are derived as follows:

$$xInt_C = (xA_L >> 1) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}160)$$
$$yInt_C = (yA_L >> 1) + (mvCLX[1] >> 3) + y_C \quad (8\text{-}161)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8\text{-}162)$$
$$yFrac_C = mvCLX[1] \& 7 \quad (8\text{-}163)$$

- The prediction sample value $predPartLX_{Cb}(x_C, y_C)$ 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 $predPartLX_{Cr}(x_C, y_C)$ 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 samples of the selected reference picture $refPicLX_L$

Output of this process is a predicted luma sample value $predPartLX_L(x_L, y_L)$.

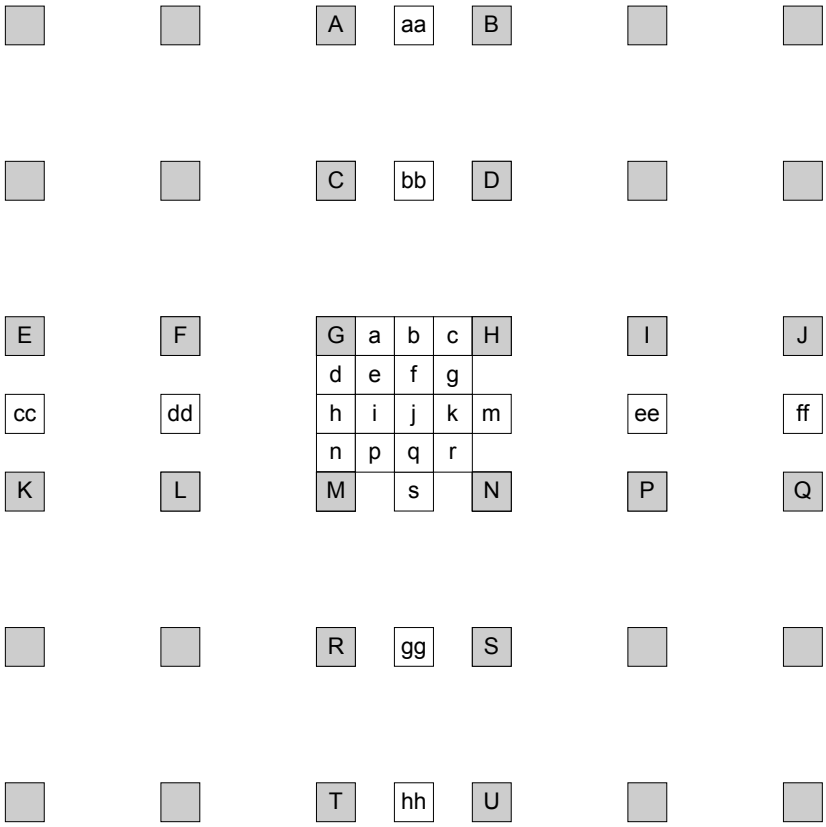


Figure 8-5 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation.

In Figure 8-5, 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:

$$xZ_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L + xDZ_L)$$
$$yZ_L = Clip3(0, PicHeightInSamples_L - 1, yInt_L + yDZ_L) \quad (8\text{-}164)$$

Table 8-11 specifies (xDZ_L, yDZ_L) for different replacements of Z.

Table 8-11 – 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:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}165)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}166)$$

The final prediction values b and h shall be derived using:

$$b = \text{Clip1}((b_1 + 16) >> 5) \quad (8\text{-}167)$$
$$h = \text{Clip1}((h_1 + 16) >> 5) \quad (8\text{-}168)$$

– 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.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}169)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}170)$$

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:

$$j = \text{Clip1}((j_1 + 512) >> 10) \quad (8\text{-}171)$$

– 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:

$$s = \text{Clip1}((s_1 + 16) >> 5) \quad (8\text{-}172)$$
$$m = \text{Clip1}((m_1 + 16) >> 5) \quad (8\text{-}173)$$

– 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:

$$a = (G + b + 1) >> 1 \quad (8\text{-}174)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}175)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}176)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}177)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}178)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}179)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}180)$$
$$q = (j + s + 1) >> 1. \quad (8\text{-}181)$$

– 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

$$e = (b + h + 1) >> 1 \quad (8\text{-}182)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}183)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}184)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}185)$$

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 8-12. The value of $predPartLX_L(x_L, y_L)$ shall be the output.

Table 8-12 – 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 8-6, 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.

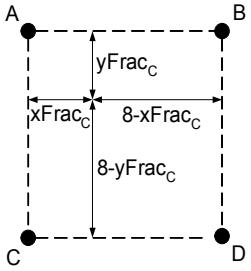


Figure 8-6 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D.

These samples may be used for generating the predicted chroma sample value $predPartLX_C(x_C, y_C)$.

$$xA_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C) \quad (8\text{-}186)$$
$$xB_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C + 1) \quad (8\text{-}187)$$
$$xC_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C) \quad (8\text{-}188)$$
$$xD_C = Clip3(0, PicWidthInSamples_C - 1, xInt_C + 1) \quad (8\text{-}189)$$

$$yA_C = Clip3(0, PicHeightInSamples_C - 1, yInt_C) \quad (8\text{-}190)$$
$$yB_C = Clip3(0, PicHeightInSamples_C - 1, yInt_C) \quad (8\text{-}191)$$
$$yC_C = Clip3(0, PicHeightInSamples_C - 1, yInt_C + 1) \quad (8\text{-}192)$$
$$yD_C = Clip3(0, PicHeightInSamples_C - 1, yInt_C + 1) \quad (8\text{-}193)$$

Given the chroma samples A, B, C, and D at full-sample locations, the predicted chroma sample value $predPartLX_C(x_C, y_C)$ is derived as follows:

$$predPartLX_C(x_C, y_C) = ((8 - xFrac_C) * (8 - yFrac_C) * A + xFrac_C * (8 - yFrac_C) * B + (8 - xFrac_C) * yFrac_C * C + xFrac_C * yFrac_C * D + 32) >> 6 \quad (8\text{-}194)$$

8.4.2.3 Weighted sample prediction process

Inputs to this process are the current partition given by the partition index mbPartIdx and the sub-macroblock partition index subMbPartIdx as well as a (partWidth)x(partHeight) array $predPartL0_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL0_{Cb}$, and $predPartL0_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 0 prediction) and/or a (partWidth)x(partHeight) array $predPartL1_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL1_{Cb}$, and $predPartL1_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 1 prediction), and refIdxL0 and refIdxL1.

Outputs of this process are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPart_{Cb}$, and $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

For P and SP slices, if weighted_pred_flag is equal to 0, then the default weighted sample prediction process as described in subclause 8.4.2.3.1 is used for macroblocks or partitions with predFlagL0 equal to 1. For P and SP slices, if weighted_pred_flag is equal to 1, then explicit weighted prediction is applied as described in subclause 8.4.2.3.2, for macroblocks or partitions with predFlagL0 equal to 1.

For B slices, if weighted_bipred_idc is equal to 0, then the default weighted sample prediction process as described in subclause 8.4.2.3.1 is used, for macroblocks or partitions with predFlagL0 and/or predFlagL1 equal to 1.

For B slices, if weighted_bipred_idc is equal to 1, explicit weighted sample prediction is used to form the final sample predictor, as described in subclause 8.4.2.3.2, for macroblocks or partitions with predFlagL0 and/or predFlagL1 equal to 1.

For B slices, if weighted_bipred_idc is equal to 2, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is used for macroblocks or partitions with one and only one of predFlagL0 or predFlagL1 equal to 1, and implicit weighted sample prediction as described in subclause 8.4.2.3.2 is used for skipped macroblocks or macroblock or partitions with both predFlagL0 and predFlagL1 equal to 1.

8.4.2.3.1 Default weighted sample prediction process

Input to this process are the current partition given by the partition index mbPartIdx and the sub-macroblock partition index subMbPartIdx as well as a (partWidth)x(partHeight) array $predPartL0_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL0_{Cb}$, and $predPartL0_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 0 prediction) and/or a (partWidth)x(partHeight) array $predPartL1_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL1_{Cb}$, and $predPartL1_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 1 prediction), and refIdxL0 and refIdxL1.

Output of this process are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPart_{Cb}$, and $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

This process is invoked for macroblocks or partitions that meet any of the following conditions:

- in P slices with weighted_pred_flag equal to 0 and predFlagL0 is equal to 1
- in B slices with weighted_bipred_idc equal to 0 and predFlagL0 and/or predFlagL1 equal to 1
- in B slices with weighted_bipred_idc equal to 2 and one but not both of predFlagL0 and predFlagL1 are equal to 1

All samples $predPart_C(x, y)$ of the array of final predicted luma or chroma samples $predPart_C$ are derived by using the samples $predPartL0_C(x, y)$ and/or $predPartL1_C(x, y)$, corresponding to the same location (x, y).

For the luma block, C = L, x = 0 .. partWidth - 1, y = 0 .. partHeight - 1,

and for the two chroma blocks, C = Cb and Cr, x = 0 .. partWidth / 2 - 1, y = 0 .. partHeight / 2 - 1

For partitions with predFlagL0 equal to 1 and predFlagL1 equal to 0

$$predPart_C(x, y) = predPartL0_C(x, y) \quad (8\text{-}195)$$

For partitions with predFlagL0 equal to 0 and predFlagL1 equal to 1

$$predPart_C(x, y) = predPartL1_C(x, y) \quad (8\text{-}196)$$

For partitions with both predFlagL0 and predFlagL1 equal to 1,

$$predPart_C(x, y) = (predPartL0_C(x, y) + predPartL1_C(x, y) + 1) >> 1. \quad (8\text{-}197)$$

8.4.2.3.2 Weighted sample prediction process

Input to this process are the current partition given by the partition index mbPartIdx and the sub-macroblock partition index subMbPartIdx as well as a (partWidth)x(partHeight) array $predPartL0_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL0_{Cb}$, and $predPartL0_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 0 prediction) and/or a (partWidth)x(partHeight) array $predPartL1_L$ of

prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPartL1_{Cb}$, and $predPartL1_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr (derived by list 1 prediction), refIdxL0 and refIdxL1, and the reference picture lists RefPicList0 and RefPicList1.

Output of this process are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and two (partWidth/2)x(partHeight/2) arrays $predPart_{Cb}$, and $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

This process is invoked for macroblocks or partitions that meet any of the following conditions:

- in P slices with weighted_pred_flag equal to 1 and predFlagL0 is equal to 1
- in B slices with weighted_bipred_idc equal to 1 and either predFlagL0 is equal to 1 and/or predFlagL1 is equal to 1
- in B slices with weighted_bipred_idc equal to 2 and predFlagL0 is equal to 1 and predFlagL1 is equal to 1

All samples $predPart_C(x, y)$ of the array of final predicted luma or chroma samples $predPart_C$ are derived by using the samples $predPartL0_C(x, y)$ and/or $predPartL1_C(x, y)$, corresponding to the same location (x, y).

The final predicted luma sample values $predPart_L(x, y)$ are derived by invoking this process with C = L, for x = = 0 .. partWidth - 1, y = 0 .. partHeight - 1.

The final predicted chroma sample values $predPart_{Cb}(x, y)$ and $predPart_{Cr}(x, y)$ are derived by invoking this process with C = Cb and Cr, for x = 0 .. partWidth/2 - 1, y = 0 .. partHeight/2 - 1.

If the partition given by mbPartIdx (and subMbPartIdx) has predFlagL0 equal to 1 and predFlagL1 equal to 0, the final predicted sample values $predPart_C(x, y)$ are derived as follows

$$\begin{aligned}&\text{if(LWD >= 1)}\\&\quad predPart_C(x, y) = Clip1(((predPartL0_C(x, y) * W_0 + 2^{LWD - 1}) >> LWD) + O_0)\\&\text{else}\\&\quad predPart_C(x, y) = Clip1(predPartL0_C(x, y) * W_0 + O_0)\end{aligned} \tag{8-198}$$

Else if the partition given by mbPartIdx (and subMbPartIdx) has predFlagL0 equal to 0 and predFlagL1 equal to 1, the final predicted sample values $predPart_C(x, y)$ are derived as follows.

$$\begin{aligned}&\text{if(LWD >= 1)}\\&\quad predPart_C(x, y) = Clip1(((predPartL1_C(x, y) * W_1 + 2^{LWD - 1}) >> LWD) + O_1)\\&\text{else}\\&\quad predPart_C(x, y) = Clip1(predPartL1_C(x, y) * W_1 + O_1)\end{aligned} \tag{8-199}$$

Else if the partition given by mbPartIdx (and subMbPartIdx) has both predFlagL0 and predFlagL1 equal to 1, the final predicted sample values $predPart_C(x, y)$ are derived by

$$\begin{aligned}predPart_C(x, y) = Clip1(((predPartL0_C(x, y) * W_0 + predPartL1_C(x, y) * W_1 + 2^{LWD}) >>\\(LWD + 1)) + ((O_0 + O_1 + 1) >> 1))\end{aligned} \tag{8-200}$$

If weighted_bipred_idc is equal to 2 and the slice_type is equal to B,

$$LWD = 5 \tag{8-201}$$

$$O_0 = 0 \tag{8-202}$$

$$O_1 = 0 \tag{8-203}$$

$$W_1 = DistScaleFactor >> 2 \tag{8-204}$$

where DistScaleFactor is specified in subclause 8.4.1.2.3, and

$$W_0 = 64 - W_1 \tag{8-205}$$

except if DiffPicOrderCnt(pic1, pic0) with pic1 being the picture referred by RefPicList1[refIdxL1] and pic0 being the picture referred by RefPicList0[refIdxL0] is equal to 0 or if one or both reference pictures is a long-term reference picture or if $W_1 < -64$ or if $W_1 > 128$, in which case,

$W_0 = 32$ (8-206)

$W_1 = 32$ (8-207)

Otherwise if weighted_bipred_idc is equal to 1, explicit mode WP is used as follows.

- If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

  refIdxL0WP = refIdxL0 >> 1 (8-208)

  refIdxL1WP = refIdxL1 >> 1 (8-209)

- Otherwise

  refIdxL0WP = refIdxL0 (8-210)

  refIdxL1WP = refIdxL1 (8-211)

- If C is equal to L for luma samples

  LWD = luma_log2_weight_denom (8-212)

  W_0 = luma_weight_l0[refIdxL0WP] (8-213)

  W_1 = luma_weight_l1[refIdxL1WP] (8-214)

  O_0 = luma_offset_l0[refIdxL0WP] (8-215)

  O_1 = luma_offset_l1[refIdxL1WP] (8-216)

- Otherwise if C is equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr,

  LWD = chroma_log2_weight_denom (8-217)

  W_0 = chroma_weight_l0[refIdxL0WP][iCbCr] (8-218)

  W_1 = chroma_weight_l1[refIdxL1WP][iCbCr] (8-219)

  O_0 = chroma_offset_l0[refIdxL0WP][iCbCr] (8-220)

  O_1 = chroma_offset_l1[refIdxL1WP][iCbCr] (8-221)

For explicit mode the following conditions shall be met

$-128 <= W_0 + W_1 <= 127$ (8-222)

NOTE – For implicit mode, weights are guaranteed to be in the range is $-64 \le W_0, W_1 \le 128$.

[Ed. clarify with regard to P, SP, B slice types]

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 Inter or Intra prediction samples for the applicaple component $pred_L$, $pred_{Cb}$, $pred_{Cr}$. [Ed. specify size of pred arrays]

Outputs of this process are the constructed samples arrays prior to deblocking for the applicaple component S'_L, S'_{Cb}, S'_{Cr}.

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 to 0.

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 2-D array c as the output.
2. The scaling and transformation process for residual luma 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and x'' 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_{yx} for x, y = 0..3 is derived as

$$u_{yx} = \text{Clip1}(pred_L(xO + x, yO + y) + x''_{yx}) \quad (8\text{-}223)$$

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 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 2-D array c as the output.
   b. The scaling and transformation process for luma DC 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 8-7 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.

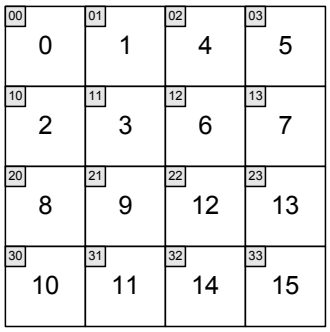


Figure 8-7 – Assignment of the indices of dcY to luma4x4BlkIdx.

The elements in lumaList with index k = 1..15 are specified as

lumaList[k] = Intra16x16ACLevel[luma4x4BlkIdx][k - 1] (8-224)

b. The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with lumaList as the input and the 2-D array c as the output.

c. The scaling and transformation process for residual luma 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and x'' 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_{yx} for x, y = 0..3 is derived as

$$u_{yx} = \text{Clip1}(\text{pred}_L(xO + x, yO + y) + x''_{yx}) \quad (8\text{-}225)$$

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 = 0 for Cb and iCbCr = 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 luma DC 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

$$c = \begin{pmatrix} \text{ChromaDCLevel}[\text{iCbCr}][0] & \text{ChromaDCLevel}[\text{iCbCr}][1] \\ \text{ChromaDCLevel}[\text{iCbCr}][2] & \text{ChromaDCLevel}[\text{iCbCr}][3] \end{pmatrix} \quad (8\text{-}226)$$

   b. The scaling and transformation process for chroma DC 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 8-8 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.

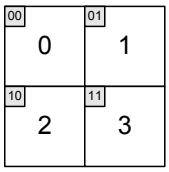


Figure 8-8 – Assignment of the indices of dcC to chroma4x4BlkIdx.

The elements in chromaList with index k = 1..15 are specified as

chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k - 1] (8-227)

b. The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with chromaList as the input and the 2-D 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 x'' 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) (8-228)

yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1)

e. The 4x4 array u with elements u_{yx} for x, y = 0..3 is derived as

u_{yx} = Clip1($pred_C$(xO + x, yO + y) + x''_{yx}) (8-229)

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 2-D 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 8-9 are used.

The inverse zig-zag scan shall be used for frame macroblocks and the inverse field scan shall be used for field macroblocks.

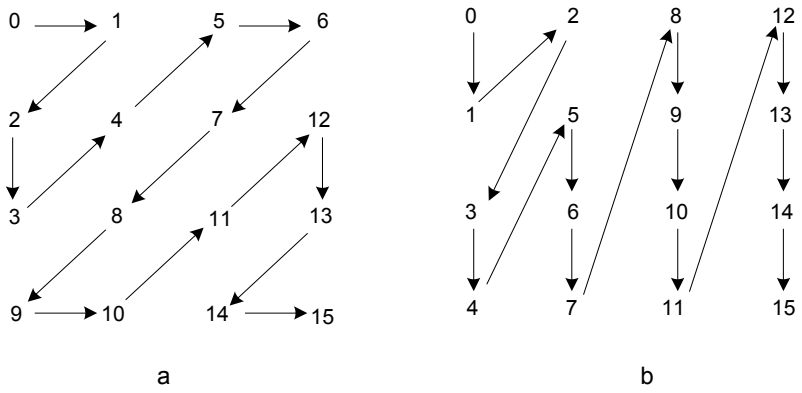


Figure 8-9 – a) Zig-zag scan. b) Field scan

Table 8-13 provides the mapping from the index idx of the 1-D list input list of 16 elements to indices i and j of the 2-D array c.

Table 8-13 – 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 quantisation parameters and scaling function

Input to this process is a two-dimensional array of transform coefficient levels.

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)

QP quantisation parameter values QP_Y, QP_C, QS_Y, and QS_C shall be in the range from 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 quantisation parameter doubles for every increment of 6 in QP_Y. Thus, there is an increase in scaling magnitude 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 8-14 based on the indexing denoted qP_I. The value of qP_I shall be derived as follows.

$$qP_I = \text{Clip3}(0, 51, QP_Y + \text{chroma\_qp\_index\_offset}) \quad (8\text{-}230)$$

Table 8-14 – 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:

$$\text{LevelScale}(m,i,j) = \begin{cases} v_{m0} & \text{for} \quad (i,j) \in \{(0,0),(0,2),(2,0),(2,2)\}, \\ v_{m1} & \text{for} \quad (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ v_{m2} & \text{otherwise;} \end{cases} \tag{8-231}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{8-232}$$

8.5.6 Scaling and transformation process for luma DC coefficients for Intra_16x16 macroblock type

Inputs to this process are transform coefficient level values for luma DC coefficients of Intra_16x16 macroblocks as a 4x4 array c of 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 of elements dcY_{ij}.

The inverse transform for the 4x4 luma DC coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-233}$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element of f that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After the inverse transform, scaling is performed according to the following:

- If QP_Y is greater than or equal to 12, then the scaled result shall be derived as

$$dcY_{ij} = [\, f_{ij} * \text{LevelScale}\,(\, QP_Y \,\%6,\, 0,\, 0\,)\,] << (\, QP_Y / 6 - 2\,), \quad i, j = 0..3\,. \tag{8-234}$$

- Otherwise, the scaled results shall be derived as

$$dcY_{ij} = [\, f_{ij} * \text{LevelScale}\,(\, QP_Y \,\%\, 6,\, 0,\, 0\,) + 2^{1-QP/6}\,]\, >> (\, 2 - QP_Y / 6\,), \quad i, j = 0..3\,. \tag{8-235}$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in element of dcY_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.5.7 Scaling and transformation process for chroma DC coefficients

Inputs to this process are transform coefficient level values for chroma DC coefficients of one chroma component of the macroblock as a 2x2 array c of 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 of elements dcC_{ij}.

The inverse transform for the 2x2 chroma DC coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{8-236}$$

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 according to the following.

- If QP_C is greater than or equal to 6, then the scaling result shall be derived as

$$dcC_{ij} = [\, f_{ij} * LevelScale(\, QP_C \,\%\, 6, 0, 0\,)\,] << (\, QP_C \,/\, 6 - 1\,), \quad i, j = 0, 1\,. \tag{8-237}$$

- Otherwise, the scaling results shall be derived by

$$dcC_{ij} = [\, f_{ij} * LevelScale\,(\, QP \,\%\, 6, 0, 0\,)\,] >> 1, \quad i, j = 0, 1\,. \tag{8-238}$$

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in any element of dcC_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.5.8 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 2-D array c of elements c_{mn} which is either an array relating to a luma residual block or to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array x of elements x_{mn}.

If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, the variable sMbFlag is set equal to 1, otherwise it 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

$$qP = QP_Y \tag{8-239}$$

- If the input array c relates to a luma residual block and sMbFlag is equal to 1

$$qP = QS_Y \tag{8-240}$$

- If the input array c relates to a chroma residual block and sMbFlag is equal to 0

$$qP = QP_C \tag{8-241}$$

- Otherwise (the input array c relates to a chroma residual and sMbFlag is equal to 1)

$$qP = QS_C \tag{8-242}$$

Scaling of 4x4 block 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

$$w_{00} = c_{00} \tag{8-243}$$

- Otherwise

$$w_{ij} = [\, c_{ij} * LevelScale(\, qP \,\%\, 6, m, n\,)\,] << (\, qP \,/\, 6\,), \quad i, j = 0..3\,. \tag{8-244}$$

The bitstream shall not contain data that results in a value of w_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After constructing an entire 4x4 block of scaled transform coefficients and assembling these into a 4x4 array w of elements w_{ij} illustrated as

$$w = \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \tag{8-245}$$

The transform process shall convert the block of reconstructed transform coefficients to a block of output samples in a manner mathematically equivalent to the following process:

1. Each column of reconstructed transform coefficients is transformed using a one-dimensional inverse transform, and
2. Each row of the resulting matrix is transformed using the same one-dimensional inverse transform.

The one-dimensional inverse transform is specified as follows for four input samples w_0, w_1, w_2, w_3, where the subscript indicates the one-dimensional frequency index.

1. A set of intermediate values is computed:

$$z_0 = w_0 + w_2 \tag{8-246}$$

$$z_1 = w_0 - w_2 \tag{8-247}$$

$$z_2 = (w_1 >> 1) - w_3 \tag{8-248}$$

$$z_3 = w_1 + (w_3 >> 1) \tag{8-249}$$

2. The transformed result is computed from these intermediate values:

$$x_0 = z_0 + z_3 \tag{8-250}$$

$$x_1 = z_1 + z_2 \tag{8-251}$$

$$x_2 = z_1 - z_2 \tag{8-252}$$

$$x_3 = z_0 - z_3 \tag{8-253}$$

The bitstream shall not contain data that results in a value of z_0, z_1, z_2, z_3, x_0, x_1, x_2, or x_3 that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive, in either the first (horizontal) or second (vertical) stage of application of this transformation process. The bitstream shall not contain data that results in a value of x_0, x_1, x_2, or x_3 that exceeds the range of integer values from -2^{15} to $2^{15}-33$, inclusive, in the second (vertical) stage of application of this transformation process.

After performing the both the one-dimensional horizontal and one-dimensional vertical inverse transforms to produce a block of transformed samples,

$$x' = \begin{bmatrix} x'_{00} & x'_{01} & x'_{02} & x'_{03} \\ x'_{10} & x'_{11} & x'_{12} & x'_{13} \\ x'_{20} & x'_{21} & x'_{22} & x'_{23} \\ x'_{30} & x'_{31} & x'_{32} & x'_{33} \end{bmatrix}, \tag{8-254}$$

the final reconstructed sample residual values shall be derived as

$$x''_{mn} = [x'_{mn} + 2^5] >> 6 \tag{8-255}$$

8.5.9 Picture construction process prior to deblocking filter process

Inputs to this process are

- luma4x4BlkIdx or chroma4x4BlkIdx
- a constructed sample residual 4x4 array x" with elements x''_{yx} which is either a luma or chroma residual block
- the prediction sample 4x4 array $pred_L$, $pred_{Cb}$, $pred_{Cr}$

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 x'' is a luma block, for each sample at position (x, y) 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).
- If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_L(xP + xO + x, yP + 2 * (yO + y)) = u_{yx} \quad (8\text{-}256)$$

- Otherwise,

$$S'_L(xP + xO + x, yP + yO + y) = u_{yx} \quad (8\text{-}257)$$

When x'' is a chroma block, for each sample at position (x, y) 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

$$xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) \quad (8\text{-}258)$$

$$yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) \quad (8\text{-}259)$$

- If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_C((xP >> 1) + xO + x, ((yP + 1) >> 1) + 2 * (yO + y)) = u_{yx} \quad (8\text{-}260)$$

- Otherwise,

$$S'_C((xP >> 1) + xO + x, ((yP + 1) >> 1) + yO + y) = u_{yx} \quad (8\text{-}261)$$

8.6 Decoding process for SP and SI slices

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 coefficient levels and the predicted luma samples for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to deblocking.

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 slices. Unlike P slices, 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 provide functionalities for bitstream switching, splicing, random access, VCR functionalities such as fast-forward, 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.

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.

Input to this process are Inter prediction samples for the current macroblock from subclause 8.4 and the prediction residual 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 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 an 4x4 array of prediction samples with element p_{ij} being derived as follows.

$$p_{ij} = pred_L(y + j, x + i) \tag{8-262}$$

The variable p is transformed producing transform coefficients c^p according to:

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{8-263}$$

The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the 2-D array c^r as the output with elements c_{ij}^r.

The prediction residual 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.

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * LevelScale(QP_Y \% 6, i, j) * A_{ij}) << (QP_Y / 6)) >> 6) \tag{8-264}$$

where LevelScale(m, i, j) is specified in Equation 8-231, and where A_{ij} is specified as:

$$A_{ij} = \begin{cases} 16 & \text{for} \quad (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ 25 & \text{for} \quad (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ 20 & \text{otherwise;} \end{cases} \tag{8-265}$$

The function LevelScale2(m, i, j), used in the formulas below, is specified as:

$$LevelScale2(m, i, j) = \begin{cases} w_{m0} & \text{for} \quad (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ w_{m1} & \text{for} \quad (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ w_{m2} & \text{otherwise;} \end{cases} \tag{8-266}$$

where the first and second subscripts of w are row and column indices, respectively, of the matrix specified as:

$$w = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \quad (8\text{-}267)$$

The resulting sum, c^s, is quantised with a quantisation parameter QS_Y and with i, j = 0..3 as follows.

$$c_{ij} = (\text{Sign}(c_{ij}^{\,s}) * (\text{Abs}(c_{ij}^{\,s}) * \text{LevelScale2}(QS_Y \,\%\, 6, i, j) + (1 << (14 + QS_Y / 6)))) >> (15 + QS_Y / 6) \quad (8\text{-}268)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and x'' as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(x''_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}269)$$

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 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

$$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \quad (8\text{-}270)$$

$$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \quad (8\text{-}271)$$

- Let p be an 4x4 array of prediction samples with element p_{ij} being derived as follows.

$$p_{ij} = \text{pred}_C(y + j, x + i) \quad (8\text{-}272)$$

- The 4x4 array p is transformed producing transform coefficients c^p(chroma4x4BlkIdx) using Equation 8-263.
- The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dc^r. This corresponding value is derived below.
- The elements in chromaList with index k = 1..15 are specified as

$$\text{chromaList}[k] = \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] \quad (8\text{-}273)$$

- The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with chromaList as the input and the 2-D array c^r as the output.

- The prediction residual 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.

$$c_{ij}^{\ s} = c_{ij}^{\ p}(\text{chroma4x4BlkIdx}) + (((c_{ij}^{\ r} * \text{LevelScale}(QP_C \% 6, i, j) * A_{ij}) << (QP_C / 6)) >> 6) \quad (8\text{-}274)$$

- 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.

$$c_{ij} = (\text{Sign}(c_{ij}^{\ s}) * (\text{Abs}(c_{ij}^{\ s}) * \text{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \quad (8\text{-}275)$$

After calculating the coefficients of the 4 chroma blocks of a chroma component of a macroblock, the DC coefficients of the 4 chroma blocks are assembled into a 2x2 matrix of elements c^p_{00}(chroma4x4BlkIdx) and an additional 2x2 transform is applied to the DC coefficients of these blocks. The 2 dimensional 2x2 transform procedure is specified by:

$$dc^p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c^p_{00}(0) & c^p_{00}(1) \\ c^p_{00}(2) & c^p_{00}(3) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}276)$$

The parsed chroma DC prediction residual coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are scaled using quantisation parameter QP, and added to the DC transform coefficients of the prediction block, as given by:

$$dc_{ij}^{\ s} = dc_{ij}^{\ p} + (((\text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] * \text{LevelScale}(QP_C \% 6, 0, 0) * A_{00}) << (QP_C / 6)) >>5) \quad \text{with i, j = 0, 1} \quad (8\text{-}277)$$

The resulting array of sums, dc^s, is quantised with a quantisation parameter QS_C, as given by:

$$dc_{ij}^{\ r} = (\text{Sign}(dc_{ij}^{\ s}) * (\text{Abs}(dc_{ij}^{\ s}) * \text{LevelScale2}(QS_C, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with i, j = 0, 1} \quad (8\text{-}278)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} dc^r_{00} & dc^r_{01} \\ dc^r_{10} & dc^r_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (8\text{-}279)$$

Scaling is performed according to the following:

- If QS_C is greater than or equal to 6, then the scaling result shall be derived as

$$c_{00}(j * 2 + i) = (f_{ij} * \text{LevelScale}(QS_C \% 6, 0, 0)) << (QS_C / 6 - 1) \quad \text{with i, j = 0, 1} \quad (8\text{-}280)$$

- Otherwise, the scaling results shall be derived by

$$c_{00}(j * 2 + i) = (f_{ij} * \text{LevelScale}(QS_C \% 6, 0, 0)) >> 1 \quad \text{with i, j = 0, 1} \quad (8\text{-}281)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and x'' as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(x''_{ij}) \text{ with i, j = 0..3} \quad (8\text{-}282)$$

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.

8.6.2 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 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 deblocking.

8.6.2.1 Luma transform coefficient decoding process

Inputs to this process are prediction luma samples $pred_L$ and the luma prediction residual 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 8-263 to produce transform coefficients c^p. These transform coefficients are then quantised with the quantisation parameter QS_Y, as follows:

$$c_{ij}^s = (Sign(c_{ij}^p) * (Abs(c_{ij}^p) * LevelScale2(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6)))) >> (15 + QS_Y / 6) \quad \text{with i, j = 0..3} \qquad (8\text{-}283)$$

The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the 2-D array c^r as the output with elements c_{ij}^r.

The 4x4 array c with elements c_{ij} with i, j = 0..3 is derived as follows.

$$c_{ij} = c_{ij}^r + c_{ij}^s \quad \text{with i, j = 0..3} \qquad (8\text{-}284)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.8 is invoked with c as the input and x'' as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = Clip1(x''_{ij}) \text{ with i, j = 0..3} \qquad (8\text{-}285)$$

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 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 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 8-263 to produce transform coefficients c^p. 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}^s

$$c_{ij}^s = (Sign(c_{ij}^p) * (Abs(c_{ij}^p) * LevelScale2(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \qquad (8\text{-}286)$$

- The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dc^r. This corresponding value is derived below.
- The elements in chromaList with index k = 1..15 are specified as

$$chromaList[k] = ChromaACLevel[iCbCr][chroma4x4BlkIdx][k - 1] \qquad (8\text{-}287)$$

- The inverse transform coefficient scanning process as described in subclause 8.5.4 is invoked with chromaList as the input and the 2-D array c^r(chroma4x4BlkIdx) as the output with elements c_{ij}^r(chroma4x4BlkIdx).

- 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.

$$c_{ij}(\text{chroma4x4BlkIdx}) = c_{ij}^{r}(\text{chroma4x4BlkIdx}) + c_{ij}^{s}(\text{chroma4x4BlkIdx}) \quad (8\text{-}288)$$

- 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 x'' as the output.
- The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}(x''_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}289)$$

- 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.

After calculating the coefficients of the 4 chroma blocks of a chroma component of a macroblock, the DC coefficients of the 4 chroma blocks, c_{00}^{p}(chroma4x4BlkIdx), are assembled into a 2x2 matrix of elements and an additional 2x2 transform is applied to the DC coefficients of these blocks according to Equation (dcp = …){Ed Note (MN) Need to insert a link to the correct equation}, resulting in DC transform coefficients dc_{ij}^{P}.

These DC transform coefficients are then quantised with the quantisation parameter QS_C, as given by:

$$dc_{ij}^{s} = (\text{Sign}(dc_{ij}^{p}) * (\text{Abs}(dc_{ij}^{p}) * \text{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \text{ with } i, j = 0, 1 \quad (8\text{-}290)$$

The parsed chroma DC prediction residual coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are added to these quantised DC transform coefficients of the prediction block, as given by:

$$dc_{ij}^{r} = dc_{ij}^{s} + \text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] \text{ with } i, j = 0, 1 \quad (8\text{-}291)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} dc_{00}^{r} & dc_{01}^{r} \\ dc_{10}^{r} & dc_{11}^{r} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (8\text{-}292)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is copied using:

$$c_{00}(j * 2 + i) = f_{ij} \text{ with } i, j = 0, 1 \quad (8\text{-}293)$$

8.7 Deblocking filter process

Inputs to this process are the decoded sample values of a picture, constructed as described in subclause 8.5.9; the prediction mode information (whether intra or inter and whether I_PCM) for each macroblock; information on which 4x4 luma blocks in the picture were coded using non-zero coefficients; the motion vectors used for generating motion-compensated predictions; the quantisation parameter values, QP, used for coding each macroblock; the deblocking filter syntax elements and variables disable_deblocking_filter_idc, FilterOffsetA, and FilterOffsetB, computed as described in subclause 7.4.3; and, when MbaffFrameFlag is equal to 1, the coding mode (frame/field) for each macroblock pair in a frame.

Outputs of this process are the filtered sample values for a decoded picture.

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. For the deblocking filter process and the effect of the filter control variables, the edges controlled by a macroblock are specified to be the block edges internal to the macroblock and the top and left edges between the current macroblock and its neighbouring macroblocks.

This filtering process shall be performed on a macroblock basis, with all macroblocks in a picture processed in order of increasing macroblock addresses.

For each macroblock in the picture:

– If disable_deblocking_filter_idc for the current slice is equal to 1, then deblocking is disabled for all edges controlled by the current macroblock.
– If disable_deblocking_filter_idc for the current slice is equal to 2, then deblocking is applied to the edges controlled by the current macroblock, with the exception of the edges that are also slice boundaries.
– Otherwise, deblocking is applied to all of the edges controlled by the current macroblock.

Prior to the operation of the deblocking filter process for each macroblock, the deblocked samples of the macroblock or macroblock pair above and the macroblock or macroblock pair to the left of the current macroblock shall be available.

The deblocking filter process applies to both luma and chroma. 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 in each direction, as shown on the left side of Figure 8-10. The chroma deblocking filter process is performed in the same order on two 8-sample edges in each direction, as shown on the right side of Figure 8-10. 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.

In field mode filtering, horizontal edges are filtered using samples from only one field. In frame mode filtering, horizontal edges are filtered using samples from both fields. Vertical edges are always filtered within a row of samples of a single field, regardless of whether the filtering is frame mode or field mode filtering.

If MbaffFrameFlag is equal to 1, then the following rules apply:

– If the current macroblock is a field macroblock, all edges controlled by the macroblock are filtered in field mode.
– If the current macroblock is a frame macroblock, all edges controlled by the macroblock with the exception of the top edge of the macroblock are filtered in frame mode. The top edge is filtered in frame mode if it borders a neighbouring frame macroblock above it; otherwise it is filtered in field mode.

NOTE - When field mode filtering is applied to the horizontal edges of a frame mode 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 and 1 horizontal chroma edge are filtered that are internal to a macroblock. When field mode filtering is applied to the top edges of a frame mode macroblock, 2 horizontal luma and 2 horizontal chroma edges between the frame mode macroblock and the above macroblock pair are filtered using field mode filtering, for a total of 5 horizontal luma edges and 3 horizontal chroma edges filtered that are considered to be controlled by the frame mode macroblock. In all other cases, at most 4 horizontal luma and 2 horizontal chroma edges are filtered that are considered to be controlled by a particular macroblock.

Otherwise (if MbaffFrameFlag is equal to 0):

– If field_pic_flag is equal to 1, field mode filtering is applied to the edges controlled by the macroblock.
– Otherwise, frame mode filtering is applied to the edges controlled by the macroblock.

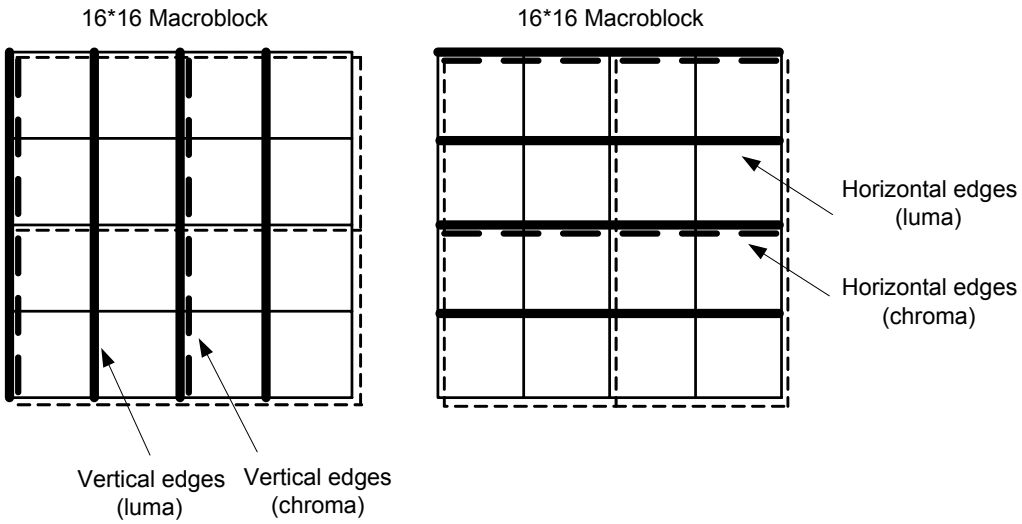


Figure 8-10 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dashed lines)

In the following description, the set of eight samples across a 4x4 block horizontal or vertically boundary is denoted as p_i and q_i (i = 0..3) as shown in Figure 8-11 with the actual boundary lying between p_0 and q_0. For chroma edges, samples p_2, p_3, q_2, and q_3 are never involved in the filtering process.

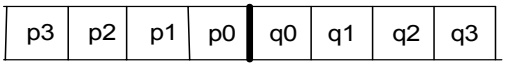


Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

Let s' be a variable specifying a luma or chroma sample array. If frame mode filtering is applied, s' specifies the luma or chroma sample array of the frame. Otherwise (field mode filtering is applied), s' specifies the luma or chroma sample array of the field.

The set of samples p_i and q_i (i = 0..3) are specified as follows.

- p_0 is the sample to the right of a vertical edge or the sample below a horizontal edge with location (x, y) relative to the upper left sample of the luma or chroma array s'.
- The samples p_i with i = 1..3 and q_i with i = 0..4 are specified as follows [Ed.(TW) clarify for edges of the picture].
  - For the filtering of vertical edges,

$$q_i = s'(x + i, y) \quad (8\text{-}294)$$

$$p_i = s'(x – i – 1, y) \quad (8\text{-}295)$$

  - For the filtering of horizontal edges,

$$q_i = s'(x, y + i) \quad (8\text{-}296)$$

$$p_i = s'(x, y – i – 1) \quad (8\text{-}297)$$

8.7.1 Derivation process for the content dependent boundary filtering strength

Inputs to this process are the prediction mode information (intra or inter) for each macroblock; information on which 4x4 luma blocks were coded using non-zero coefficients; the motion vectors used for generating motion-compensated predictions; and, when MbaffFrameFlag is equal to 1, the coding mode (frame/field) for each macroblock pair.

Outputs of this process is the Boundary Strength variable, Bs, for each line of an edge to be filtered, where a line of an edge is represented by a set of samples p_i and q_i (i = 0..3).

As specified in Figure 8-10, every block boundary of a chroma block corresponds to a specific boundary of a luma block. The Bs value used for filtering a specific line of a chroma edge is selected as follows:

- When filtering in frame mode, the Bs used for filtering a line of a horizontal or vertical chroma edge is the same as the value of Bs for filtering the line 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.
- When filtering in field mode, the Bs used for filtering a line of a horizontal or vertical chroma edge is the same as the value of Bs for filtering the line 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.

If Bs is equal to 0, filtering is skipped for that particular edge. In all other cases filtering is dependent on the local sample properties and the value of Bs for this particular set of samples p_i and q_i (i = 0..3).

Let MixedModeEdgeFlag be assigned a value of 1 if MbaffFrameFlag is equal to 1 and the samples p_i and q_i (i = 0..3) (see **Error! Reference source not found.**) are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair; otherwise, MixedModeEdgeFlag is assigned a value of 0.

A value of Bs equal to 4 is assigned if the block boundary is also a macroblock boundary and any of the following conditions are true:

– the samples p_0 and q_0 are both in frame macroblocks and either of the macroblocks containing samples p_0 and q_0 is coded using an Intra macroblock prediction mode

– MbaffFrameFlag is equal to 1, the edge is a vertical edge and either of the macroblocks containing samples p_0 and q_0 is coded using an Intra macroblock prediction mode.

If the value of Bs has not been assigned based on the above conditions, then a value of Bs equal to 3 is assigned if any of the following conditions are true:

– MixedModeEdgeFlag is equal to 0 and either of the macroblocks containing samples p_0 and q_0 is coded using an Intra macroblock prediction mode

– MixedModeEdgeFlag is equal to 1, the edge is a horizontal edge, and either of the macroblocks containing samples p_0 and q_0 is coded using an Intra macroblock prediction mode

If the value of Bs has not been assigned based on the above conditions, then a value of Bs equal to 2 is assigned if:

– the 4x4 luma block containing sample p_0 or the 4x4 luma block containing sample q_0 contains non-zero transform coefficient levels

If the value of Bs has not been assigned based on the above conditions, then a value of Bs equal to 1 is assigned if any of the following conditions are true:

– MixedModeEdgeFlag is equal to 1

– MixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock partition containing the sample p_0 different reference pictures or a different number of reference pictures are used than for the prediction of the macroblock partition containing the sample q_0.

– 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 vector used is greater than 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 a motion vector used in the prediction of the two the macroblock/sub-macroblock partitions for the same reference picture is greater than 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 4 in quarter-sample luma units 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 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 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 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, Bs shall be assigned the value 0, resulting in filtering not being applied for that particular boundary segment.

8.7.2 Derivation process for the thresholds for each block boundary

Input to this process are the input sample values p_i and q_i (i = 0..3) of a single line of an edge to be filtered, the quantisation parameter values, QP, used for coding the macroblocks containing the samples p_0 and q_0, the deblocking filter control variables FilterOffsetA and FilterOffsetB that apply to this edge segment, and the value of the Boundary Strength variable, Bs, for the corresponding line of an edge.

Output of this process is a Boolean decision as to whether sample values will be modified on the input line of the edge, and the value of $Index_A$, which is used to access the table of C0 values for filtering lines of edges with Bs smaller than 4, as specified in subclause 8.7.3, the values of the threshold variables (α and β), which are used for luma filtering of lines of edges with Bs equal to 4, as specified in subclause 8.7.4.

In the following, uppercase letters indicate filtered samples and lower case letters indicate unfiltered samples with regard to the current edge filtering operation. However, p_1, p_2 and p_3 may indicate samples that have been modified by the filtering of a previous block edge. A line of an edge is only filtered if the condition

$$Bs \neq 0 \;\; \&\& \;\; Abs(p_0 - q_0) < \alpha \;\; \&\& \;\; Abs(p_1 - p_0) < \beta \;\; \&\& \;\; Abs(q_1 - q_0) < \beta \tag{8-298}$$

is true. The values of the thresholds α and β are dependent on the average value of QP for the macroblock(s) containing the samples p_0 and q_0 as well as on a pair of index offsets FilterOffsetA and FilterOffsetB that may be transmitted in the slice header for the purpose of modifying the characteristics of the filter. The average QP value for the is computed as

$$QPav = (QP_p + QP_q + 1) >> 1 \tag{8-299}$$

where QP_p is the quantisation parameter of the macroblock containing the sample p_0 and QP_q is the quantisation parameter of the macroblock containing the sample q_0.

NOTE – The above statement is true regardless of the value of MbaffFrameFlag.

For luma sample filtering, if the macroblock containing the sample p_0 (or q_0) is an I_PCM macroblock, the value 0 shall be used for QP_p (or QP_q) in Equation 8-299; otherwise, the value of QP_Y for the macroblock containing the sample p_0 (or q_0) shall be used.

For chroma sample filtering, if the macroblock containing the sample p_0 (or q_0) is an I_PCM macroblock, the value of QP_C that corresponds to a value of 0 for QP_Y shall be used for QP_p (or QP_q) in Equation 8-299; otherwise, the value of QP_C for the macroblock containing the sample p_0 (or q_0) shall be used.

The index used to access the α table (Table 8-15), as well as the C0 table (Table 8-16) that is used in filtering of edges with Bs smaller than 4, is computed as:

$$IndexA = Clip3(0, 51, QP_{av} + FilterOffsetA) \tag{8-300}$$

NOTE - In SP and SI slices, QP_{av} is derived in the same way as in other slice types. QS_Y from Equation 7-18 is not used in the deblocking filter.

The index used to access the β table (Table 8-15) is computed as:

$$IndexB = Clip3(0, 51, QP_{av} + FilterOffsetB) \quad (8\text{-}301)$$

The relationships between the indices computed using Equations 8-300 and 8-301 and the thresholds (α and β) are shown in Table 8-15.

Table 8-15 – QP_{av} and offset dependent threshold variables α and β

	$Index_A$ (for α) or $Index_B$ (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 8-15 (continued)

	$Index_A$ (for α) or $Index_B$ (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

8.7.3 Filtering process for edges with Bs smaller than 4

Inputs to this process are the input sample values p_i and q_i (i = 0..3) of a single line of an edge that is to be filtered, the value of the Boundary Strength variable, Bs, for the corresponding line of an edge, the value of the threshold variable β, and the value of $Index_A$, as specified in 8.7.2.

Outputs of this process are the filtered result sample values for each line of input sample values.

Two types of filtering are specified. In the case of lines of edges with 0 < Bs < 4, the samples having values p_0 and q_0 shall be modified to filtered values P_0 and Q_0 as given by:

$$\Delta = Clip3(–C, C, ((((q_0 – p_0) << 2) + (p_1 – q_1) + 4) >> 3)) \quad (8\text{-}302)$$

$$P_0 = Clip1(p_0 + \Delta) \quad (8\text{-}303)$$

$$Q_0 = Clip1(q_0 – \Delta) \quad (8\text{-}304)$$

where C is determined as specified below.

The two intermediate threshold variables

$$a_p = Abs(p_2 – p_0) \quad (8\text{-}305)$$

$$a_q = Abs(q_2 – q_0) \quad (8\text{-}306)$$

shall be used to determine whether filtering for the luma samples p_1 and q_1 is taking place at this position of the edge.

If $a_p < \beta$ for a line of a luma edge, the sample having value p_1 shall be modified to a filtered value P_1 as specified by

$$P_1 = p_1 + Clip3(–C0, C0, (p_2 + ((p_0 + q_0 + 1) >> 1) – (p_1 << 1)) >> 1) \quad (8\text{-}307)$$

If $a_q < \beta$ for a line of a luma edge, the sample having value q_1 shall be modified to a filtered value Q_1 as specified by

$$Q_1 = q_1 + Clip3(–C0, C0, (q_2 + ((p_0 + q_0 + 1) >> 1) – (q_1 << 1)) >> 1) \quad (8\text{-}308)$$

where C0 is specified in Table 8-16. Chroma samples p_1 and q_1 are never filtered.

For luma, C is determined by setting it equal to C0 and then incrementing it by one if $a_p < \beta$, and again by one if $a_q < \beta$. For chroma, C is equal to C0 + 1.

Table 8-16 – Value of filter clipping variable C0 as a function of $Index_A$ and Bs

	$Index_A$																									
	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 8-16 (continued)

	$Index_A$																									
	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

8.7.4 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 line of an edge that is to be filtered, the values of the threshold variables α and β for the line of the edge segment, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values for each line of input sample values.

In this subclause, the intermediate variables a_p and a_q are computed as specified in Equations 8-305 and 8-306.

For luma samples, if the following condition holds:

$$a_p < \beta \;\&\&\; Abs(p_0 - q_0) < ((\alpha >> 2) + 2) \tag{8-309}$$

filtering of the left/upper side of the line of the edge shall be applied by replacing the samples having values p_0, p_1, and p_2 with the filtered values P_0, P_1, and P_2 according to:

$$P_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3 \tag{8-310}$$

$$P_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \tag{8-311}$$

$$P_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3 \tag{8-312}$$

For chroma samples, and for luma samples for which the condition described in Equation 8-309 does not hold, the filter shall be applied by replacing the sample having value p_0 with the filtered value P_0 according to:

$$P_0 = (2*p_1 + p_0 + q_1 + 2) >> 2 \tag{8-313}$$

Similarly, for filtering of luma samples on the right/lower side of the line of the edge, if the following condition holds:

$$a_q < \beta \;\&\&\; Abs(p_0 - q_0) < ((\alpha >> 2) + 2) \tag{8-314}$$

filtering shall be applied by replacing the samples having values q_0, q_1, and q_2 with the filtered values Q_0, Q_1, and Q_2 according to:

$$Q_0 = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3 \quad (8\text{-}315)$$

$$Q_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}316)$$

$$Q_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}317)$$

For chroma samples, and for luma samples for which the condition described in Equation 8-314 does not hold, the filter shall be applied by replacing the sample having value q_0 with the filtered value Q_0 according to:

$$Q_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}318)$$

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax elements.

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 if entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax elements.

This subclause specifies the parsing process for the Exp-Golomb-coded or truncated Exp-Golomb-coded syntax elements.

Syntax elements coded as ue(v), me(v), or se(v) are 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 zero. 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:

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

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
…	…

When the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

When 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.

When 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.

When 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.

If x is greater than 1, then codeNum shall be parsed in the same way as syntax elements coded as ue(v), me(v), or se(v).

If x is equal to 1, the parsing process for codeNum for syntax elements coded as te(v) is given by a process equivalent to:

```
b = read_bits( 1 )
codeNum = !b
```

Finally, the value of the syntax element is equal to codeNum.

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
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 coded_block_pattern codewords 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 coefficients

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, an array coeffLevel, a maximum number of coefficients maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx of the current block of transform coefficients.

Output of this process is the coeffLevel array containing transform coefficients.

The process is specified in the following ordered steps:

1. All coefficients, with indices from 0 to maxNumCoeff - 1, in the coeffLevel array are set to zero.
2. The total number of coefficients total_coeff and the number of trailing one coefficients trailing_ones are derived by parsing coeff_token (see subclause 9.2.1). If the number of coefficients total_coeff is equal to 0, the coeffLevel array containing 0 values is returned and no further step is carried out. Otherwise, the following steps are carried out.
3. The levels of non-zero coefficients are derived by parsing trailing_ones_sign_flag and coeff_level (see subclause 9.2.2).
4. The runs of zero coefficients before each non-zero coefficient are derived by parsing total_zeros and run_before (see subclause 9.2.3).
5. The level and run information are combined into the coeffLevel array (see subclause 9.2.4).

9.2.1 Parsing process for total number of coefficients and trailing ones

Inputs to this process are bits from slice data, a maximum number of coefficients maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx of the current block of transform.

Outputs of this process are total_coeff and trailing_ones.

The syntax element coeff_token is decoded using one of the five VLCs specified in five columns of Table 9-5. Each VLC specifies both total_coeff and trailing_ones 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, then nC is set to –1,

Otherwise, the following applies.

- If the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set to 0.
- 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 A, and the 4x4 luma block specified by mbAddrB\luma4x4BlkIdxB is assigned to B.
- Let nA and nB be the number of non-zero coefficients (given by total_coeff) in the block of transform coefficients A located to the left of the current block and the block of transform coefficients B located above the current block, respectively.
- 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 A, and the 4x4 chroma block specified by mbAddrB\iCbCr\luma4x4BlkIdxB is assigned to B.

With N replaced by A and B, in mbAddrN and nN the following applies.

- If any of the following conditions is true, nN is set to zero.
  - mbAddrN is not available
  - The current macroblock is coded as Intra prediction mode, constrained_intra_pred_flag is equal to 1 and mbAddrN is coded as Inter prediction and slice data partitioning is in use (nal_unit_type is in the range of 2 through 4, inclusive).
- If mbAddrN is an I_PCM macroblock, nN is set to 16.
- Otherwise, nN is set to the syntax element total_coeff of the neighbouring block N.

NOTE - The values nA and nB that are derived using total_coeff do not include the DC coefficients in Intra 16x16 macroblocks or DC coefficients in chroma blocks, because these coefficients 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 coefficients.

NOTE - When parsing for Intra16x16DCLevel, the values nA and nB are based on the number of coded coefficients in adjacent 4x4 blocks and not on the number of coded DC coefficients in adjacent 16x16 blocks.

- If both mbAddrA and mbAddrB are available, the variable nC is set to (nA + nB + 1) >> 1. Otherwise, nC is set to nA + nB.

The value total_coeff resulting from decoding coeff_token shall not exceed maxNumCoeff.

Table 9-5 – coeff_token mapping to total_coeff and trailing_ones

trailing_ones	total_coeff	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 coefficients total_coeff and, the number of trailing one coefficients trailing_ones.

Output of this process is a list level containing coefficient levels.

Initially an index i is set to zero. Then the following procedure is iteratively applied trailing_ones() times to decode the trailing one coefficients, if any:

- A 1-bit syntax element trailing_ones_sign_flag is decoded.
- If trailing_ones_sign_flag is equal to 0, the value +1 is assigned to level[i]. Otherwise, the value -1 is assigned to level[i].

- The index i is incremented by 1.

Following the decoding of the trailing one coefficients, a variable suffixLength is initialised as follows.

- If total_coeff is larger than 10 and trailing_ones is smaller than 3, suffixLength is set to 1.
- Otherwise suffixLength is set to 0.

The following procedure is then applied iteratively (total_coeff – trailing_ones) times to decode the remaining levels, if any:

- The syntax element coeff_level is decoded in two steps. In a first step a value of levelPrefix is decoded using the VLC specified in Table 9-6. In a second step an unsigned integer levelSuffix is read from the slice data. The size in bits of the unsigned integer is equal to suffixLength with the exception of the following two cases:
    - if levelPrefix is equal to 14 and suffixLength is 0, the size is 4 bits
    - if levelPrefix is equal to 15, the size is 12 bits.
- A variable levelCode is set to (levelPrefix << suffixLength) + levelSuffix.
- If levelPrefix is equal to 15 and suffixLength is equal to 0, levelCode is incremented by 15.
- If the index i is equal to trailing_ones and trailing_ones is smaller than 3, levelCode is incremented by 2.
- 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].
- If suffixLength is equal to zero, suffixLength is set to 1.
- If the absolute value of level[i] is larger than (3 << (suffixLength – 1)) and suffixLength is smaller than 6, suffixLength is incremented by 1.
- The index i is incremented by 1.

Table 9-6 – Codeword table for levelPrefix

levelPrefix	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 coefficients total_coeff, and the maximum number of coefficients maxNumCoeff.

Output of this process is a list of runs of zero coefficients preceding non-zero coefficients called run.

Initially an index i is set to 0. If the number of coefficients total_coeff is equal to the maximum number of coefficients maxNumCoeff, a variable zerosLeft is set to 0. Otherwise, total_zeros is decoded and zerosLeft is set 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 VLCs from Table 9-7 and Table 9-8 are used.

The following procedure is then applied iteratively (total_coeff – 1) times:

- If zerosLeft is larger than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. run[i] is set to run_before. Otherwise run[i] is set to 0.
- The value of run[i] is subtracted from zerosLeft and the result assigned to zerosLeft. The result of the subtraction shall be larger 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 total_coeff() 1 to 7

total_zeros( )	total_coeff( )						
	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 total_coeff() 8 to 15

total_zeros( )	total_coeff( )							
	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( )	total_coeff( )		
	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 coefficient levels called level, a list of runs called run, and the number of non-zero coefficients total_coeff.

Output of this process is an array coeffLevel of transform coefficients.

A variable coeffNum is set to -1 and an index i is set to (total_coeff – 1). The following procedure is iteratively applied total_coeff times:

- coeffNum is incremented by run[i] + 1.
- coeffLevel[coeffNum] is set 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 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. If 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.

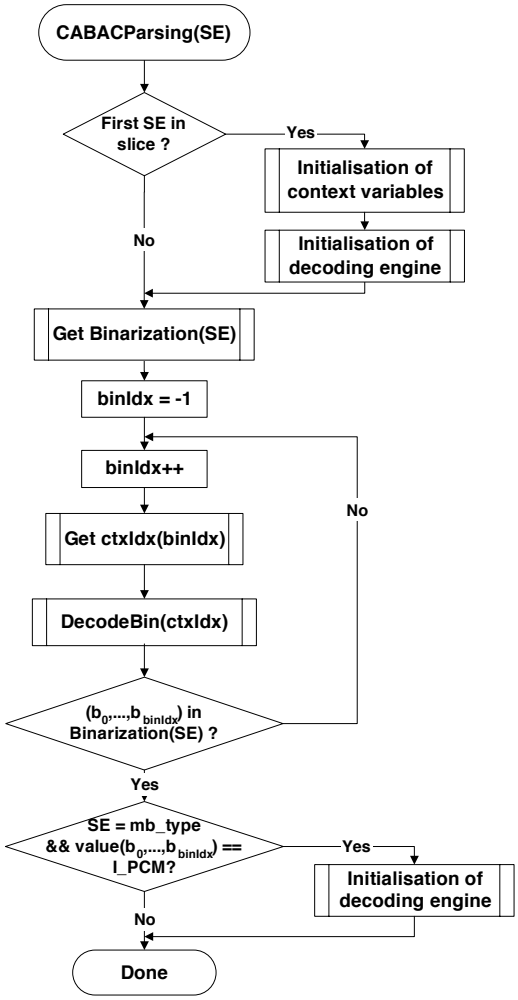


Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

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 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 7-17. Given the two table entries (m, n),

1. preCtxState = Clip3(1, 126, ((m * $SliceQP_Y$) >> 4) + n)
2. if(preCtxState <= 63) {

        pStateIdx = 63 - preCtxState

        valMPS = 0

    } else {

        pStateIdx = preCtxState - 64

        valMPS = 1

    }

In the following table, 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 depend also on the value of the cabac_init_idc syntax parameters. 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.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, as shown in Figure 9-2, codIRange is set to 0x01FE and codIOffset is set 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.

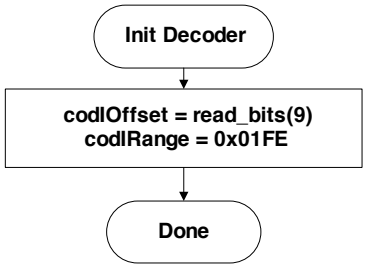


Figure 9-2 – Flowchart of initialisation of the decoding engine

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 is 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 bitstrings and suffix bitstrings are referenced as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part 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. If two values for each of these variables are specified for one syntax element of a given slice type 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.

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 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. In all other cases, 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.

The possible values of the context index ctxIdx range from 0 to 398. 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	Value of 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 (CodedBlockPatternLuma part only)	as specified in subclause 9.3.2.6	3	73
coded_block_pattern (CodedBlockPatternChroma part only)		1	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 value C is a bitstring of length C+1 indexed by BinIdx. The bins for binIdx less than C are equal to 1. The bin with binIdx equal to C 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 values smaller 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 prefixes of the binarization are specified by invoking the TU binarization process for prefS = Min(uCoff, Abs(S)) of a syntax element value S as specified in subclause 9.3.2.2 with cMax = uCoff, where uCoff > 0.

The bin string of a syntax element having value S consists only of a prefix bit string, if one of the following is true:

- signedValFlag is equal to 0 and S is less than uCoff.
- signedValFlag is equal to 1 and S is equal to 0.

Otherwise, the bin string of the UEGk suffix part of a syntax element value S is specified by a process equivalent to the following pseudo-code:

```
if( Abs( S ) >= uCoff ) {
    sufS = Abs( S ) − uCoff;
```

```
        while( 1 ) {
            if( sufS >= (unsigned int)( 1 << k ) ) {
                put( 1 );
                sufS = sufS − ( 1<<k );
                k++;
            } else {
                put( 0 );
                while( k− − )
                    put( ( sufS >> k ) & 0x01 );
                break;
            }
        }
    }
    if ( signedValFlag == 1 ) {
        if ( S != 0 ) {
            if ( S > 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 L-bit unsigned integer bin string of the syntax element value, where L = 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 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.

When entropy_coding_mode_flag is equal to 1, 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 a prefix as specified in Table 9-27 and a suffix 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 prefix 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 bitstring is parsed for each syntax element.

After parsing each bit, the resulting bitstring is compared to all bin strings of the binarization of the syntax element.

If the bitstring is equal to one of the bin strings, the corresponding value of the syntax element is the output. Otherwise, the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx=0 for the first bin. If the bin strings are concatenated, the variable binIdx is set to 0 for the first bin of each part of the bin string (prefix and suffix part).

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, 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 the 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, then the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-29. If more than one value is listed for a binIdx, then the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.
- Otherwise, the ctxIdx is specified to be the sum of 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 larger 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

Value (name) of 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 ctxIdx is specified by ctxIdxOffset + ctxIdxBlockCatOffset(ctxBlockCat) + ctxIdxInc(ctxBlockCat). 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.

If mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule 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 condTermN (with N being either A or B) be a variable that is set as follows.

- If mbAddrN is not available or mb_skip_flag for the macroblock mbAddrN is equal to 1, condTermN is set to 0.
- Otherwise, condTermN is set to 1.

ctxIdxInc is derived as

$$ctxIdxInc = condTermA + condTermB \tag{9-1}$$

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 specified in subclause 6.4.6 is invoked and the output is assigned to mbAddrA and mbAddrB.

If both macroblocks mbAddrN and mbAddrN+1 are skipped, the inference rule specified in subclause 7.4.4 shall be applied.

Let condTermN (with N being either A or B) be a variable that is set as follows.

- If any of the following conditions is true, then condTermN is set to 0,
  - mbAddrN is not available
  - the macroblock mbAddrN is a frame macroblock.

- Otherwise, condTermN is set to 1.

ctxIdxInc is derived as

$$ctxIdxInc = condTermA + condTermB \quad (9\text{-}2)$$

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 condTermN (with N being either A or B) be a variable that is set as follows.

- condTermN is set to 0 if any of the following conditions is true
  - 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, condTermN is set to 1.

ctxIdxInc is derived as

$$ctxIdxInc = condTermA + condTermB \quad (9\text{-}3)$$

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.

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 condTermN (with N being either A or B) be a variable that is set as follows.
  - If any of the following conditions is true, then condTermN is set to 0
    - mbAddrN is not available [Ed. Note (DM): removal of conflicting condition with regard to JM implementation, also addressed by Ed. Note (PH) below]
    - mb_type for the macroblock mbAddrN is equal to I_PCM
    - if the macroblock mbAddrN is not skipped and ((CodedBlockPatternLuma >> luma8x8BlkIdxN) & 1) is not equal to 0 for the macroblock mbAddrN
  - Otherwise, condTermN is set to 1.
- ctxIdxInc is derived as

$$ctxIdxInc = condTermA + 2 * condTermB \quad (9\text{-}4)$$

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 condTermN (with N being either A or B) be a variable that is set as follows.
  - condTermN is set to 1 if mbAddrN is available and mb_type for the macroblock mbAddrN is equal to I_PCM

- condTermN is set to 0 if any of the following conditions is true
  - 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, condTermN is set to 1.

- ctxIdxInc is derived as

$$ctxIdxInc = condTermA + 2 * condTermB + (binIdx == 1 \ ? \ 4 : 0) \quad (9\text{-}5)$$

9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta

Output of this process is ctxIdxInc.

Let mbAddrN be the macroblock address of the macroblock that precedes the current macroblock in decoding order. If the current macroblock is the first macroblock of a slice, mbAddrN is marked as not available.

ctxIdxInc is set to 0, if any of the following conditions is true

- mbAddrN is not available or the macroblock mbAddrN is skipped
- mb_type of the macroblock mbAddrN is equal to I_PCM
- The macroblock mbAddrN is not coded in Intra_16x16 prediction mode and both CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock mbAddrN are equal to 0
- mb_qp_delta for the macroblock mbAddrN is equal to 0

Otherwise, ctxIdxInc is set 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 index list suffix lX, where X = 0 or 1.

Output of this process is ctxIdxInc.

The derivation process for neighbouring partitions specified in subclause 6.4.7.5 is invoked with mbPartIdx and subMbPartIdx = 0 as input and the output is assigned to mbAddrA\mbPartIdxA and mbAddrB\mbPartIdxB.

Let refIdxZeroFlagN (with N being either A or B) be a variable that is set as follows.

- If MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

$$refIdxZeroFlagN = (ref\_idx\_lX[mbPartIdxN] > 1 \ ? \ 0 : 1) \quad (9\text{-}6)$$

- Otherwise

$$refIdxZeroFlagN = (ref\_idx\_lX[mbPartIdxN] > 0 \ ? \ 0 : 1) \quad (9\text{-}7)$$

Let condTermN (with N being either A or B) be a variable that is set as follows.

- condTermN is set to 0 if any of the following conditions is true
  - mbAddrN is not available or the macroblock mbAddrN is skipped
  - The macroblock mbAddrN is coded in Intra prediction mode
  - mb_type for the macroblock mbAddrN is equal to B_Direct_16x16
  - sub_mb_type[mbPartIdxN] for the macroblock mbAddrN is equal to B_Direct_8x8
  - refIdxZeroFlagN is equal to 1
- Otherwise, condTermN is set to 1.

ctxIdxInc is derived as

$$ctxIdxInc = condTermA + 2 * condTermB \quad (9\text{-}8)$$

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 index list suffix lX, and ctxIdxOffset

Output of this process is ctxIdxInc.

The derivation process for neighbouring partitions specified in subclause 6.4.7.5 is invoked with mbPartIdx and subMbPartIdx as input and the output is assigned to mbAddrA\mbPartIdxA\subMbPartIdxA and mbAddrB\mbPartIdxB\subMbPartIdxB.

Let compIdx be a variable specifying the component of the motion vector difference to decode. compIdx is set to 0 if ctxIdxOffset is equal to 40. Otherwise, compIdx is set to 1.

Let absMvdCompN (with N being either A or B) be a variable that is set as follows.

- absMvdCompN is set to 0 if any of the following conditions is true
  - mbAddrN is not available or the macroblock mbAddrN is skipped
  - The macroblock mbAddrN is coded in Intra prediction mode
  - mb_type for the macroblock mbAddrN is equal to B_Direct_16x16
  - sub_mb_type[mbPartIdxN] for the macroblock mbAddrN is equal to B_Direct_8x8
- 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)

  - 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)

ctxIdxInc is derived as follows

- If (absMvdCompA + absMvdCompB) is less than 3, ctxIdxInc is set to 0.
- If (absMvdCompA + absMvdCompB) is greater than 32, ctxIdxInc is set to 2.
- Otherwise, ctxIdxInc is set 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 condTermN (with N being either A or B) be a variable that is set as follows.

- condTermN is set to 0 if any of the following conditions is true
  - 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, condTermN is set to 1.

ctxIdxInc is derived as

ctxIdxInc = condTermA + condTermB (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

- If ctxBlockCat is equal to 1 or 2, luma4x4BlkIdx
- If ctxBlockCat is equal to 3, the chroma component index compIdx
- If ctxBlockCat is equal to 4, chroma4x4BlkIdx and the chroma component index compIdx

Output of this process is ctxIdxInc(ctxBlockCat).

Let transBlockN (with N being either A or B) be a variable representing a block of transform coefficients, it is set 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).
  - 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.
- 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).
  - 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)) is not equal to 0 for the macroblock mbAddrN, then the 4x4 luma block with luma4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.
- 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).
  - 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, then the chroma DC block of chroma component compIdx of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.
- If 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).
  - 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, then the 4x4 chroma block with chroma4x4BlkIdxN of the chroma component compIdx of macroblock mbAddrN is assigned to transBlockN.
  - Otherwise, transBlockN is marked as not available.

Let condTermN (with N being either A or B) be a variable that is set as follows.

- condTermN is set to 0 if any of the following conditions is true
  - 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).

- condTermN is set to 1 if any of the following conditions is true
  - 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, condTermN is set to the value of the coded_block_flag of the transform block transBlockN that was decoded for the macroblock mbAddrN.

ctxIdxInc(ctxBlockCat) is derived as

$$ctxIdxInc(ctxBlockCat) = condTermA + 2 * condTermB \quad (9\text{-}13)$$

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

Input 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 coefficients (for macroblock coded in Intra16x16 prediction mode)	16	0:Luma-Intra16-DC
block of luma AC coefficients (for macroblock coded in Intra16x16 prediction mode)	15	1:Luma-Intra16-AC
block of luma coefficients (for macroblock not coded in Intra16x16 prediction mode)	16	2:Luma-4x4
block of chroma DC coefficients	4	3:Chroma-DC
block of chroma AC coefficients	15	4:Chroma-AC

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:

$$ctxIdxInc = scanningPos \tag{9-14}$$

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 coefficients with absolute value equal to 1, and let numDecodAbsLevelGt1 denotes the accumulated number of decoded coefficients 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 as

$$ctxIdxInc = ((numDecodAbsLevelGt1 \; != 0 \;) \; ? \; 0 : Min(4, 1 + numDecodAbsLevelEq1)) \tag{9-15}$$

- Otherwise, ctxIdxInc is derived as

$$ctxIdxInc = 5 + Min(4, numDecodAbsLevelGt1) \tag{9-16}$$

9.3.3.2 Arithmetic decoding process

Input 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 9-3 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.
- 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, DecodeDecision( ) as specified in subclause 9.3.3.2.1 shall be applied.

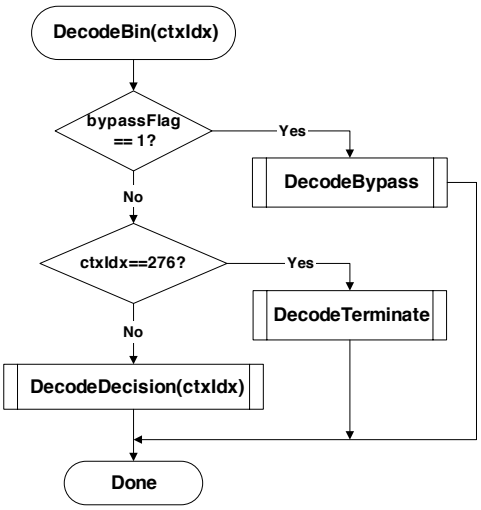


Figure 9-3 – Overview of the arithmetic decoding process for a single bin (informative)

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) | 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 64×4 pre-computed product values of $Q_i \times p_{LPS}$(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 S, and the updated variables codIRange and codIOffset.

Figure 9-4 shows the flowchart for decoding a single decision (DecodeDecision). In a first step, the value of the variable codIRangeLPS associated with the coding sub-interval related to the LPS decision is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index of a quantised value of codIRange, which is denoted by the variable qCodIRangeIdx:

$$qCodIRangeIdx = (codIRange >> 6) \,\&\, 0x03 \qquad (9\text{-}17)$$

Given qCodIRangeIdx and pStateIdx associated with ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-33 is assigned to codIRangeLPS:

$$codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] \qquad (9\text{-}18)$$

In a second step, the value of codIRange – codIRangeLPS is assigned to codIRange to which the current value of codIOffset is compared. If codIOffset is larger than or equal to codIRange the logical complement of valMPS is assigned to the decoded value S, codIOffset is decremented by codIRange and codIRange is set to codIRangeLPS*;* otherwise valMPS is assigned to S.

Given the decoded value, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization will be 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 S 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 S, the update of the two variables pStateIdx and valMPS associated with ctxIdx is derived as follows:

```
if( S  = =  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.

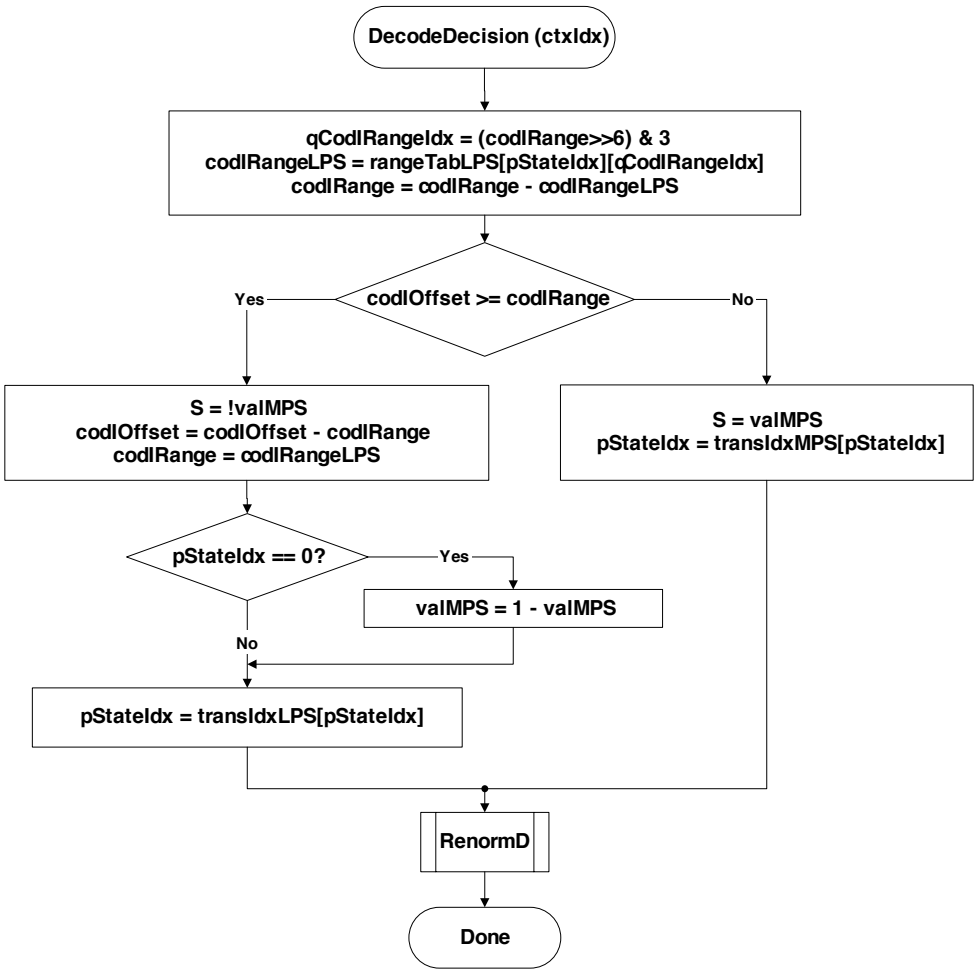


Figure 9-4 – Flowchart for decoding a decision

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 9-5. The current value of codIRange is first compared to 0x0100: If it is larger than or equal to 0x0100, no renormalization is needed and the RenormD process is finished; otherwise, 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).

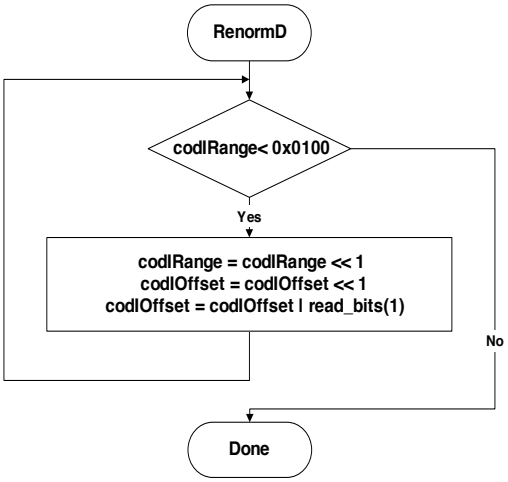


Figure 9-5 – Flowchart of renormalization

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 variables codIRange and codIOffset, and the decoded value.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9-6 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. If codIOffset is larger than or equal to codIRange, a value of 1 is decoded and codIOffset is decremented by codIRange; otherwise, a value of 0 is decoded.

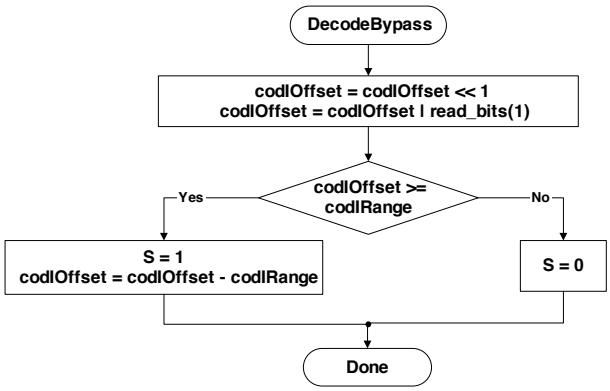


Figure 9-6 – Flowchart of bypass decoding process

9.3.3.2.4 Decoding process for binary decisions before termination

Input to this process are bits from slice data and the variables codIRange and codIOffset.

Output of this process are the updated variables codIRange and codIOffset, and the decoded value.

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 9-7 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. If codIOffset is larger than or equal to codIRange, a value of 1 is decoded; otherwise, a value of 0 is decoded. When the decoded value is equal to 1 no renormalization is carried out and CABAC decoding is terminated. In such a case the last bit inserted in register codIOffset is rbsp_stop_one_bit.

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.

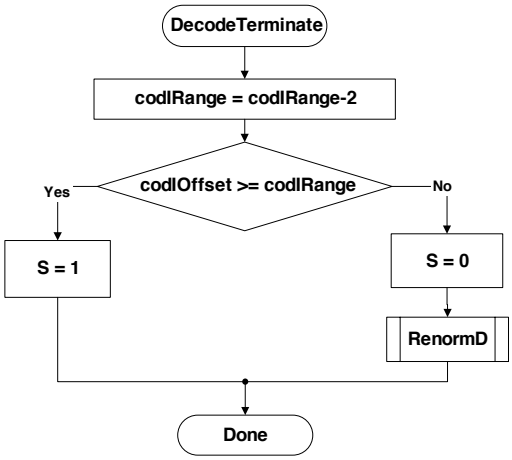


Figure 9-7 – Flowchart of decoding a decision before termination

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, if entropy_coding_mode_flag is equal to 1.

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, as illustrated in Figure 9-8, codILow is set to zero, and the range codIRange is set to 0x01FE. Furthermore, a firstBitFlag is set to 1, and bitsOutstanding, and symCnt counters are set to zero.

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. If 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)).

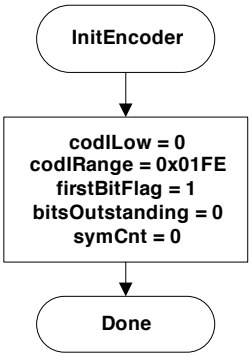


Figure 9-8 – Flowchart of initialisation of the encoding engine

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 decision to be encoded S, and the variables codIRange, codILow and symCnt.

Outputs of this process are the variables codIRange, codILow and symCnt.

Figure 9-9 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS associated with the coding sub-interval related to the LPS decision 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 S is compared to valMPS associated with ctxIdx. If S is different from valMPS, codIRange is added to codILow and then codIRange is set 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.

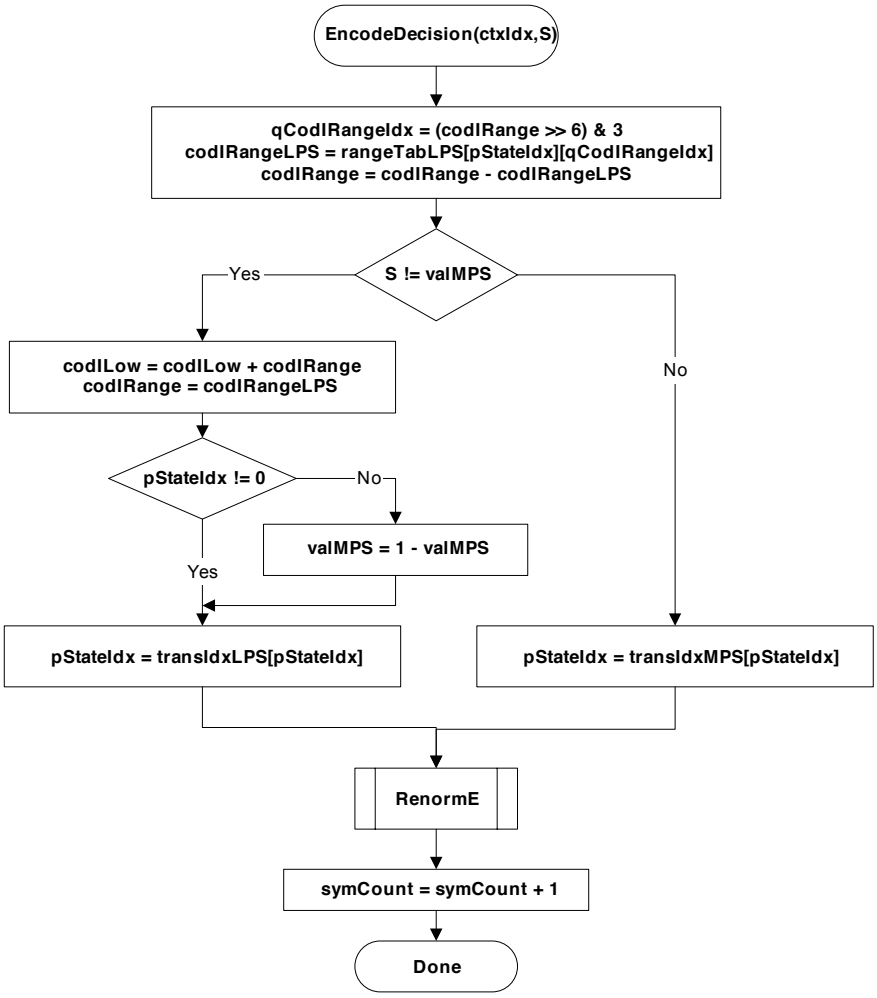


Figure 9-9 – Flowchart for encoding a decision

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 9-10. The current value of codIRange is first compared to 0x0100. If it is larger than or equal to that value, no renormalization is needed and RenormE is finished. Otherwise, the renormalization loop is entered. Within this loop, it is first determined if a 0 or a 1 can safely be output, i.e., there is no possibility for a carry over in a future iteration. Otherwise, the variable bitsOutstanding is incremented by 1. Finally, the values of codILow and codIRange are doubled.

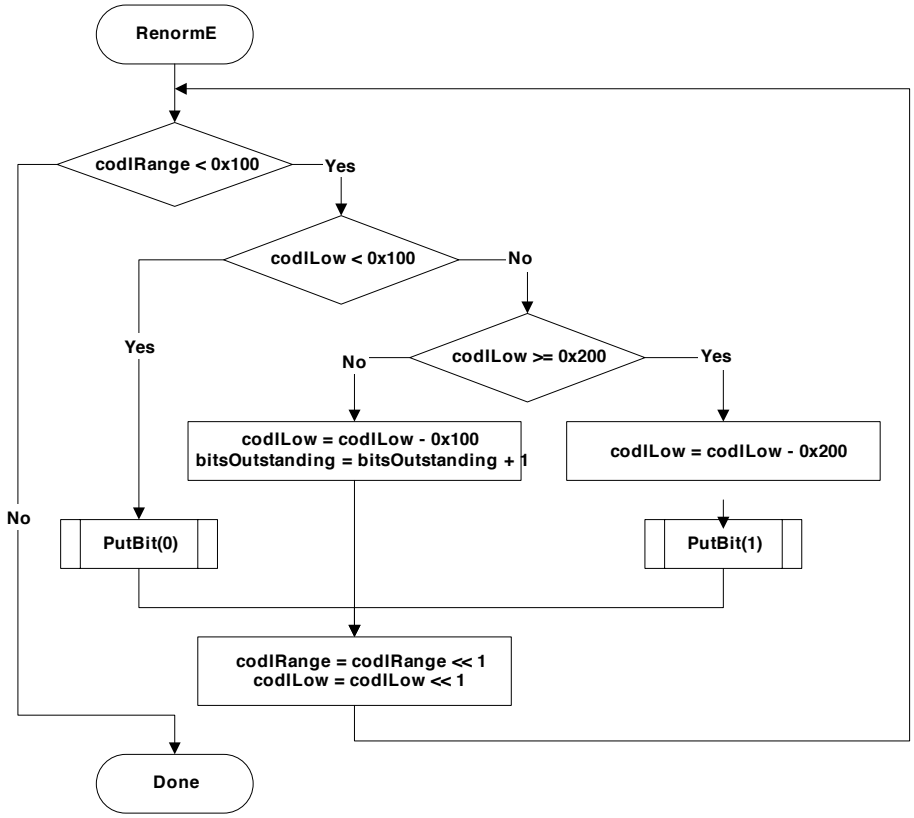


Figure 9-10 – Flowchart of renormalization in the encoder

The PutBit procedure described in Figure 9-11 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.

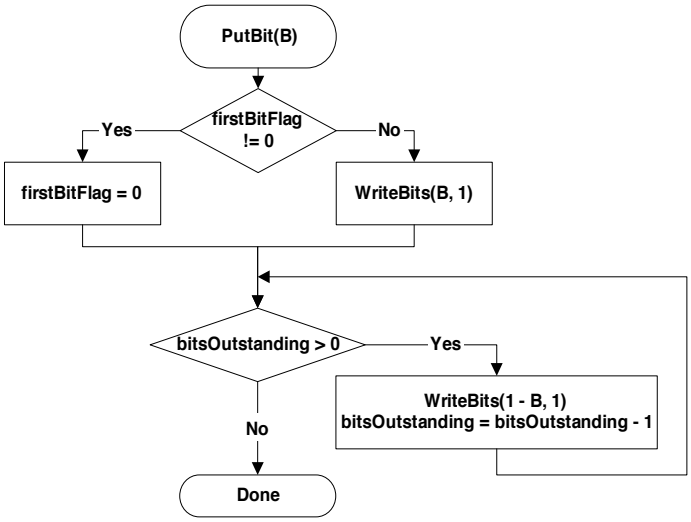


Figure 9-11 – Flowchart of putting bit

9.3.4.4 Bypass encoding process for binary decisions (informative)

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

Input to this process is a binary symbol S and the variables codILow, codIRange, bitsOutstanding, and symCnt.

Output of this process is a bit written to the RBSP and the updated variables codILow, codIRange, 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 9-12.

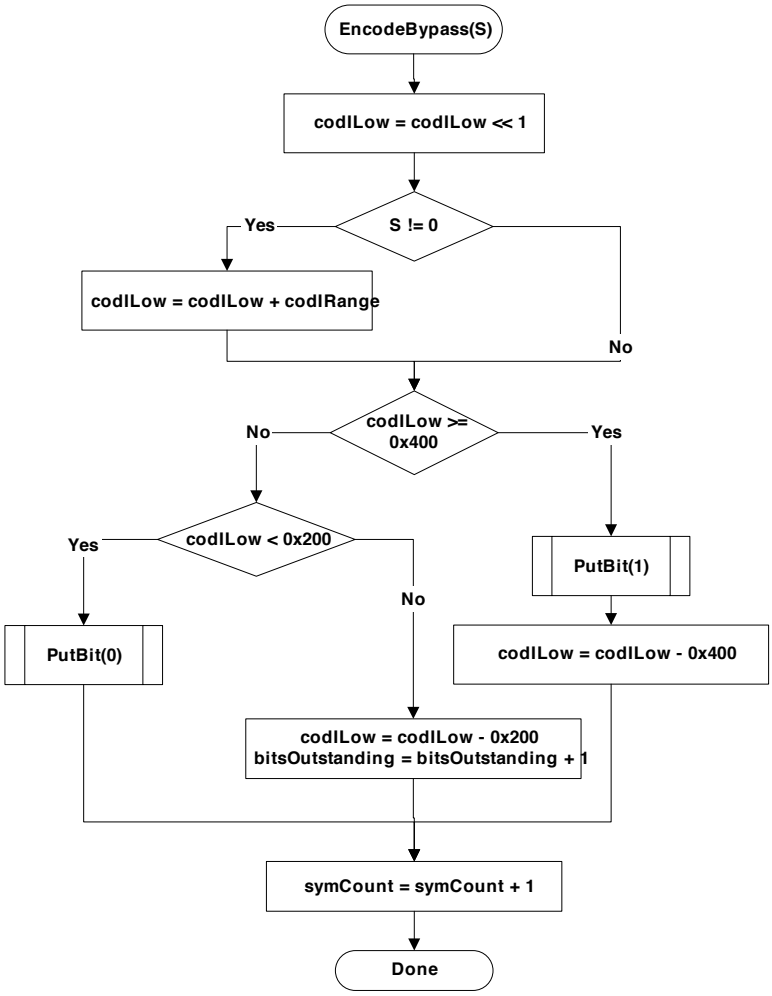


Figure 9-12 – Flowchart of encoding bypass

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 a binary symbol S and the variables 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 9-13 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.

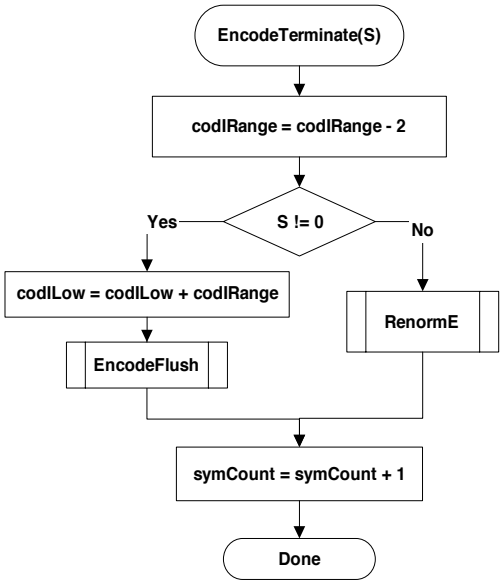


Figure 9-13 – Flowchart of encoding a decision before termination

When the symbol S to encode is equal to 1 CABAC encoding is terminated and the flushing procedure shown in Figure 9-14 is applied after encoding the end_of_slice_flag. In such a case the last bit written by WriteBits(B, N) contains the rbsp_stop_one_bit.

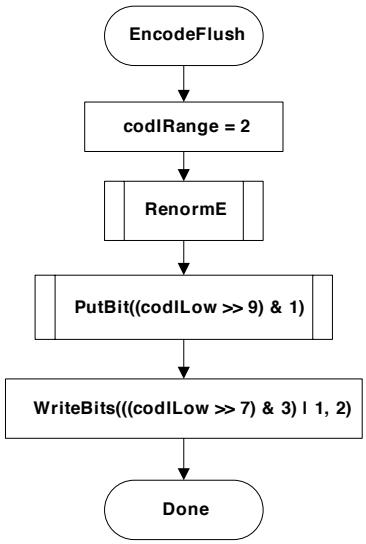


Figure 9-14 – Flowchart of flushing at termination

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 NumBytesInNALunitsTotal of all NAL units of a picture with nal_unit_type equal to 1, 2, 3, 4 or 5, the number of macroblocks PicSizeInMbs in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all slice layer NAL units of the picture with nal_unit_type equal to 1, 2, 3, 4 or 5.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let K be Ceil((Ceil((3 * BinCountsInNALunits – 3 * 96 * PicSizeInMbs) / 32) – NumBytesInNALunitsTotal) / 3). If K is smaller than or equal to zero, no cabac_zero_word is appended to the NAL unit. Otherwise, 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 limits on the capabilities needed to decode the bitstream. 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 if 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 equal to 2 through 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 less than 8.

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 conforming to the Baseline profile at the specified level and at all levels below 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 equal to 2 through 4, inclusive.
- Sequence parameter sets shall have arbitrary_slice_order_allowed_flag equal to 0 only.
- Sequence parameter sets shall have more_than_one_slice_group_allowed_flag equal to 0 only.

– Sequence parameter sets shall have redundant_pictures_allowed_flag equal to 0 only.
– For bitstreams conforming to levels 1, 1.1, 1.2, 1.3, 2, 5 and 5.1, sequence parameter sets shall have frame_mbs_only_flag equal to 1.

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 conforming to the Main profile at the specified level and at all levels below the specified level.

Decoders conforming to the Main profile at a specified level shall also be capable of decoding all bitstreams conforming to the Baseline profile (as specified by profile_idc being equal to 66) at the specified level and at all levels below the specified level in which the following additional sequence parameter set syntax constraints are obeyed:

– more_than_one_slice_group_allowed_flag is equal to 0,
– arbitrary_slice_order_allowed_flag is equal to 0, and
– redundant_pictures_allowed_flag is equal to 0.

A.2.3 Extended profile

Bitstreams conforming to the Extended profile shall obey the following constraints:

– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.
– Picture parameter sets shall have num_slice_groups_minus1 less than 8.
– For bitstreams conforming to levels 1, 1.1, 1.2, 1.3, 2, 5, and 5.1, sequence parameter sets shall have frame_mbs_only_flag equal to1.

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 conforming to the Extended profile at the specified level and at all levels below the specified level.

Decoders conforming to the Extended profile at a specified level shall also be capable of decoding all bitstreams conforming to the Baseline profile (as specified by profile_idc being equal to 66) at the specified level and at all levels below the specified level.

A.3 Levels

A.3.1 Profile-independent level limits

Bitstreams conforming to any profile at a specified level shall obey the following constraints:

a) The difference between consecutive removal times as defined in subclause C.1.2, satisfies the constraint that $t_r(n) - t_r(n-1) >= \max(PicSizeInMbs \div MaxMBPS, FR)$, where MaxMBPS is specified in Table A-1. If picture n-1 is a frame, FR = 1 ÷ 172. If picture n-1 is a field, FR = 1 ÷ (172 * 2). [Ed. (->AG): Also include the similar constraint on output times.]

b) the total number of bytes of NAL unit data (i.e., the total of the NumBytesInNALunit variables for the picture) associated with the primary coded picture n is <= 256 * ChromaFormatFactor * MaxMBPS * ($t_r(n) - t_r(n-1)$) ÷ MinCR. [Ed. Note (->AG): Also, consider the cases for n=0 and the last picture. Also, consider relationship to output timing.]

c) FrameSizeInMbs <= MaxFS, where MaxFS is specified in Table A-1

d) PicWidthInMbs <= Sqrt(MaxFS * 8)

e) FrameHeightInMbs <= Sqrt(MaxFS * 8)

f) DPB size max_dec_frame_buffering <= MaxDPB / (FrameSizeInMbs * 256 * ChromaFormatFactor). MaxDPB is given in Table A-1 in the unit of bytes. max_dec_frame_buffering is a sequence VUI parameter. DPB size is used by the HRD (see Annex C). [Ed. Note (GJS) This should be clarified such that if max_dec_frame_buffering is not present, the value here is the default. If max_dec_frame_buffering is present it must obey the constraint stated here. And if max_dec_frame_buffering is present, then the constraint expressed there is the one that must be obeyed rather than the one here.]

g) bit_rate[k] <= MaxBR and cpb_size[k] <= the level limit given, where bit_rate[k] is given by Equation E-13 and cpb_size[k] is given by Equation E-14 if vcl_hrd_parameters_present_flag is equal to 1. MaxBR is specified in Table A-1. The bitstream shall satisfy these conditions for at least one value of k in the range 0 to

cpb_cnt_minus1. [Ed. Note (GJS/AG): When HRD parameters not present, the bitstream must still obey something.]

h) and bit_rate[k] <= 1.2 * MaxBR for bit_rate_value[k] and cpb_size[k] <= 1.2 * MaxCPB, where bit_rate[k] is given by Equation E-13 and cpb_size[k] is given by Equation E-14 if nal_hrd_parameters_present_flag equal to 1. MaxBR and MaxCPB are specified in Table A-1. The bitstream shall satisfy these conditions for at least one value of k in the range 0 to cpb_cnt_minus1. [Ed. Note (GJS/AG): Consider when HRD parameters not present.]

i) required CPB buffer size is <= the level limit given

j) vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1

k) horizontal motion vector range does not exceed [-2048, 2047.75] in units of luma samples

l) number of motion vectors per two consecutive macroblocks in decoding order (including from the last macroblock of a slice to the first macroblock of the next slice in decoding order) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1.

m) number of bits of macroblock_layer( ) data for any macroblock is not greater than 128 + 2048 * ChromaFormatFactor. When entropy_coding_mode_flag is equal to 1, the number of bits for the current 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 current macroblock.

Table A-1 below gives the parameter 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)	Max CPB size MaxCPB (1000 bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of MVs 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
5	552 960	21 696	40 680.0	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 difference between consecutive removal times satisfies the constraint that the number of slices in coded picture n is <= MaxMBperSec * ($t_r(n) - t_r(n-1)$) ÷ SliceRate, where SliceRate is specified in Table A-3.

b) In bitstreams conforming to the Main and Extended profiles, direct_8x8_inference_flag shall be equal to 1 for the levels specified in Table A-3 for the Main profile and in Table A-4 for the Extended profile.

c) 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.

d) In bitstreams conforming to the Baseline and Extended profiles, ($xInt_{max} - xInt_{min} + 6$) * ($yInt_{max} - yInt_{min} + 6$) <= 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 num_sub_mb_part(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

- $xInt_{min}$ as the minimum value of $xInt_L$ among all luma sample predictions for the sub-macroblock
- $xInt_{max}$ as the maximum value of $xInt_L$ among all luma sample predictions for the sub-macroblock
- $yInt_{min}$ as the minimum value of $yInt_L$ among all luma sample predictions for the sub-macroblock
- $yInt_{max}$ as the maximum value of $yInt_L$ among all luma sample predictions for the sub-macroblock

For each level at which a numerical value of MaxSubMbRectSize is specified in Table A-2 for the Baseline profile and in Table A-4 for the Extended profile, the following constraint shall be true for each 8x8 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	-
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.

Table A-3 – Main profile level limits

Level number	SliceRate	MinLumaBiPredSize	direct_8x8_inference_flag
1	-	-	-
1.1	-	-	-
1.2	-	-	-
1.3	-	-	-
2	-	-	-
2.1	-	-	-
2.2	-	-	-
3	22	-	1
3.1	60	8x8	1
3.2	60	8x8	1
4	60	8x8	1
4.1	24	8x8	1
5	24	8x8	1
5.1	24	8x8	1

A.3.2.3 Extended Profile Limits

Table A-4 specifies limits for each level that are specific to bitstreams conforming to the Extended profile. Entries marked "-" in Table A-4 denote the absence of a corresponding limit.

Table A-4 – Extended profile level limits

Level number	MaxSubMbRectSize	MinLumaBiPredSize	direct_8x8_inference_flag
1	576	-	1
1.1	576	-	1
1.2	576	-	1
1.3	576	-	1
2	576	-	1
2.1	576	-	1
2.2	576	-	1
3	576	-	1
3.1	-	8x8	1
3.2	-	8x8	1
4	-	8x8	1
4.1	-	8x8	1
5	-	8x8	1
5.1	-	8x8	1

A.3.3 Effect of level limits on frame rate (informative)

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

Table A-5 – Maximum frame rates (frames per second) for some example picture sizes

Level number:					**1**	**1.1**	**1.2**	**1.3**	**2**	**2.1**	**2.2**
Max frame size (macroblocks):					**99**	**396**	**396**	**396**	**396**	**792**	**1 620**
Max macroblocks/second:					**1 485**	**3 000**	**6 000**	**11 880**	**11 880**	**19 800**	**20 250**
Max picture size (samples):					25 344	101 376	101 376	101 376	101 376	202 752	414 720
Max samples/second:					380 160	768 000	1 536 000	3 041 280	3 041 280	5 068 800	5 184 000
Format	**MBs Width**	**MBs Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	30.9	62.5	125.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	15.0	30.3	60.6	120.0	120.0	172.0	172.0
QVGA	**320**	**240**	300	76 800	-	10.0	20.0	39.6	39.6	66.0	67.5
525 SIF	**352**	**240**	330	84 480	-	9.1	18.2	36.0	36.0	60.0	61.4
CIF	**352**	**288**	396	101 376	-	7.6	15.2	30.0	30.0	50.0	51.1
525 HHR	**352**	**480**	660	168 960	-	-	-	-	-	30.0	30.7
625 HHR	**352**	**576**	792	202 752	-	-	-	-	-	25.0	25.6
VGA	**640**	**480**	1 200	307 200	-	-	-	-	-	-	16.9
525 4SIF	**704**	**480**	1 320	337 920	-	-	-	-	-	-	15.3
525 SD	**720**	**480**	1 350	345 600	-	-	-	-	-	-	15.0
4CIF	**704**	**576**	1 584	405 504	-	-	-	-	-	-	12.8
625 SD	**720**	**576**	1 620	414 720	-	-	-	-	-	-	12.5
SVGA	**800**	**600**	1 900	486 400	-	-	-	-	-	-	-
XGA	**1024**	**768**	3 072	786 432	-	-	-	-	-	-	-
720p HD	**1280**	**720**	3 600	921 600	-	-	-	-	-	-	-
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	-	-	-	-
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	-	-	-	-
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	-	-	-
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	-	-	-
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	-	-	-
1080 HD	**1920**	**1080**	8 160	2 088 960	-	-	-	-	-	-	-
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	-	-	-
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-

Table A-5 (concluded) – Maximum frame rates (frames per second) for some example picture sizes

Level number:					**3**	**3.1**	**3.2**	**4**	**4.1**	**5**	**5.1**
Max picture size (macroblocks):					**1 620**	**3 600**	**5 120**	**8 192**	**8 192**	**21 696**	**36 864**
Max macroblocks/second:					**40 500**	**108 000**	**216 000**	**245 760**	**245 760**	**552 960**	**983 040**
Max picture size (samples):					414 720	921 600	1 310 720	2 097 152	2 097 152	5 554 176	9 437 184
Max samples/second:					10 368 000	27 648 000	55 296 000	62 914 560	62 914 560	141 557 760	251 658 240
Format	**MBs Width**	**MBs Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	**320**	**240**	300	76 800	135.0	172.0	172.0	172.0	172.0	172.0	172.0
525 SIF	**352**	**240**	330	84 480	122.7	172.0	172.0	172.0	172.0	172.0	172.0
CIF	**352**	**288**	396	101 376	102.3	172.0	172.0	172.0	172.0	172.0	172.0
525 HHR	**352**	**480**	660	168 960	61.4	163.6	172.0	172.0	172.0	172.0	172.0
625 HHR	**352**	**576**	792	202 752	51.1	136.4	172.0	172.0	172.0	172.0	172.0
VGA	**640**	**480**	1 200	307 200	33.8	90.0	172.0	172.0	172.0	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	30.7	81.8	163.6	172.0	172.0	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	30.0	80.0	160.0	172.0	172.0	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	25.6	68.2	136.4	155.2	155.2	172.0	172.0
625 SD	**720**	**576**	1 620	414 720	25.0	66.7	133.3	151.7	151.7	172.0	172.0
SVGA	**800**	**600**	1 900	486 400	-	56.8	113.7	129.3	129.3	172.0	172.0
XGA	**1024**	**768**	3 072	786 432	-	35.2	70.3	80.0	80.0	172.0	172.0
720p HD	**1280**	**720**	3 600	921 600	-	30.0	60.0	68.3	68.3	153.6	172.0
4VGA	**1280**	**960**	4 800	1 228 800	-	-	45.0	51.2	51.2	115.2	172.0
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	42.2	48.0	48.0	108.0	172.0
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	46.5	46.5	104.7	172.0
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	38.8	38.8	87.3	155.2
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	32.8	32.8	73.7	131.1
1080 HD	**1920**	**1080**	8 160	2 088 960	-	-	-	30.1	30.1	67.8	120.5
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	30.0	30.0	67.5	120.0
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	45.0	80.0
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	28.8	51.2

The following should be noted.

- This is a variable-picture-size specification. The specific picture sizes in Table A-5 are illustrative examples only.
- As used in Table A-5, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region), and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

- XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.
- Frame rates given are correct for progressive scan modes, and for interlaced if the frame height is divisible by 32.

Annex B
Byte stream format

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

This annex specifies syntax and semantics of a byte stream format specified for use by systems that transmit some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as ITU-T Recommendation H.222.0 | ISO/IEC 13818-1 systems or ITU-T Recommendation H.320 systems. For bit-oriented transmission, the network bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal_unit(NumBytesInNALunit) syntax structure. It may (and under some circumstances, it shall) also contain some additional zero_byte syntax elements.

B.1 Byte stream NAL unit syntax and semantics

B.1.1 Byte stream NAL unit syntax

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
if(more_data_in_byte_stream( )) {		
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
}		
}		

B.1.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the syntax order of the NAL units contained in the byte stream NAL units (see subclause 7.4.1.1). The content of each byte stream NAL unit is associated with the same primary coded picture as the NAL unit contained in the byte stream NAL unit (see subclause 7.4.1.2).

zero_byte is a single byte equal to 0x00.

The minimum required number of zero_byte syntax elements preceding the start_code_prefix_one_3bytes is 1 when either of the following conditions are fulfilled:

– If the nal_unit_type within the nal_unit( ) is equal to 7 (sequence parameter set) or 8 (picture parameter set), or

– If the byte stream NAL unit syntax structure contains the first NAL unit associated with a primary coded picture in decoding order, as specified by subclause 7.4.1.2.

Any number of additional zero_byte syntax elements may immediately precede the start code prefix within the byte stream NAL unit syntax structure.

start_code_prefix_one_3bytes is a fixed-value string of 3 bytes equal to 0x000001.

B.2 Byte stream NAL unit decoding process

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder initialises its current position in the byte stream to the beginning of the byte stream.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream:

1. The decoder examines the byte stream, starting at the current position, to detect the location of the next byte-aligned three-byte sequence equal to 0x000001.

   NOTE – This three-byte sequence equal to 0x000001 is a start_code_prefix_one_3bytes syntax element, and all bytes starting at the current position in the byte stream and preceding the start_code_prefix_one_3bytes, if any, are zero_byte syntax elements equal to 0x00.

2. All bytes preceding and including this three-byte sequence are discarded and the current position in the byte stream is set to the position of the byte following this three-byte sequence.

3. NumBytesInNALunit is set equal to the number of byte-aligned bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of any of the following conditions:

   a. A subsequent byte-aligned three-byte sequence equal to 0x000000, or

   b. A subsequent byte-aligned three-byte sequence equal to 0x000001, or

   c. The end of the byte stream, as determined by unspecified means.

4. NumBytesInNALunit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNALunit bytes. This sequence of bytes is nal_unit(NumBytesInNALunit) and is decoded using the NAL unit decoding process.

B.3 Decoder byte-alignment recovery (informative)

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

Many applications provide data to a decoder in a manner that is inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this subclause.

If the decoder does not have byte alignment with the encoder's byte stream, the decoder can examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte. Upon detecting this pattern, the decoder will be byte aligned with the encoder.

Once byte aligned with the encoder, the decoder can examine the incoming byte stream for the three-byte sequences 0x000001 and 0x000003.

If the three-byte sequence 0x000001 is detected, this is a start code prefix.

If the three-byte sequence 0x000003 is detected, the third byte (0x03) is an emulation_prevention_three_byte to be discarded as specified in subclause 7.4.1.

The byte alignment detection procedure described in this subclause is equivalent to searching a byte sequence for three consecutive zero-valued bytes (0x000000), starting at any alignment position. Detection of this pattern indicates that the next non-zero byte contains the end of a start code prefix, and the first non-zero bit in that next non-zero byte is the last bit of an aligned byte.

Annex C
Hypothetical reference decoder

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

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

[Ed. Note: update subclause to reflect Pattaya meeting adoptions.]

This annex specifies the hypothetical reference decoder (HRD), which represents a set of normative requirements on bitstreams. These constraints shall be enforced by an encoder, and can be assumed by a decoder or multiplexor to be true.

This subclause specifies the normative requirements of the HRD.

Two types of bitstreams are subject to the HRD requirements of this Recommendation | International Standard: a stream of VCL-only NAL units (that includes all VCL NAL units for all coded pictures) called a VCL NAL unit stream, and any bitstream (either a byte stream or a NAL unit stream) that includes all VCL NAL units for all coded pictures and some non-VCL NAL units. Figure C-1 shows how each of these two bitstream types are constructed from the RBSPs.

Accordingly, two sets of HRD parameters are used. The HRD parameter sets are signalled through video usability information (subclauses E.1 and E.2), which is part of the sequence parameters sets.

In order to check conformance of a bitstream by the HRD, all sequence parameter sets and picture parameters sets referred to in the VCL NAL units, and appropriate buffering period and picture timing SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-VCL NAL units), or by other unspecified means.

The syntax elements of non-VCL NAL units (or their default values), required for the HRD, are specified in the semantic subclauses.

NOTE - When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. If the HRD information is not present, as is the case for all stand-alone VCL streams, conformance can only be verified if the HRD data is supplied by some means not specified in this Recommendation | International Standard.

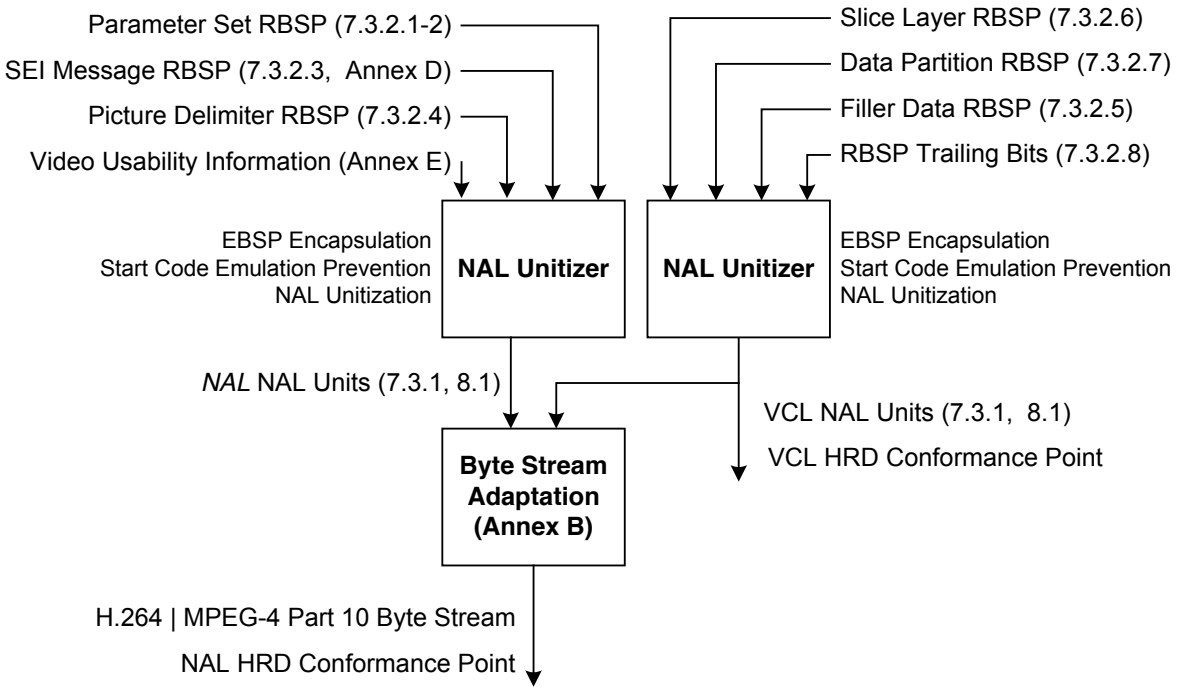


Figure C-1 – Structure of byte streams and NAL unit streams and HRD conformance points

The HRD contains a coded picture buffer (CPB), an instantaneous decoder and a decoded picture buffer (DPB) as shown in Figure C-2:

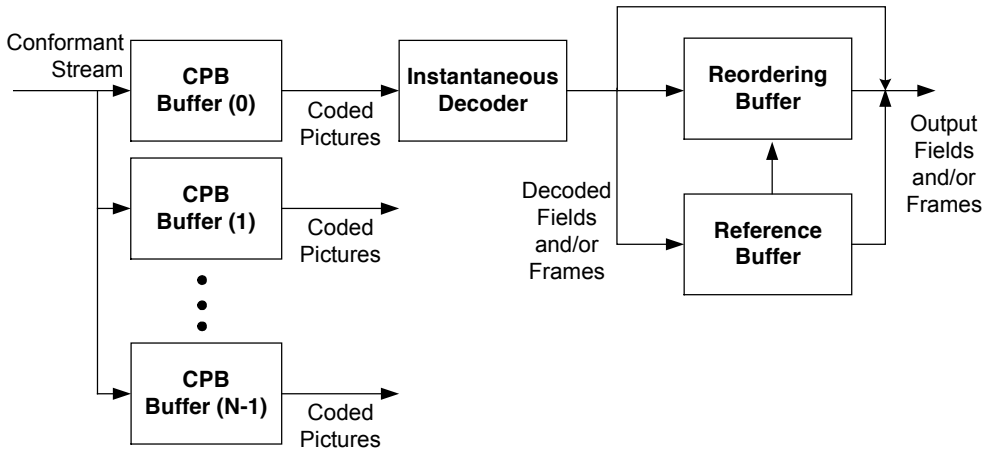


Figure C-2 – HRD buffer model

The HRD operates as follows: Data associated with primary coded pictures flow into the CPB according to a specified arrival schedule generated by the stream scheduler. The data associated with each coded picture is removed and decoded instantaneously by the instantaneous decoding process at CPB removal times. Each decoded picture is placed in the DPB at its CPB removal time. Finally, each picture is removed from the DPB as needed at the later of the DPB output time and the time that it is no longer needed as reference for decoding.

The operation of the CPB is specified in subclause C.1. The Instantaneous Decoder operation is specified in clauses 8 and 9. The decoding of a picture is instantaneous. The operation of the DPB is specified in subclause C.2.

HRD information concerning the number of CPBs and their associated bit rates and buffer sizes is specified in subclauses E.1.1, E.1.2, E.2.1 and E.2.2 as part of the video usability information of the sequence parameter set. The HRD is initialised as specified by the buffering period SEI message (subclauses D.1.1 and D.2.1). The removal timing of pictures from the CPB and output timing from the DPB are specified in the picture timing SEI message (subclauses D.1.2 and D.2.2). All timing information relating to a specific picture shall arrive prior to the CPB removal time of the first VCL NAL units associated with the picture.

The HRD is used to check conformance of bitstreams and decoders as specified in subclauses C.3 and C.4, respectively.

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

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

In the following description, let t_c = num_units_in_tick ÷ time_scale be specified as the clock tick. Also let be[t] and te[b] be the bit equivalent of a time t and the time equivalent of a number of bits b, with the conversion factor being the CPB arrival maximum bit rate.

C.1 Operation of coded picture buffer (CPB)

This specification applies independently to each set of CPB parameters within the HRD VUI portion of the sequence parameter set. The bits associated with each picture include all NAL units associated with the primary coded picture as specified in subclause 7.4.1.2. This description applies to both Type II and VCL conformance.

C.1.1 Timing of bitstream arrival

The HRD may be initialised by any buffering period SEI message. Prior to the first initialisation, the buffer is initially empty. The first bit associated with the picture that is associated with the buffering period SEI message begins to enter the buffer at initial arrival time $t_{ai}(0)$=0. The other bits associated with this first picture arrive continuously at a bit rate specified by bit_rate[k] associated with the CPB (see subclause E.2.2). The last bit of the first picture finishes arriving at final arrival time

$$t_{af}(0) = b(0) \div bit\_rate[k], \quad \text{(C-1)}$$

where b(n) is the size in bits of the n-th picture in coded order. The final arrival time for each picture is always the sum of the initial arrival time and the time required for the bits associated with that picture to enter the CPB:

$$t_{af}(n) = t_{ai}(n) + b(n) \div bit\_rate[k]. \quad \text{(C-2)}$$

If vbr_cbr_flag[k] is equal to 0, the initial arrival time for each subsequent picture n, is equal to the final arrival time of picture n-1, i.e.

$$t_{ai}(n) = t_{af}(n-1) \quad \text{(C-3)}$$

If vbr_cbr_flag[k] is equal to 1, the initial arrival time for each subsequent picture n, is

$$t_{ai}(n) = \max(t_{af}(n-1), t_{ai,earliest}(n)) \quad \text{(C-4)}$$

where:

$$t_{ai,earliest}(n) = t_c * cpb\_removal\_delay(n) - (initial\_cpb\_removal\_delay\_offset \div 90000) \quad \text{(C-5)}$$

See subclauses D.1.1, D.1.2, D.2.1 and D.2.2 for the syntax and semantics of cpb_removal_delay and initial_cpb_removal_delay_offset.

C.1.2 Timing of coded picture removal

For the first picture after HRD initialisation, the coded data associated with the picture is removed from the CPB at a removal time equal to the following:

$$t_r(0) = initial\_cpb\_removal\_delay \div 90000 \quad \text{(C-6)}$$

where initial_cpb_removal_delay is specified in the buffering period SEI message.

After the first picture is removed, the buffer is examined at subsequent points of time, each of which is delayed from the previous one by an integer multiple of the clock tick t_c.

The removal time $t_r(n)$ of coded data for picture n is delayed with respect to that of picture 0; the delay is equal to the time specified in the cpb_removal_delay syntax element present in the picture timing SEI Message.

$$t_r(n) = t_r(0) + t_c * cpb\_removal\_delay(n) \quad \text{(C-7)}$$

At this time, the coded data for coded picture n is removed from the CPB.

If low_delay_hrd_flag is equal to 1, and if the amount of coded data for picture n, b(n), is so large that it prevents removal at the computed removal time, $t_{r,d}(n, m^*)$, given by

$$t_{r,d}(n,m^*) = t_r(0) + t_c * m^*, \quad \text{(C-8)}$$

where m^* is such that $t_{r,d}(n, m^*-1) < t_{af}(n) \leq t_{r,d}(n, m^*)$. This is an aspect of low-delay operation.

C.2 Operation of the decoded picture buffer (DPB)

The decoded picture buffer contains frame stores. Each of the frame stores may contain a decoded frame, a decoded complementary field pair or a single (non-paired) decoded field which are to be used for reference in future decoding (reference pictures) and/or are held for future output (reordered and delayed pictures). At $t_r(0)$, the DPB fullness, measured in frames, is initialised to 0. The following four steps all happen instantaneously at $t_r(n)$, the CPB removal time of picture n, and in the sequence listed.

C.2.1 Picture decoding

Picture n is decoded and its DPB output time $t_{o,dpb}(n)$ is computed by

$$t_{o,dpb}(n) = t_r(n) + t_c * dpb\_output\_delay(n) \quad (C\text{-}9)$$

If n is not equal to 0, then the value of $\Delta t_{o,dpb}(n)$ shall be computed as:

$$\Delta t_{o,dpb}(n) = t_{o,dpb}(n) - t_{o,dpb}(n - 1) \quad (C\text{-}10)$$

The decoded picture is temporarily stored (not in the DPB).

If the decoded picture is an IDR picture, and no_output_of_prior_pics_flag is equal to 1, all frame stores in the DPB are emptied and DPB fullness is reset to 0. If the decoded picture is an IDR picture, and no_output_of_prior_pics_flag is equal to 0, the reference pictures in the DPB are marked as specified in subclause 8.2.7.

C.2.2 Pictures output

Any picture m in the DPB with $t_{o,dpb}(m) <= t_r(n)$ that are not marked as "non-existing" are output from the HRD. If, for the current decoded picture $t_{o,dpb}(m) = t_r(n)$, it is also output. The order of output, if more than one picture is output is according to the values of the associated picture order count. The output pictures are cropped using the cropping rectangle specified in the picture parameter set for the picture.

C.2.3 Reference picture marking and picture removal (without output)

The reference pictures in the DPB are marked as specified in subclause 8.2.7.

Some pictures are removed from the DPB. Picture m is removed if it satisfies both of the following rules:

- It has been marked as “unused for reference” or "non-reference" during the decoding process of previously decoded pictures. If it is a reference frame picture, both its fields have to be marked as "unused for reference" in order to be removed from the DPB
- It is marked as "non-existing" or its DPB output time is less than or equal to the CPB removal time of the current picture; i.e., $t_{o,dpb}(m) \leq t_r(n)$

Following the removal of any frame or the removal of the last field in a frame buffer, the DPB fullness is decremented by one.

C.2.4 Current decoded picture marking and storage

C.2.4.1 Storage of a reference decoded picture into the DPB

The decoded picture is stored in the DPB and marked as “used for short-term reference” or "used for long-term reference" as specified in subclause 8.2.7. If the current decoded picture is not a second field of a complementary reference field pair, it is stored into an empty frame buffer and the DPB fullness is incremented. If the current decoded picture is a second field of a complementary reference field pair, it is stored in the same frame buffer as the previous decoded field.

C.2.4.2 Storage of a non-reference picture into the DPB

If the DPB output time of the current decoded picture exceeds its CPB removal time, it is stored in the DPB. If the current decoded picture is not a second field of a complementary non-reference field pair, it is stored in an empty frame buffer and the DPB fullness is incremented. If the current decoded picture is a second field (in decoding order) of a complementary non-reference field pair, it is stored in the same frame buffer as the first field of the pair.

C.3 Bitstream conformance

A bitstream of coded data conforming to this Recommendation | International Standard fulfills the following requirements.

The bitstream is constructed according to the syntax and semantics specified in clause 7.

VCL NAL unit streams shall fulfill the HRD requirement for at least one k of the VCL HRD parameters with input bit rate and CPB size as constrained by Annex D and by Annex A for the Profile and Level specified in the bitstream.

A type II bitstream shall fulfill the HRD requirement for at least one k of the Type II HRD parameters. In addition, it shall fulfill the requirement for at least one k of the VCL HRD parameters with input bit rate and CPB size as constrained by Annex D and by Annex A for the profile and level specified in the bitstream, where, for the input bit rate and CPB storage, only VCL NAL units of the bitstream are counted.

- **Removal time consistency.** For each picture, the removal times $t_r(n)$ from the CPB computed using different buffering periods SEI messages as starting points for conformance verification shall be consistent to within the precision of the clocks used (90 kHz clock used for initial removal time, t_c clock used for subsequent removal time calculations, and bit_rate[k] used for arrival times).

  Note - This can be ensured at the encoder by computing the initial CPB removal delay for a buffering period SEI message from the arrival and removal times computed using Equations C-4 and C-6. That is, if the last picture before the buffering period SEI message is picture n-1, then the initial CPB removal delay for the next buffering period SEI message could be computed by

$$\text{initial\_cpb\_removal\_delay} = 90000 * (t_r(n) - t_r(n - 1) + t_{ai}(n)) \quad \text{(C-11)}$$

- **CPB Underflow and Overflow Prevention.** The CPB shall never overflow or underflow.

  NOTE - In terms of the arrival and removal schedules, this means that, with the exception of some pictures in low-delay mode that are described below, all bits from a picture must be in the CPB at the picture's computed removal time $t_r(n)$. In other words, its final arrival time must be no later than its removal time: $t_{af}(n) \leq t_r(n)$. Further, the removal time $t_r(n)$ must be no later than the time-equivalent of the buffer size te[cpb_size[k]]. Note that this prevents both underflow and overflow.

- **Big Picture Removal Time, CPB Overflow Prevention and Resynchronisation of Underflow Prevention.** If the final arrival time $t_{af}(n)$ of picture n to the CPB exceeds its computed removal time $t_r(n)$, its size must be such that it can be removed from the buffer without overflow at $t_{r,d}(n,m^*)$ as specified above.

- **Maximum Removal Rate from the CPB.** The difference between consecutive removal times, $t_r(n) - t_r(n-1)$, shall not be less than the ratio of the number of macroblocks in picture n to the maximum MB processing rate for the bitstream's level in macroblocks per second (see subclause A.3.1).

- **DPB Overflow Prevention.** Immediately after any picture is added to the DPB, the fullness of the DPB shall be less than or equal to the DPB size as constrained by Annex D and Annex A for the profile and level specified in the bitstream.

- **Maximum Output Rate from the DPB.** The value of $\Delta t_{o,dpb}(n)$ as given by Equation C-10, which is the difference between the output time of a picture and that of the picture output immediately preceding it, shall be larger than the ratio of the number of macroblocks in the picture to the maximum macroblock processing rate for the bitstream's Level in macroblocks per second (see subclause A.3.1).

C.4 Decoder conformance

A conformant decoder is conformant to at least one profile and one level. A decoder claiming conformance to a specific profile and level shall be able decode successfully all conforming bitstreams specified for decoder conformance in Annex A, provided that all sequence parameter sets and picture parameters sets referred to in the VCL NAL units, and appropriate buffering period and picture timing SEI messages shall be conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified by this Recommendation | International Standard.

There are two types of conformance that can be claimed by a decoder: Output timing conformance and output order conformance.

To check a decoder conformance, test bitstreams conforming to the claimed profile and level are delivered by a bitstream delivery scheduler as described above, both to an HRD and to the decoder under test (DUT). The output of both decoders shall be the same. I.e., all pictures output by the HRD are also output by the DUT and all decoded sample values are the same.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both HRD and the DUT up to a fixed delay.

For output order decoder conformance, an HRD as described below is used and the order of picture output is the same for both HRD and the DUT.

C.4.1 Operation of the output order DPB

The decoded picture buffer contains frame buffers. Each of the frame buffers may contain a decoded frame, a decoded complementary field pair [Ed. Note (AG): Need a def for a complementary non-reference field pair] or a single (non-paired) decoded field that are to be used for reference in future decoding (reference pictures) and/or are held for future output (reordered pictures).

C.4.2 Picture decoding

Picture n is decoded and is temporarily stored (not in the DPB).

If the decoded picture is an IDR picture, and non_ref_pic_reset_flag is equal to 1, all frame buffers in the DPB are emptied (but not output) and DPB fullness is reset to 0. If the decoded picture is an IDR picture, and non_ref_pic_reset_flag is equal to 0, the reference pictures in the DPB are marked as specified in subclause 8.2.7.

If the decoded picture is an IDR picture and non_ref_pic_reset_flag is equal to 0, the pictures in the DPB are output in ascending order of PicOrder, all frame buffers in the DPB are emptied and DPB fullness is reset to 0. [Ed. Note (Miska): Added this last sentence to deal with picture order in the neighbourhood of IDR pictures.]

C.4.3 Reference picture marking

The reference pictures in the DPB are marked as specified in subclause 8.2.7.

C.4.4 Current decoded picture marking and storage

PicOrder is stored with each decoded picture: For a decoded frame, its FrameOrderCnt is stored, for a decoded top field, its TopFieldOrderCnt is stored and for a decoded bottom field, its BottomFieldOrderCnt is stored.

C.4.4.1 Storage of a reference decoded picture into the DPB

The decoded picture is added to the decoded picture buffer and marked as “used for short-term reference” or "used for long-term reference", as specified in subclause 8.2.7, and "needed for output". If the current decoded picture is not a second field of a complementary reference field pair, it is stored in an empty frame buffer and the DPB fullness is incremented. If there is no empty frame buffer (i.e., DPB capacity is equal to DPB size), one is emptied by the 'bumping" process described below. If the current decoded picture is a second field (in decoding order) of a complementary reference field pair, it is stored in the same frame buffer as the first decoded field.

C.4.4.2 Storage and marking of a non-reference decoded picture into the DPB

The decode picture is marked as "non-reference" and "needed for output". If the current decoded picture is not a second field of a complementary non-reference field pair, it is stored in an empty frame buffer and the DPB fullness is incremented. If there is no empty frame buffer (i.e., DPB capacity is equal to DPB size), one is emptied by the 'bumping" process described below. If the current decoded picture is a second field (in decoding order) of a complementary non-reference field pair, it is stored in the same frame buffer as the first field of the pair.

C.4.4.3 "Bumping" process

The "bumping" operation, when an empty frame buffer is needed for a (non IDR) picture, is executed with the following steps:

a) When an empty frame buffer is needed for a non-reference picture, if this picture has a lower value of POC than all fields or frames in the DPB marked as "needed for output" , it is output instead of being stored and the bumping operation is terminated.

b) The field or frame marked as "needed for output" that has a lowest value of POC of all fields or frames in the DPB marked as "needed for output", is output, and marked as "not needed for output". [Ed.Note(Miska): This does not take into account that the DPB may contain pictures from multiple video sequences. The output order of pictures in different sequences cannot be resolved by POC but rather it is determined by the decoding order relative to the previous IDR picture. See C.4.2 for a proposed solution.]

c) The frame buffer that included the field or frame output in step (b) is checked, and if one of the following conditions is satisfied, the frame buffer is emptied, DPB fullness is decremented and the bumping operation is terminated. Otherwise, steps (b) and (c) are repeated until termination.

c1) The frame buffer includes a non-reference frame or a non-reference non-paired field

c2) The frame buffer includes a complementary non-reference field pair with both fields marked as "not needed for output".

c3) The frame buffer includes a non-paired reference field marked as "unused for reference".

c4) The frame buffer includes a reference frame with both fields marked as "unused for reference".

c5) The frame buffer includes a complementary reference field pair with both fields marked as "unused for reference" and "not needed for output".

Annex D
Supplemental enhancement information

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

This annex specifies syntax and semantic for supplemental enhancement information that provides data constructs that are synchronous with the video data content. Each sei_payload( ) specifies payloadType and payloadSize parameters. This annex specifies supplemental enhancement information (SEI) that provides a data delivery mechanism construct that is delivered synchronous with the video data content. SEI assists in the processes related to decoding, display or other purposes. SEI is not required for reconstructing the luma or chroma samples by a video decoder, and decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex C for the definition of output order conformance). If present in the bitstream, SEI messages shall follow the syntax and semantics specified in this annex. If the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex.

[Ed. Note (GJS): Should check on scope/persistence issues for SEI, in particular things that have a scope/persistence of more than one picture and things that have a scope of only a slice.]

D.1 SEI payload syntax

sei_payload(payloadType, payloadSize) {	C	Descriptor
if(payloadType = = 0)		
buffering_period(payloadSize)	5	
else if(payloadType = = 1)		
pic_timing(payloadSize)	5	
else if(payloadType = = 2)		
pan_scan_rect(payloadSize)	5	
else if(payloadType = = 3)		
filler_payload(payloadSize)	5	
else if(payloadType = = 4)		
user_data_registered_itu_t_t35(payloadSize)	5	
else if(payloadType = = 5)		
user_data_unregistered(payloadSize)	5	
else if(payloadType = = 6)		
recovery_point(payloadSize)	5	
else if(payloadType = = 7)		
dec_ref_pic_marking_repetition(payloadSize)	5	
else if(payloadType = = 8)		
spare_pic(payloadSize)	5	
else if(payloadType = = 9)		
scene_info(payloadSize)	5	
else if(payloadType = = 10)		
sub_seq_info(payloadSize)	5	
else if(payloadType = = 11)		
sub_seq_layer_characteristics(payloadSize)	5	
else if(payloadType = = 12)		
sub_seq_characteristics(payloadSize)	5	
else if(payloadType = = 13)		
full_frame_freeze(payloadSize)	5	
else if(payloadType = = 14)		
full_frame_freeze_release(payloadSize)	5	
else if(payloadType = = 15)		
full_frame_snapshot(payloadSize)	5	
else if(payloadType = = 16)		
progressive_refinement_segment_start(payloadSize)	5	
else if(payloadType = = 17)		
progressive_refinement_segment_end(payloadSize)	5	
else if(payloadType = = 18)		
motion_constrained_slice_group_set(payloadSize)	5	
else		
reserved_sei_message(payloadSize)	5	
if(!byte_aligned( )) {		
bit_equal_to_one /* equal to 1 */	5	f(1)
while(!byte_aligned( ))		
bit_equal_to_zero /* equal to 0 */	5	f(1)
}		
}		

D.1.1 Buffering period SEI message syntax

buffering_period(payloadSize) {	C	Descriptor
seq_parameter_set_id	5	ue(v)
if(nal_hrd_parameters_present_flag == 1) {		
for(k = 0; k <= cpb_cnt_minus1; k++) {		
initial_cpb_removal_delay[k **]**	5	u(v)
initial_cpb_removal_delay_offset[k **]**	5	u(v)
}		
}		
if(vcl_hrd_parameters_present_flag == 1) {		
for(k = 0; k <= cpb_cnt_minus1; k++) {		
initial_cpb_removal_delay[k **]**	5	u(v)
initial_cpb_removal_delay_offset[k **]**	5	u(v)
}		
}		
}		

D.1.2 Picture timing SEI message syntax

pic_timing(payloadSize) {	C	Descriptor
if(nal_hrd_parameters_present_flag || vcl_hrd_parameters_present_flag) {		
cpb_removal_delay	**5**	u(v)
dpb_output_delay	**5**	u(v)
}		
pic_struct_present_flag	**5**	u(1)
if(pic_struct_present_flag) {		
pic_struct	**5**	u(4)
for(i = 0; i < NumClockTS ; i++) {		
clock_timestamp_flag[i]	**5**	u(1)
if (clock_timestamp_flag[i]) {		
ct_type	**5**	u(2)
nuit_field_based_flag	**5**	u(1)
counting_type	5	u(5)
full_timestamp_flag	5	u(1)
discontinuity_flag	5	u(1)
cnt_dropped_flag	5	u(1)
nframes	5	u(8)
if(full_timestamp_flag) {		
seconds_value /* 0..59 */	5	u(6)
minutes_value /* 0..59 */	5	u(6)
hours_value /* 0..23 */	5	u(5)
} else {		
seconds_flag	5	u(1)
if(seconds_flag) {		

seconds_value /* range 0..59 */	5	u(6)
minutes_flag	5	u(1)
if(minutes_flag) {		
minutes_value /* 0..59 */	5	u(6)
hours_flag	5	u(1)
if(hours_flag)		
hours_value /* 0..23 */	5	u(5)
}		
}		
}		
if(time_offset_length > 0)		
time_offset	5	i(v)
}		
}		
}		
}		

D.1.3 Pan-scan rectangle SEI message syntax

[Ed. Note (GJS): Needs persistence indicator and multiple pan-scan count. See disposition notes for JVT-F013.]

pan_scan_rect(payloadSize) {	**C**	**Descriptor**
pan_scan_rect_id	5	ue(v)
pan_scan_rect_left_offset	5	se(v)
pan_scan_rect_right_offset	5	se(v)
pan_scan_rect_top_offset	5	se(v)
pan_scan_rect_bottom_offset	5	se(v)
}		

D.1.4 Filler payload SEI message syntax

filler_payload(payloadSize) {	**C**	**Descriptor**
for(k = 0; k < payloadSize; k++)		
ff_byte /* equal to 0xFF */	5	f(8)
}		

D.1.5 User data registered by ITU-T Recommendation T.35 SEI message syntax

user_data_registered_itu_t_t35(payloadSize) {	C	Descriptor
itu_t_t35_country_code	5	b(8)
if(itu_t_t35_country_code != 0xFF)		
i = 1		
else {		
itu_t_t35_country_code_extension_byte	5	b(8)
i = 2		
}		
do {		
itu_t_t35_payload_byte	5	b(8)
i++		
} while(i < payloadSize)		
}		

D.1.6 User data unregistered SEI message syntax

user_data_unregistered(payloadSize) {	C	Descriptor
uuid_iso_11578	**5**	b(128)
for(i = 16; i < payloadSize; i++)		
user_data_payload_byte	5	b(8)
}		

D.1.7 Recovery point SEI message syntax

recovery_point(payloadSize) {	C	Descriptor
recovery_frame_cnt	5	ue(v)
exact_match_flag	5	u(1)
broken_link_flag	5	u(1)
changing_slice_group_idc	5	u(2)
}		

D.1.8 Decoded reference picture marking repetition SEI message syntax

dec_ref_pic_marking_repetition(payloadSize) {	C	Descriptor
original_idr_flag	5	u(1)
original_frame_num	5	ue(v)
if(!frame_mbs_only_flag) {		
original_field_pic_flag	5	u(1)
if(original_field_pic_flag)		
original_bottom_field_flag	5	u(1)
}		
dec_ref_pic_marking( )	5	
}		

D.1.9 Spare picture SEI message syntax

spare_pic(payloadSize) {	C	Descriptor
target_frame_num	5	ue(v)
spare_field_flag	5	u(1)
if(spare_field_flag)		
target_bottom_field_flag	5	u(1)
num_spare_pics_minus1	5	ue(v)
for(i = 0; i < num_spare_pics_minus1+1; i++) {		
delta_spare_frame_num[i **]**	5	ue(v)
if(spare_field_flag)		
spare_bottom_field_flag[i **]**	5	u(1)
spare_area_idc[i **]**	5	ue(v)
if(spare_area_idc[i] == 1)		
for(j = 0; j < PicSizeInMapUnits; j++)		
spare_unit_flag[i **][** j **]**	5	u(1)
else if(spare_area_idc[i] == 2) {		
mapUnitCnt = 0, j = 0		
do {		
zero_run_length[i **][** j **]**	5	ue(v)
mapUnitCnt += zero_run_length[i][j++] + 1		
} while(mapUnitCnt< PicSizeInMapUnits)		
}		
}		
}		

D.1.10 Scene information SEI message syntax

scene_info(payloadSize) {	C	Descriptor
scene_info_known_flag	5	u(1)
if(scene_info_known_flag) {		
scene_id	5	ue(v)
scene_transition_type	5	ue(v)
if(scene_transition_type > 3)		
second_scene_id	5	ue(v)
}		
}		

D.1.11 Sub-sequence information SEI message syntax

sub_seq_info(payloadSize) {	C	Descriptor
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
first_ref_pic_flag	5	u(1)
leading_non_ref_pic_flag	5	u(1)
last_pic_flag	5	u(1)
sub_seq_frame_num_flag	5	u(1)
if(sub_seq_frame_num_flag)		
sub_seq_frame_num	5	ue(v)
}		

D.1.12 Sub-sequence layer characteristics SEI message syntax

sub_seq_layer_characteristics(payloadSize) {	C	Descriptor
num_sub_seq_layers_minus1	5	ue(v)
for(layer = 0; layer <= num_sub_seq_layers_minus1; layer++) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
}		

D.1.13 Sub-sequence characteristics SEI message syntax

sub_seq_characteristics(payloadSize) {	C	Descriptor
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
duration_flag	5	u(1)
if(duration_flag)		
sub_seq_duration	5	u(32)
average_rate_flag	5	u(1)
if(average_rate_flag) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
num_referenced_subseqs	5	ue(v)
for(n = 0; n < num_referenced_subseqs; n++) {		
ref_sub_seq_layer_num	5	ue(v)
ref_sub_seq_id	5	ue(v)
ref_sub_seq_direction	5	u(1)
}		
}		

D.1.14 Full-frame freeze SEI message syntax

full_frame_freeze(payloadSize) {	C	Descriptor
}		

D.1.15 Full-frame freeze release SEI message syntax

full_frame_freeze_release(payloadSize) {	C	Descriptor
}		

D.1.16 Full-frame snapshot SEI message syntax

full_frame_snapshot(payloadSize) {	C	Descriptor
snapshot_id	5	ue(v)
}		

D.1.17 Progressive refinement segment start SEI message syntax

progressive_refinement_segment_start(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
num_refinement_steps_minus1	5	ue(v)
}		

D.1.18 Progressive refinement segment end SEI message syntax

progressive_refinement_segment_end(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
}		

D.1.19 Motion-constrained slice group set SEI message syntax

motion_constrained_slice_group_set(payloadSize) {	C	Descriptor
num_slice_groups_in_set_minus1	5	ue(v)
for(i = 0; i <= num_slice_groups_in_set_minus1; i++)		
slice_group_id[i **]**	5	u(v)
exact_match_flag	5	u(1)
pan_scan_rect_flag	5	u(1)
if(pan_scan_rect_flag == 1)		
pan_scan_rect_id	5	ue(v)
}		

D.1.20 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	C	Descriptor
for(i = 0; i < payloadSize; i++)		
reserved_sei_message_payload_byte	5	b(8)
}		

D.2 SEI payload semantics

D.2.1 Buffering period SEI message semantics

If nal_hrd_parameters_present_flag and vcl_hrd_parameters_present_flag are both equal to 0, no buffering period SEI messages shall be present in the bitstream for the sequence.

Otherwise, a buffering period SEI message can be associated with any picture in the bitstream, and a buffering period SEI message shall be associated with each IDR picture and with each picture associated with a recovery point SEI message.

NOTE – For some applications, the frequent presence of a buffering period SEI message may be desirable.

A buffering period is specified as the set of pictures between two instances of the buffering period SEI message.

seq_parameter_set_id specifies the sequence parameter set that contains the sequence HRD attributes. The value of seq_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referenced by the picture associated with the buffering period SEI message. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

initial_cpb_removal_delay: This syntax element represents the delay between the time of arrival in the CPB of the first bit of the coded data associated with the picture associated with the buffering period SEI message and the time of removal from the CPB of the coded data associated with the same picture. The syntax element is a fixed length code whose length in bits is given by initial_cpb_removal_delay_length_minus1 + 1. It is in units of a 90 kHz clock. The initial_cpb_removal_delay syntax element is used in conjunction with the CPBs as specified in Annex C. initial_cpb_removal_delay shall not be equal to 0 and shall not exceed the time-equivalent of the CPB size.

initial_cpb_removal_delay_offset: This syntax element is used in combination with the cpb_removal_delay to compute the initial delivery time of coded pictures to the CPB. It is in units of a 90 kHz clock. The initial_cpb_removal_delay_offset syntax element is a fixed length code whose length in bits is given by initial_cpb_removal_delay_length_minus1+1. This syntax element is not used by decoders and is needed only for the delivery scheduler described in Annex C.

Over the entire sequence, the sum of initial_cpb_removal_delay and initial_cpb_removal_delay_offset shall be constant and shall not exceed the time-equivalent of the CPB size.

D.2.2 Picture timing SEI message semantics

If nal_hrd_parameters_present_flag and vcl_hrd_parameters_present_flag are both equal to 0, no picture timing SEI messages shall be present in the bitstream for the sequence.

Otherwise, a picture timing SEI Message shall be associated with each picture subsequent to the first picture in the bitstream.

cpb_removal_delay specifies how many clock ticks (see Annex C) to wait after removal from the HRD CPB of the picture associated with the most recent buffering period SEI message before removing from the buffer the picture data associated with the picture timing SEI message. This value is also used to calculate an earliest possible time of arrival of picture data into the CPB for the delivery scheduler, as specified in Annex C. The syntax element is a fixed length code whose length in bits is given by cpb_removal_delay_length_minus1+1. The cpb_removal_delay is the remainder of a $2^{(\text{cpb\_removal\_delay\_length\_minus1} + 1)}$ counter.

dpb_output_delay is used to compute the DPB output time. It specifies how many clock ticks to wait after removal of a picture from the CPB before it can be output from the DPB (see Annex C).

NOTE - A picture is not removed from the DPB at its output time if it is still marked as "used for short-term reference" or "used for long-term reference".

NOTE - Only one dpb_output_delay is specified for a decoded picture.

The size of the syntax element dpb_output_delay shall remain constant for the sequence and is given in bits by dpb_output_delay_length_minus1+1.

The picture output order implied by the values of this syntax element shall not contradict the order implied by the values of PicOrder associated with the same picture (see subclause C.4.4). [Ed. Note (GJS): Need to reword to avoid vagueness of "not contradict".]

pic_struct_present_flag: If and only if this syntax element is equal to 1, the pic_struct syntax element is present and follows immediately. [Ed. Note (AG/GJS): Clarify relation to timing, POC, etc.]

pic_struct: This parameter indicates whether a picture should be displayed as a frame or one or more fields, according to Table D-1.

Table D-1 – Interpretation of pic_struct

Value	Indicated display of picture	Restrictions	NumClockTS
0	frame	field_pic_flag shall be 0	1
1	top field	field_pic_flag shall be 1, bottom_field_flag shall be 0	1
2	bottom field	field_pic_flag shall be 1, bottom_field_flag shall be 1	1
3	top field, bottom field, in that order	field_pic_flag shall be 0	2
4	bottom field, top field, in that order	field_pic_flag shall be 0	2
5	top field, bottom field, top field repeated, in that order	field_pic_flag shall be 0	3
6	bottom field, top field, bottom field repeated, in that order	field_pic_flag shall be 0	3
7	frame doubling	field_pic_flag shall be 0	2
8	frame tripling	field_pic_flag shall be 0	3
9..15	reserved		

NumClockTS is determined by pic_struct as specified in Table D-1. There are up to NumClockTS sets of clock timestamp information for a picture, as specified by clock_timestamp_flag[i] for each set. The sets of clock timestamp information apply to the field(s) or the frame associated with the picture by pic_struct. In the case of fields, the order of the timestamps is the same as the output order of the fields, as specified by pic_struct. Two fields associated with a frame may have different values of ct_type. If EquivalentTimestamp is equal for two fields of opposite parity that are adjacent in output order, both with ct_type equal to 0 (progressive) or ct_type equal to 2 (unknown), then the two fields are specified to have come from the same original frame. Two adjacent fields in output order shall have different values of EquivalentTimestamp if the value of ct_type for either field is 1 (interlaced).

The contents of the clock timestamp SEI message specify an ideal output time computed as

$$\begin{aligned} \text{EquivalentTimestamp} = &((HH * 60 + MM) * 60 + SS) * time\_scale + \\ &NF * (num\_units\_in\_tick * (1 + nuit\_field\_based\_flag)) + TO, \end{aligned} \quad \text{(D-1)}$$

in units of ticks of a clock with clock frequency equal to time_scale Hz. The EquivalentTimestamp is the ideal output time assuming a display that can display frames and fields at any time without regard to a frame or field rate. [Ed.Note: (TW) no assumptions in normative text] If two or more frames with pic_struct equal to 0 are adjacent in output order and have equal values of EquivalentTimestamp, the ideal display duration for all such frames except for the last frame in output order is 0, and ideally only the last frame is displayed. Note that nframes and NF are frame-based counts.

Frame doubling indicates that the frame should be displayed two times consecutively and frame tripling indicates that the frame should be displayed three times consecutively. Frame doubling and frame tripling can be used for display of progressive frame rates.

NOTE - Frame doubling facilitates for the decoder to construct and display for example 50p from 25p broadcasted and 59.94p from 29.97p. Using frame doubling and frame tripling in combination on every other frame facilitates to display and reconstruct 59.94p from 23.98p.

clock_timestamp_flag[i]: If this syntax element is equal to 1, a number of clock timestamp fields are present and follow immediately.

ct_type: This parameter indicates the type (interlaced or progressive) of the original material as follows:

Table D-2 – Mapping of ct_type to original picture scan

Value	Original picture scan
0	progressive
1	interlaced
2	unknown
3	reserved

nuit_field_based_flag: Used in calculating EquivalentTimestamp, as specified in Equation D-1.

counting_type: Specifies the method of dropping values of the nframes parameter as specified in Table D-3.

Table D-3 – Definition of counting_type values

Value	Interpretation
0	no dropping of nframes count values and no use of time_offset
1	no dropping of nframes count values
2	dropping of individual zero values of nframes count
3	dropping of individual MaxFPS-1 values of nframes count
4	dropping of the two lowest (value 0 and 1) nframes counts when seconds_value is equal to 0 and minutes_value is not an integer multiple of 10
5	dropping of unspecified individual nframes count values
6	dropping of unspecified numbers of unspecified nframes count values
7..31	reserved

full_timestamp_flag specifies whether the nframes parameter is followed by seconds_value or seconds_flag.

discontinuity_flag specifies whether the time difference between the current value of EquivalentTimestamp and the value of EquivalentTimestamp computed from the previously clock timestamp in decoding order can be interpreted as a true time difference. A value of 0 specifies that the difference represents a true time difference.

cnt_dropped_flag specifies the skipping of one or more values of nframes using the counting method specified by counting_type.

nframes specifies the value of *NF* used to compute the EquivalentTimestamp. nframes shall be less than

$$\text{MaxFPS} = \text{Ceil}(\text{time\_scale} \div \text{num\_units\_in\_tick}) \quad \text{(D- 2)}$$

If counting_type is equal to 2 and cnt_dropped_flag is equal to 1, nframes shall be equal to 1 and the value of nframes for the previous picture in output order shall not be equal to 0 unless discontinuity_flag is equal to 1.

NOTE – When counting_type is equal to 2, the need for increasingly large magnitudes of *TO* in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 25 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value nframes equal to 0 when counting (e.g., counting nframes from 0 to 12, then incrementing seconds_value and counting nframes from 1 to 12, then incrementing seconds_value and counting nframes from 0 to 12, etc.).

If counting_type is equal to 3 and cnt_dropped_flag is equal to 1, nframes shall be equal to 0 and the value of nframes for the previous picture in output order shall not be equal to MaxFPS – 1 unless discontinuity_flag is equal to 1.

NOTE – When counting_type is equal to 3, the need for increasingly large magnitudes of *TO* in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 25 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value nframes equal to MaxFPS when counting (e.g., counting nframes from 0 to 12, then incrementing seconds_value and counting nframes from 0 to 11, then incrementing seconds_value and counting nframes from 0 to 12, etc.).

If counting_type is equal to 4 and cnt_dropped_flag is equal to 1, nframes shall be equal to 2 and the specified value of *SS* shall be zero and the specified value of *MM* shall not be an integer multiple of ten and nframes for the previous picture in output order shall not be equal to 0 or 1 unless discontinuity_flag is equal to 1.

NOTE – When counting_type is equal to 4, the need for increasingly large magnitudes of *TO* in Equation D-1 when using fixed non-integer frame rates (e.g., 30000÷1001 frames per second with time_scale equal to 60000 and num_units_in_tick equal to 1 001 and nuit_field_based_flag equal to 1) can be reduced by occasionally skipping over the value nframes equal to MaxFPS when counting (e.g., counting nframes from 0 to 29, then incrementing seconds_value and counting nframes from 0 to 29, etc., until the seconds_value is zero and minutes_value is not an integer multiple of ten, then counting nframes from 2 to 29, then incrementing seconds_value and counting nframes from 0 to 29, etc.). This counting method is well known in industry and is often referred to as "NTSC drop-frame" counting.

If counting_type is equal to 5 or 6 and cnt_dropped_flag is equal to 1, nframes shall not be equal to 1 plus the value of nframes for the previous picture in output order modulo MaxFPS unless discontinuity_flag is equal to 1.

NOTE – When counting_type is equal to 5 or 6, the need for increasingly large magnitudes of *TO* in Equation D-1 when using fixed non-integer frame rates can be avoided by occasionally skipping over some values of nframes when counting. The specific values of nframes that are skipped are not specified when counting_type is equal to 5 or 6.

seconds_flag specifies whether seconds_value is present when full_timestamp_flag is equal to 0.

seconds_value specifies the value of *SS* used to compute the EquivalentTimestamp. seconds_value shall not exceed 59. If not present, the previous seconds_value in decoding order shall be used as *SS* to compute the EquivalentTimestamp.

minutes_flag specifies whether seconds_value is present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1.

minutes_value specifies the value of *MM* used to compute the EquivalentTimestamp. minutes_value shall not exceed 59. If not present, the previous minutes_value in decoding order shall be used as *MM* to compute the EquivalentTimestamp.

hours_flag specifies whether seconds_value is present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1 and minutes_flag is equal to 1.

hours_value specifies the value of *HH* used to compute the EquivalentTimestamp. hours_value shall not exceed 23. If not present, the previous hours_value in decoding order shall be used as *HH* to compute the EquivalentTimestamp.

time_offset specifies the value of *TO* used to compute the EquivalentTimestamp. The number of bits used to represent time_offset shall be equal to time_offset_length. If time_offset is not present, the value 0 shall be used as *TO* to compute the EquivalentTimestamp.

D.2.3 Pan-scan rectangle SEI message semantics

The pan-scan rectangle SEI message parameters define the coordinates of a rectangle relative to the cropping rectangle of the picture parameter set. Each coordinate of this rectangle is specified in units of one-sixteenth sample spacing relative to the luma sampling grid.

[Ed. Note (GJS): Needs persistence indicator and multiple pan-scan count. See disposition notes for JVT-F013.]

pan_scan_rect_id contains an identifying number that may be used to identify the purpose of the pan-scan rectangle (for example, to identify the rectangle as the area to be shown on a particular display device or as the area that contains a particular actor in the scene). pan_scan_rect_id shall not exceed 2^{32}-1.

Values of pan_scan_rect_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of pan_scan_rect_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of pan_scan_rect_id in the range of 256 to 511 or in the range of $2^{31}$to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

pan_scan_rect_left_offset, pan_scan_rect_right_offset, pan_scan_rect_top_offset, and **pan_scan_rect_bottom_offset** specify, as signed integer quantities in units of one-sixteenth sample spacing relative to the luma sampling grid, the location of the pan-scan rectangle. The values of each of these four syntax elements shall be inside the interval from -2^{31} to 2^{31}-1, inclusive.

The pan-scan rectangle is specified, in units of one-sixteenth sample spacing relative to the luma sampling grid, as the area of the rectangle with horizontal coordinates from 32 * frame_crop_left_offset + pan_scan_rect_left_offset to 32 * [8 * PicWidthInMbs – frame_crop_right_offset] + pan_scan_rect_right_offset – 1 and with vertical coordinates from 32 * frame_crop_top_offset + pan_scan_rect_top_offset to 32 * [8 * PicHeightInMbs – frame_crop_bottom_offset] +

pan_scan_rect_bottom_offset – 1, inclusive. If this rectangular area includes samples outside of the cropping rectangle, the region outside of the cropping rectangle may be filled with synthesized content (such as black video content or neutral grey video content) for display.

The value of 32 * frame_crop_left_offset + pan_scan_rect_left_offset shall be less than or equal to 32 * [8 * PicWidthInMbs – frame_crop_right_offset] + pan_scan_rect_right_offset – 1.

The value of 32 * frame_crop_top_offset + pan_scan_rect_top_offset shall be less than or equal to 32 * [8 * PicHeightInMbs – frame_crop_bottom_offset] + pan_scan_rect_bottom_offset – 1. [Ed. Note: The values of the left and right parameters need also to be limited to guarantee that the indicated rectangle is not empty. (GJS): I believe they already are – run through the numbers and see.] [Ed. Note (GJS): Probably need to specify such that if frame_mbs_only is false, the height of the cropping rectangle is a multiple of 4 luma samples instead of 2 – otherwise, association of chroma samples to fields gets tricky.]

D.2.4 Filler payload SEI message semantics

This message contains a series of payloadSize bytes of value 0xFF, which can be discarded.

ff_byte shall be a byte having the value 0xFF.

D.2.5 User data registered by ITU-T Recommendation T.35 SEI message semantics

This message contains user data registered as specified by ITU-T Recommendation T.35, the contents of which are not specified by this Recommendation | International Standard.

itu_t_t35_country_code shall be a byte having a value specified as a country code by ITU-T Recommendation T.35 Annex A.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as a country code by ITU-T Recommendation T.35 Annex B.

itu_t_t35_payload_byte shall be a byte containing data registered as specified by ITU-T Recommendation T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the itu_t_t35_payload_byte, in the format specified by the Administration that issued the terminal provider code. Any remaining itu_t_t35_payload_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

D.2.6 User data unregistered SEI message semantics

This message contains unregistered user data identified by a UUID, the contents of which are not specified by this Recommendation | International Standard.

uuid_iso_11578 shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A.

user_data_payload_byte shall be a byte containing data having syntax and semantics as specified by the UUID generator.

D.2.7 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable pictures for display after the decoder initiates random access or after the encoder indicates a broken link in the sequence. All decoded pictures at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content, if the decoding process is started at the first picture in decoding order after the recovery point SEI message. Decoded pictures produced by random access at or before the picture associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the picture associated with the recovery point SEI message may contain references to pictures not available in the decoded picture buffer.

In addition, by use of the broken_link_flag, the recovery point SEI message can indicate to the decoder the location of some pictures in the bitstream that can result in serious visual artefacts if displayed, even when the decoding process was begun at a previous IDR point in decoding order.

NOTE – The broken_link_flag can be used by encoders to indicate the location of a splice point after which the decoding process for the decoding of some pictures may cause references to pictures that, though available for use in the decoding process, are not the pictures that were used for reference when the bitstream was originally encoded.

The recovery point is specified as a count in units of coded pictures subsequent to the current picture at the position of the SEI message.

NOTE – If HRD information is present in the bitstream, a buffering period SEI message should be associated with the picture associated with the recovery point SEI message in order to establish initialisation of the HRD buffer model after a random access.

recovery_frame_cnt specifies the recovery point of output pictures in output order. All decoded pictures in output order are indicated to be correct or approximately correct in content starting at the output order position of the reference picture having the frame_num equal to the frame_num of the next slice incremented by recovery_frame_cnt in modulo MaxFrameNum arithmetic. recovery_frame_cnt shall not exceed MaxFrameNum-1.

exact_match_flag indicates whether decoded pictures at and subsequent to the specified recovery point in output order derived by starting the decoding process at the picture associated with the recovery point SEI message shall be an exact match to the pictures that would be produced by a decoder starting at the previous IDR point in the NAL unit stream. The value 0 indicates that the match need not be exact and the value 1 indicates that the match shall be exact.

If decoding starts from the location of the recovery point SEI message, all references to unavailable reference pictures shall be inferred as references to pictures containing only intra macroblocks and having sample values given by Y=Cb=Cr=128 (mid-level grey) for purposes of determining the conformance of the value of exact_match_flag.

NOTE – When performing random access, decoders should infer all references to unavailable reference pictures as references to pictures containing only intra macroblocks and having sample values given by Y=Cb=Cr=128 (mid-level grey), regardless of the value of exact_match_flag.

broken_link_flag indicates the presence or absence of a broken link in the NAL unit stream at the location of the recovery point SEI message. If broken_link_flag is equal to 1, pictures produced by starting the decoding process at the previous IDR point may contain undesirable visual artefacts due to splicing operations and the pictures at and subsequent to the picture associated with the recovery point SEI message in decoding order should not be displayed until the specified recovery point in output order. If broken_link_flag is equal to 0, no indication is given regarding any potential presence of visual artefacts.

Regardless of the value of the broken_link_flag, pictures subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

NOTE – If a sub-sequence information SEI message is transmitted in conjunction with a recovery point SEI message in which broken_link_flag is equal to 1 and if sub_seq_layer_num is 0, sub_seq_id should be different from the latest sub_seq_id for sub_seq_layer_num equal to 0 that was decoded prior to the location of the recovery point SEI message. If broken_link_flag is equal to 0, the sub_seq_id in sub-sequence layer 0 should remain unchanged.

changing_slice_group_idc shall be 0, if num_slice_groups_minus1 is 0 in any picture within the changing slice group period, i.e., the period between the picture associated with the recovery point SEI message (inclusive) and the specified recovery point (exclusive) in decoding order. changing_slice_group_idc equal to 0 indicates that if all macroblocks of the pictures within the changing slice group period are decoded, decoded pictures are correct or approximately correct in content at and subsequent to the recovery point in output order.

If changing_slice_group_idc equal to 1 or 2, num_slice_groups_minus1 shall be 1 and the macroblock-to-slice-group map type 3, 4, or 5 shall be applied in the changing slice group period.

changing_slice_group_idc equal to 1 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 0 are used in inter prediction of any macroblock within slice group 0. In addition, changing_slice_group_idc equal to 1 indicates that if all macroblocks in slice group 0 within the changing slice group period are decoded, decoded pictures are correct or approximately correct in content at and subsequent to the specified recovery point in output order regardless of whether any macroblock in slice group 1 within the changing slice group period are decoded.

changing_slice_group_idc equal to 2 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 1 are used in inter prediction of any macroblock within slice group 1. In addition, changing_slice_group_idc equal to 2 indicates that if all macroblocks in slice group 1 within the changing slice group period are decoded, decoded pictures are correct or approximately correct in content at and subsequent to the specified recovery point in output order regardless of whether any macroblock in slice group 0 within the changing slice group period are decoded.

changing_slice_group_idc shall be in the range of 0 to 2, inclusive.

D.2.8 Decoded reference picture marking repetition SEI message semantics

The decoded reference picture marking repetition SEI message is used to repeat the decoded reference picture marking syntax structure that was located in the slice header of an earlier picture in the sequence in decoding order.

original_idr_flag shall be equal to 1 if the decoded reference picture marking syntax structure occurred originally in an IDR picture. original_idr_flag shall be equal to 0 if the repeated decoded reference picture marking syntax structure did not occur in an IDR picture originally.

original_frame_num shall be equal to the frame_num of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

original_field_pic_flag shall be equal to the field_pic_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

original_bottom_field_flag shall be equal to the bottom_field_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

dec_ref_pic_marking( ) shall contain a copy of the decoded reference picture marking syntax structure of the picture whose frame_num was original_frame_num. The nal_unit_type used for specification of the repeated dec_ref_pic_marking( ) syntax structure shall be the nal_unit_type of the slice header(s) of the picture whose frame_num was original_frame_num (i.e., nal_unit_type as used in subclause 7.3.3.3 shall be considered equal to 5 if original_idr_flag is equal to 1 and shall not be considered equal to 5 if original_idr_flag is equal to 0).

D.2.9 Spare picture SEI message semantics

This SEI message indicates that certain slice group map units, called spare slice group map units, in one or more decoded reference pictures resemble the co-located slice group map units in a specified decoded picture called the target picture. A spare slice group map unit may be used to replace a co-located, incorrectly decoded slice group map unit, in the target picture. A decoded picture containing spare slice group map units is called a spare picture.

For all spare pictures identified in a spare picture SEI message, the value of frame_mbs_only_flag shall be equal to the value of frame_mbs_only_flag of the target picture in the same SEI message. If the target picture is a decoded field, then all spare pictures identified in the same SEI message shall be decoded fields. If the target picture is a decoded frame, then all spare pictures identified in the same SEI message shall be decoded frames. For all spare pictures identified in a spare picture SEI message, the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1 shall be equal to the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1, respectively, of the target picture in the same SEI message. The picture associated (as specified in subclause 7.4.1.2) with this message shall appear after the target picture, in decoding order.

target_frame_num indicates the frame_num of the target picture.

spare_field_flag equal to 0 indicates that the target picture and the spare pictures are decoded frames. spare_field_flag equal to 1 indicates that the target picture and the spare pictures are decoded fields.

target_bottom_field_flag equal to 0 indicates that the target picture is a top field. target_bottom_field_flag equal to 1 indicates that the target picture is a bottom field.

A target picture is a decoded reference picture whose corresponding primary coded picture precedes the current picture, in decoding order, and in which the values of frame_num, field_pic_flag (if present) and bottom field_flag (if present) are equal to target_frame_num, spare_field_flag and target_bottom_field_flag, respectively.

num_spare_pics_minus1 indicates the number of spare pictures for the specified target picture. The number of spare pictures equals num_spare_pics_minus1 + 1. num_spare_pics_minus1 shall be in the range of 0 to 15, inclusive.

delta_spare_frame_num[i **]** is used to identify the spare picture that contains the i-th set of spare slice group map units, hereafter called the i-th spare picture, as specified below. The delta_spare_frame_num[i] shall be in the range of 0 to MaxFrameNum - 1 - !spare_field_flag, inclusive.

The frame_num of the i-th spare picture, spareFrameNum[i], is derived as follows for all values of i from 0 to num_spare_pics_minus1, inclusive:

```
candidateSpareFrameNum = target_frame_num - !spare_field_flag
for ( i = 0; i <= num_spare_pics_minus1; i++ ) {
        if ( candidateSpareFrameNum < 0 )
                candidateSpareFrameNum = MaxFrameNum – 1
        spareFrameNum[ i ] = candidateSpareFrameNum – delta_spare_frame_num[ i ]
        if( spareFrameNum[ i ] < 0 )
                spareFrameNum[ i ] = MaxFrameNum + spareFrameNum[ i ]
        candidateSpareFrameNum = spareFrameNum[ i ] - !spare_field_flag
}                                                                    (D-3)
```

spare_bottom_field_flag[i **]** equal to 0 indicates that the i-th spare picture is a top field. spare_bottom_field_flag[i] equal to 1 indicates that the i-th spare picture is a bottom field.

The 0-th spare picture is a decoded reference picture whose corresponding primary coded picture precedes the target picture, in decoding order, and in which the values of frame_num, field_pic_flag (if present) and bottom field_flag (if present) are equal to spareFrameNum[0], spare_field_flag and spare_bottom_field_flag[0], respectively. The i-th spare picture is a decoded reference picture whose corresponding primary coded picture precedes the (i-1)th spare

picture, in decoding order, and in which the values of frame_num, field_pic_flag (if present) and bottom field_flag (if present) are equal to spareFrameNum[i], spare_field_flag and spare_bottom_field_flag[i], respectively.

spare_area_idc[i **]** indicates the method used to identify the spare slice group map units in the i-th spare picture. spare_area_idc[i] shall be in the range of 0 to 2, inclusive. spare_area_idc[i] equal to 0 indicates that all slice group map units in the i-th spare picture are spare units. spare_area_idc[i] equal to 1 indicates that the value of the syntax element **spare_unit_flag**[i][j] is used to identify the spare slice group map units. spare_unit_flag[i][j] equal to 0 indicates that the j-th slice group map unit in raster scan order in the i-th spare picture is a spare unit. spare_unit_flag[i][j] equal to 1 indicates that the that j-th slice group map unit in raster scan order in the i-th spare picture is not a spare unit. spare_area_idc[i] equal to 2 indicates that the **zero_run_length**[i][j] syntax element is used to derive the values of spareUnitFlagInBoxOutOrder[i][j], as described below [Ed. Note: might be better to assign an equation number? (YKW): Yes.]. In this case, the spare slice group map units identified in spareUnitFlagInBoxOutOrder[i][j] appear in counter-clockwise box-out order, as specified in subclause 8.2.4.4, for each spare picture. spareUnitFlagInBoxOutOrder[i][j] equal to 0 indicates that the j-th slice group map unit in counter-clockwise box-out order in the i-th spare picture is a spare unit. spareUnitFlagInBoxOutOrder[i][j] equal to 1 indicates that the that j-th slice group map unit unit in counter-clockwise box-out order in the i-th spare picture is not a spare unit.

If spare_area_idc[0] is equal to 2, spareUnitFlagInBoxOutOrder[0][j] is derived as follows:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
        for( k = 0; k < zero_run_length[ 0 ][ loop ]; k++ )
                spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 0
        spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 1
}
```

If spare_area_idc[i] is equal to 2 and the value of i is greater than 0, spareUnitFlagInBoxOutOrder[i][j] is derived as follows:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
        for( k = 0; k < zero_run_length[ i ][ loop ]; k++ )
                spareUnitFlagInBoxOutOrder[ i ][ j ] = spareUnitFlagInBoxOutOrder[ i - 1 ][ j++ ]
        spareUnitFlagInBoxOutOrder[ i ][ j ] = !spareUnitFlagInBoxOutOrder[ i - 1 ][ j++ ]
}
```

D.2.10 Scene information SEI message semantics

A scene and a scene transition are herein defined as a set of consecutive pictures in output order.

NOTE - Decoded pictures within one scene generally have similar content. The scene information SEI message is used to label pictures with scene identifiers and to indicate scene changes. The message specifies how the uncoded source pictures for the labeled pictures were created. The decoder may use the information to select an appropriate algorithm to conceal transmission errors. For example, a specific algorithm may be used to conceal transmission errors that occurred in pictures belonging to a gradual scene transition. Furthermore, the scene information SEI message may be used in a manner determined by the application, such as for indexing the scenes of a coded sequence.

A scene information SEI message labels all pictures, in decoding order, from the primary coded picture to which the SEI message is associated (inclusive), as specified in subclause [7.4.1.2], to the primary coded picture to which the next scene information SEI message, in decoding order, is associated (exclusive). These pictures are herein referred to as the target pictures.

scene_info_known_flag equal to 0 indicates that the scene or scene transition to which the target pictures belong is unspecified. scene_info_known_flag equal to 1 indicates that the target pictures belong to the same scene or scene transition.

scene_id identifies the scene to which the target pictures belong. The value of scene_id shall be the same as the value of the scene_id of the previous picture, in output order, that is marked with a value of scene_transition_type less than 4, if the value of scene_transition_type of the target pictures is greater than 3 and if the target pictures and the previous picture (in output order) that is marked with a value of scene_transition_type less than 4 originate from the same uncoded source scene.

scene_id shall be in the range of 0 to 2^{32}-1, inclusive. Values of scene_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of scene_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of scene_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

scene_transition_type specifies in which type of a scene transition, if any, the target pictures are involved. The valid values of **scene_transition_type** are specified in Table D-4.

Table D-4 – Scene transition types.

Value	Description
0	No transition
1	Fade to black
2	Fade from black
3	Unspecified transition from or to constant colour
4	Dissolve
5	Wipe
6	Unspecified mixture of two scenes

If scene_transition_type is greater than 3, the target pictures include contents both from the scene labeled by its scene_id and the next scene, in output order, which is labeled by second_scene_id (see below). The term "the current scene" is used to indicate the scene labeled by scene_id. The term "the next scene" is used to indicate the scene labeled by second_scene_id. It is not required for any following picture, in output order, to be labeled with scene_id equal to second_scene_id of the current SEI message.

Scene transition types are specified as follows.

"**No transition**" specifies that the target pictures are not involved in a gradual scene transition.

NOTE - If two consecutive pictures in output order have scene_transition_type equal to zero and different values of scene_id, a scene cut occurred between the two pictures.

"**Fade to black**" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade to black scene transition, i.e., the luma samples of the scene gradually approach zero and the chroma samples of the scene gradually approach 128. If two pictures are labeled to belong to the same scene transition and their scene_transition_type is "Fade to black", the later one, in output order, is darker than the previous one.

"**Fade from black**" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade from black scene transition, i.e., the luma samples of the scene gradually diverge from zero and the chroma samples of the scene may gradually diverge from 128. If two pictures are labeled to belong to the same scene transition and their scene_transition_type is "Fade from black", the later one in output order is lighter than the previous one.

"**Dissolve**" indicates that the sample values of each target picture (before encoding) were generated by calculating a sum of co-located weighted sample values of a picture from the current scene and a picture from the next scene. The weight of the current scene gradually decreases from full level to zero level, whereas the weight of the next scene gradually increases from zero level to full level. If two pictures are labeled to belong to the same scene transition and their scene_transition_type is "Dissolve", the weight of the current scene for the later one, in output order, is smaller than the weight of the current scene for the previous one, and the weight of the next scene for the later one, in output order, is larger than the weight of the current scene for the previous one.

"**Wipe**" indicates that some of the sample values of each target picture (before encoding) were generated by copying co-located sample values of a picture in the next scene. If two pictures are labeled to belong to the same scene transition and their scene_transition_type is "Wipe", the number of samples copied from the next scene for the later one in output order is larger than the previous one.

second_scene_id identifies another scene in the gradual scene transition in which the target pictures are involved. The value of second_scene_id shall be the same as the value of scene_id of the next picture, in output order, that is marked with a value of scene_transition_type less than 4, if the target pictures and the next picture (in output order) that is marked with a value of scene_transition_type less than 4 originate from the same uncoded source scene. The value of second_scene_id shall not be equal to the value of scene_id in the previous picture, in output order.

If the value of scene_id of a picture is equal to the value of scene_id of the following picture, in output order, and the value of scene_transition_type in both of these pictures is equal to 0, these two pictures belong to the same scene. If the value of scene_id of a picture is equal to the value of scene_id of the following picture, in output order, and the value of scene_transition_type in both of these pictures is less than 4, these two pictures originate from the same uncoded source scene. If the values of scene_id, scene_transition_type and second_scene_id of a picture are equal to the values of scene_id, scene_transition_type and second_scene_id (respectively) of the following picture, in output order, these two pictures belong to the same scene transition.

second_scene_id shall be in the range of 0 to 2^{32}-1, inclusive. Values of second_scene_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of second_scene_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of second_scene_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

D.2.11 Sub-sequence information SEI message semantics

The sub-sequence information SEI message is used to indicate the position of a picture in data dependency hierarchy that consists of sub-sequence layers and sub-sequences.

A sub-sequence layer contains a subset of the coded pictures in a sequence. Sub-sequence layers are numbered with non-negative integers. A layer having a larger layer number is a higher layer than a layer having a smaller layer number. The layers are ordered hierarchically based on their dependency on each other so that a layer does not depend on any higher layer and may depend on lower layers. In other words, layer 0 is independently decodable, pictures in layer 1 may be predicted from layer 0, pictures in layer 2 may be predicted from layers 0 and 1, etc.

NOTE: The subjective quality is expected to increase along with the number of decoded layers.

A sub-sequence is a set of coded pictures within a sub-sequence layer. A picture shall reside in one sub-sequence layer and in one sub-sequence only. A sub-sequence shall not depend on any other sub-sequence in the same or in a higher sub-sequence layer. A sub-sequence in layer 0 can be decoded independently of any picture that does not belong to the sub-sequence.

The sub-sequence information SEI message concerns the picture containing the next slice or slice data partition in decoding order. The picture to which the next slice or slice data partition belongs is herein referred to as the current picture.

The sub-sequence information SEI message shall not be present unless required_frame_num_update_behaviour_flag in the sequence parameter set referenced by the picture associated with the sub-sequence SEI message is equal to 1.

sub_seq_layer_num specifies the sub-sequence layer number of the current picture. If sub_seq_layer_num is greater than 0, memory management control operations shall not be used in any slice header of the current picture. If the current picture resides in a sub-sequence whose first picture in decoding order is an IDR picture, the value of sub_seq_id shall be the same as the value of idr_pic_id of the IDR picture. For a non-paired reference field, the value of sub_seq_layer_num shall be equal to 0. sub_seq_layer_num shall be less than 256.

sub_seq_id identifies the sub-sequence within a layer. If the current picture resides in a sub-sequence whose first picture in decoding order is an IDR picture, the value of sub_seq_id shall be the same as the value of idr_pic_id of the IDR picture. sub_seq_id shall be less than 65536.

first_ref_pic_flag equal to 1 specifies that the current picture is the first reference picture of the sub-sequence in decoding order. Otherwise, the first_ref_pic_flag shall be equal to 0.

leading_non_ref_pic_flag equal to 1 specifies that the current picture is a non-reference picture preceding any reference picture in decoding order within the sub-sequence or that the sub-sequence contains no reference pictures. Otherwise, the leading_non_ref_pic_flag shall be equal to 0.

last_pic_flag equal to 1 specifies that the current picture is the last picture of the sub-sequence (in decoding order), including all reference and non-reference pictures of the sub-sequence. For any other pictures, the last_pic_flag shall be 0.

The current picture is the first picture of a sub-sequence in decoding order, if no earlier picture in decoding order is labelled with the same sub_seq_id and sub_seq_layer_num as the current picture, or if the leading_non_ref_pic_flag is equal to 1 and the leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same sub_seq_id and sub_seq_layer_num as the current picture, or if the first_ref_pic_flag is equal to 1 and the leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same sub_seq_id and sub_seq_layer_num as the current picture, or if the last_pic_flag is equal to 1 in the previous picture in decoding order having the same sub_seq_id and sub_seq_layer_num as the current picture. Otherwise, the current picture belongs to the same sub-sequence as the previous picture in decoding order having the same sub_seq_id and sub_seq_layer_num as the current picture.

sub_seq_frame_num_flag equal to 0 specifies that sub_seq_frame_num is not present. sub_seq_frame_num_flag equal to 1 specifies that sub_seq_frame_num is present.

sub_seq_frame_num shall be equal to 0 for the first reference picture of the sub-sequence and for any non-reference picture preceding the first reference picture of the sub-sequence in decoding order. For each coded picture belonging to the sub-sequence in decoding order, sub_seq_frame_num shall be incremented by 1, in modulo MaxFrameNum operation, relative to the previous reference frame that belongs to the sub-sequence. Both fields of a frame, if present, shall have the same sub_seq_frame_num. sub_seq_frame_num shall be less than MaxFrameNum.

If the current picture is an IDR picture, it shall start a new sub-sequence in sub-sequence layer 0. Thus, the sub_seq_layer_num shall be 0, the sub_seq_id shall be different from the previous sub-sequence in sub-sequence layer 0, first_ref_pic_flag shall be 1, and leading_non_ref_pic_flag shall be equal to 0.

If the sub-sequence information SEI message is present for both coded fields of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, if present, shall be identical for both of these pictures. If the sub-sequence information SEI message is present only for one coded field of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, if present, are also applicable to the other coded field of the complementary field pair.

D.2.12 Sub-sequence layer characteristics SEI message semantics

The sub-sequence layer characteristics SEI message specifies the characteristics of sub-sequence layers.

num_sub_seq_layers_minus1 plus 1 specifies the number of sub-sequence layers in the sequence. num_sub_seq_layers_minus1 shall be less than or equal to 255.

A pair of average_bit_rate and average_frame_rate characterizes each sub-sequence layer. The first pair of average_bit_rate and average_frame_rate specifies the characteristics of sub-sequence layer 0. The second pair, if present, specifies the characteristics of sub-sequence layers 0 and 1 jointly. Each pair in decoding order specifies the characteristics for a range of sub-sequence layers from layer number 0 to the layer number specified by the layer loop counter. The values are in effect from the point they are decoded until an update of the values is decoded.

accurate_statistics_flag indicates how reliable the values of average_bit_rate and average_frame_rate are. accurate_statistics_flag equal to 1 indicates that the average_bit_rate and the average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate somewhat from the correct values.

average_bit_rate gives the average bit rate in units of 1000 bits per second. All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. The average bit rate is derived according to the picture removal time specified in Annex C of the Recommendation | International Standard. In the following, B is the number of bits in all NAL units succeeding a sub-sequence layer characteristics SEI message (including the bits of the NAL units of the current picture) and preceding the next sub-sequence layer characteristics SEI message or the end of the stream. t_1 is the removal time (in seconds) of the current picture, and t_2 is the removal time (in seconds) of the latest picture before the next sub-sequence layer characteristics SEI message or the end of the stream. Then, the average_bit_rate is derived as follows provided that $t_1 \neq t_2$:

$$\text{average\_bit\_rate} = \text{Round}(B \div ((t_2 - t_1) * 1000)) \quad \text{(D-4)}$$

If t1 = t2, average_bit_rate shall be 0, which indicates an unspecified bit rate.

average_frame_rate gives the average frame rate in units of frames/(256 seconds). All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. In the following, C is the number of frames between the current picture (inclusive) and the next sub-sequence layer characteristics SEI message or the end of the stream. t_1 is the removal time (in seconds) of the current picture, and t_2 is the removal time (in seconds) of the latest picture before the next sub-sequence layer characteristics SEI message or the end of the stream. Then, the average_frame_rate is derived as follows provided that $t_1 \neq t_2$:

$$\text{average\_frame\_rate} = \text{Round}(C * 256 \div (t_2 - t_1)) \quad \text{(D-5)}$$

If $t_1 = t_2$, average_frame_rate shall be 0, which indicates an unspecified frame rate.

D.2.13 Sub-sequence characteristics SEI message semantics

The sub-sequence characteristics SEI message indicates the characteristics of a sub-sequence. It also indicates inter prediction dependencies between sub-sequences.

sub_seq_layer_num specifies the sub-sequence layer number to which the sub-sequence characteristics SEI message applies. sub_seq_layer_num shall be less than 256.

sub_seq_id specifies the sub-sequence within a layer to which the sub-sequence characteristics SEI message applies. sub_seq_id shall be less than 65536.

This message applies to the next sub-sequence in decoding order having the specified sub_seq_layer_num and sub_seq_id. This sub-sequence is herein called the target sub-sequence.

duration_flag equal to 0 indicates that the duration of the target sub-sequence is not specified.

sub_seq_duration specifies the duration of the target sub-sequence in clock ticks of a 90-kHz clock.

average_rate_flag equal to 0 indicates that the average bit rate and the average frame rate of the target sub-sequence are unspecified.

accurate_statistics_flag indicates how reliable the values of average_bit_rate and average_frame_rate are. accurate_statistics_flag equal to 1, indicates that the average_bit_rate and the average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate from the statistically correct values.

average_bit_rate gives the average bit rate in (1000 bits)/second of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average bit rate is derived according to the picture removal time specified in subclause C.1.2. In the following, B is the number of bits in all NAL units in the sub-sequence. t_1 is the removal time (in seconds) of the first picture of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last picture of the sub-sequence (in decoding order). Then, the average_bit_rate is derived as follows provided that $t_1 \neq t_2$:

$$\text{average\_bit\_rate} = \text{Round}(B \div ((t_2 - t_1) * 1000)) \quad \text{(D-6)}$$

If $t_1 = t_2$, average_bit_rate shall be 0.

average_frame_rate gives the average frame rate in units of frames/(256 seconds) of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average frame rate is derived according to the picture removal time specified in subclause C.1.2. In the following, C is the number of frames in the sub-sequence. t_1 is the removal time (in seconds) of the first picture of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last picture of the sub-sequence (in decoding order). Then, the average_frame_rate is derived as follows provided that $t_1 \neq t_2$:

$$\text{average\_frame\_rate} = \text{Round}(C * 256 \div (t_2 - t_1)) \quad \text{(D-7)}$$

If $t_1 = t_2$, average_frame_rate shall be 0.

num_referenced_subseqs gives the number of sub-sequences that contain pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. num_referenced_subseqs shall be less than 256.

ref_sub_seq_layer_num, ref_sub_seq_id, and **ref_sub_seq_direction** identify the sub-sequence that contains pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. If ref_sub_seq_direction is equal to 0, a set of candidate sub-sequences consists of the sub-sequences whose sub_seq_id is equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and whose first picture in decoding order precedes the first picture of the target sub-sequence in decoding order. If ref_sub_seq_direction is equal to 1, a set of candidate sub-sequences consists of the sub-sequences whose sub_seq_id is equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and whose first picture in decoding order succeeds the first picture of the target sub-sequence in decoding order. The sub-sequence used as a reference for the target sub-sequence is the sub-sequence among the set of candidate sub-sequences whose first picture is the closest to the first picture of the target sub-sequence in decoding order.

D.2.14 Full-frame freeze SEI message semantics

The full-frame freeze SEI message indicates that the contents of the entire prior displayed video frame in output order should be kept unchanged, without updating the display using the contents of the current decoded picture. The displayed frame should then remain unchanged until a full-frame freeze release SEI message is received, or until timeout occurs, whichever comes first. The full-frame freeze shall lapse due to timeout after five seconds or five pictures in output order, whichever is a longer period of time. The timeout can be prevented by the issuance of another full-frame freeze SEI message prior to or upon expiration of the timeout period.

D.2.15 Full-frame freeze release SEI message semantics

The full-frame freeze release SEI message indicates that the update of the displayed video frame should resume, starting with the contents of the current decoded picture and continuing for subsequent pictures in output order. The full-frame freeze release SEI message cancels the effect of any full-frame freeze SEI message sent with pictures that precede the current picture in output order.

D.2.16 Full-frame snapshot SEI message semantics

The full-frame snapshot SEI message indicates that the current frame is labelled for use as determined by the application as a still-image snapshot of the video content.

snapshot_id specifies a snapshot identification number. snapshot_id shall not exceed $2^{32}-1$.

Values of snapshot_id from 0 to 255 and from 512 to $2^{31}-1$ may be used as determined by the application. Values of snapshot_id from 256 to 511 and from 2^{31} to $2^{32}-1$ are reserved for future use by ITU-T | ISO/IEC. Decoders

encountering a value of snapshot_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

D.2.17 Progressive refinement segment start SEI message semantics

The progressive refinement segment start SEI message specifies the beginning of a set of consecutive coded pictures that is labelled as the current picture followed by a sequence of one or more pictures of refinement of the quality of the current picture, rather than as a representation of a continually moving scene.

The tagged set of consecutive coded pictures shall continue until one of following conditions is true. When a condition below becomes true, the next slice to be decoded does not belong to the tagged set of consecutive coded pictures.

1. The next slice to be decoded belongs to an IDR picture.
2. num_refinement_steps_minus1 is greater than 0 and the frame_num of the next slice to be decoded is (currFrameNum + num_refinement_steps_minus1 + 1) % MaxFrameNum, where currFrameNum is the value of frame_num of the next slice following this SEI message in decoding order.
3. num_refinement_steps_minus1 is 0 and a progressive refinement segment end SEI message with the same progressive_refinement_id as the one in this SEI message is decoded.
4. num_refinement_steps_minus1 is 0 and a timeout of five seconds has elapsed since the decoding of this SEI message. The timeout can be prevented by the issuance of an identical progressive refinement segment start SEI message prior to or upon expiration of the timeout period.

The decoding order of picture within the tagged set of consecutive pictures should be the same as their output order. **progressive_refinement_id** specifies an identification number for the progressive refinement operation. progressive_refinement_id shall not exceed 2^{32}-1.

Values of progressive_refinement_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of progressive_refinement_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

num_refinement_steps_minus1 specifies the number of reference frames in the tagged set of consecutive coded pictures. num_refinement_steps_minus1 equal to 0 specifies that the number of reference frames in the tagged set of consecutive coded pictures is unknown. Otherwise, the number of reference frames in the tagged set of consecutive coded pictures is equal to num_refinement_steps_minus1 + 1. num_refinement_steps_minus1 shall not exceed MaxFrameNum-1.

D.2.18 Progressive refinement segment end SEI message semantics

The progressive refinement segment end SEI message specifies the end of a set of consecutive coded pictures that has been labelled by use of a progressive refinement segment start SEI message as an initial picture followed by a sequence of one or more pictures of the refinement of the quality of the initial picture, and ending with the current picture.

progressive_refinement_id specifies an identification number for the progressive refinement operation. progressive_refinement_id shall not exceed 2^{32}-1.

The progressive refinement segment end SEI message specifies the end of any progressive refinement segment previously started using a progressive refinement segment start SEI message with the same value of progressive_refinement_id.

Values of progressive_refinement_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of progressive_refinement_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32}-1 shall ignore (remove from the bitstream and discard) it.

D.2.19 Motion-constrained slice group set SEI message semantics

This SEI message indicates that inter prediction over slice group boundaries is constrained as specified below. If present, it shall only appear where it is associated, as specified in subclause 7.4.1.2, with a primary coded IDR picture.

The target picture set for this SEI message contains all consecutive pictures in decoding order starting with the associated primary coded IDR picture (inclusive) and ending with the following primary coded IDR picture (exclusive) or with the very last picture in the bitstream in decoding order if there is no following IDR picture. The slice group set is a collection of one or more slice groups, identified by the slice_group_id[i] syntax element.

This SEI message indicates that, for each picture in the target picture set, the inter prediction process is constrained as follows: No sample value outside the slice group set, and no sample value at a fractional sample position that is derived using one or more sample values outside the slice group set is used to inter predict any sample within the slice group set.

num_slice_groups_in_set_minus1 + 1 specifies the number of slice groups in the slice group set. The allowed range of num_slice_groups_in_set_minus1 is 0 to num_slice_groups_minus1, inclusive. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_id[i **]** identifies the slice group(s) contained within the slice group set. The allowed range is from 0 to num_slice_groups_in_set_minus1, inclusive. The size of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits.

[Ed. Note: Should "exact_match_flag" name be defined in 2 different places in the syntax? (see recovery point SEI message)]

exact_match_flag equal to 0 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set need not be exactly the same as the value of the same sample when all the macroblocks are decoded. exact_match_flag equal to 1 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set shall be exactly the same as the value of the same sample when all the macroblocks in the target picture set are decoded.

Note - if disable_deblocking_filter_idc equals to 2 in all slices in the target picture set, exact_match_flag should be 1.

pan_scan_rect_flag equal to 0 specifies that pan_scan_rect_id is not present. pan_scan_rect_flag equal to 1 specifies that pan_scan_rect_id is present.

pan_scan_rect_id indicates that the specified slice group set covers at least the pan-scan rectangle identified by pan_scan_rect_id within the target picture set.

Note - Multiple motion_constrained_slice_group_set SEI messages may be associated with the same IDR picture. Consequently, more than one slice group set may be active within a target picture set.

Note - The size, shape, and location of the slice groups in the slice group set may change within the target picture set.

D.2.20 Reserved SEI message semantics

This message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. Encoders conforming to this Recommendation | International Standard shall not send reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders conforming to this Recommendation | International Standard that encounter reserved SEI messages shall discard their content without effect on the decoding process, except as specified in future Recommendations | International Standards specified by ITU-T | ISO/IEC. reserved_sei_message_payload_byte is a byte reserved for future use by ITU-T | ISO/IEC.

Annex E
Video usability information

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

This Annex specifies syntax and semantics of those parts of the sequence parameter set that are not required for determining the decoded values of samples. The parameters specified in this annex can be used to facilitate the use of the decoded pictures or facilitate the resource allocation of a decoder by restricting certain video parameters beyond those limits specified by Annex A. Decoders are not required to process VUI parameters for conformance to this Recommendation | International Standard. [Ed. Note (->AG): Some parameters are needed to interpret the timing information in the SEI messages. E.g. num_units_in_tick, time_scale, cpb_removal_delay_length_minus1, or dpb_output_delay_length_minus1.]

For some of the parameters of this Annex, default values are specified in the semantics subclause. The syntax includes flags that allow avoiding the signalling of groups of parameters. If a specific group of parameters is not coded, the default values for these parameters become effective.

E.1 VUI syntax

E.1.1 VUI parameters syntax

vui_parameters( ) {	C	Descriptor
aspect_ratio_info_present_flag	0	u(1)
if(aspect_ratio_info_present_flag) {		
aspect_ratio_idc	0	u(8)
if(aspect_ratio_idc == Extended_SAR) {		
sar_width	0	u(16)
sar_height	0	u(16)
}		
}		
overscan_info_present_flag	0	u(1)
if(overscan_info_present_flag)		
overscan_appropriate_flag	0	u(1)
video_signal_type_present_flag	0	u(1)
if(video_signal_type_present_flag) {		
video_format	0	u(3)
video_full_range_flag	0	u(1)
colour_description_present_flag	0	u(1)
if(colour_description_present_flag) {		
colour_primaries	0	u(8)
transfer_characteristics	0	u(8)
matrix_coefficients	0	u(8)
}		
}		
chroma_loc_info_present_flag	0	u(1)
if (chroma_loc_info_present_flag) {		
chroma_sample_loc_type_top_field	0	ue(v)
chroma_sample_loc_type_bottom_field	0	ue(v)
}		
timing_info_present_flag	0	u(1)
if(timing_info_present_flag) {		
num_units_in_tick	0	u(32)
time_scale	0	u(32)
fixed_frame_rate_flag	0	u(1)
}		
nal_hrd_parameters_present_flag	0	u(1)
if(nal_hrd_parameters_present_flag == 1)		
hrd_parameters( )		
vcl_hrd_parameters_present_flag	0	u(1)
if(vcl_hrd_parameters_present_flag == 1)		
hrd_parameters( )		
if(nal_hrd_parameters_present_flag == 1 || vcl_hrd_parameters_present_flag == 1)		
low_delay_hrd_flag	0	u(1)
bitstream_restriction_flag	0	u(1)
if(bitstream_restriction_flag) {		

motion_vectors_over_pic_boundaries_flag	0	u(1)
max_bytes_per_pic_denom	0	ue(v)
max_bits_per_mb_denom	0	ue(v)
log2_max_mv_length_horizontal	0	ue(v)
log2_max_mv_length_vertical	0	ue(v)
num_reorder_frames	0	ue(v)
max_dec_frame_buffering	0	ue(v)
}		
}		

E.1.2 HRD parameters syntax

hrd_parameters( ) { /* coded picture buffer parameters */	C	Descriptor
cpb_cnt_minus1	0	ue(v)
bit_rate_scale	0	u(4)
cpb_size_scale	0	u(4)
for(k=0; k<=cpb_cnt_minus1; k++) {		
bit_rate_value[k **]**	0	ue(v)
cpb_size_value[k **]**	0	ue(v)
vbr_cbr_flag[k **]**	0	u(1)
}		
initial_cpb_removal_delay_length_minus1	0	u(5)
cpb_removal_delay_length_minus1	0	u(5)
dpb_output_delay_length_minus1	0	u(5)
time_offset_length	0	u(5)
}		

E.2 VUI semantics

E.2.1 VUI parameters semantics

aspect_ratio_info_present_flag: A flag that, when equal to 1, signals the presence of the aspect_ratio_idc. If the flag is 0, then the default value 0 shall apply to aspect_ratio_idc.

aspect_ratio_idc specifies the value of the sample aspect ratio of the luma samples. Table E-1 shows the meaning of the code. If aspect_ratio_idc indicates Extended_SAR, the sample aspect ratio is represented by sar_width and sar_height.

Table E-1 – Meaning of sample aspect ratio indicator

aspect_ratio_idc	Sample aspect ratio	(informative) Examples of use
0	Unspecified	
1	1:1 ("square")	1280x720 16:9 frame without overscan 1920x1080 16:9 frame without overscan (cropped from 1920x1088) 640x480 4:3 frame without overscan
2	12:11	720x576 4:3 frame with horizontal overscan 352x288 4:3 frame without overscan
3	10:11	720x480 4:3 frame with horizontal overscan 352x240 4:3 frame without overscan
4	16:11	720x576 16:9 frame with horizontal overscan 540x576 4:3 frame with horizontal overscan
5	40:33	720x480 16:9 frame with horizontal overscan 540x480 4:3 frame with horizontal overscan
6	24:11	352x576 4:3 frame without overscan 540x576 16:9 frame with horizontal overscan
7	20:11	352x480 4:3 frame without overscan 480x480 16:9 frame with horizontal overscan
8	32:11	352x576 16:9 frame without overscan
9	80:33	352x480 16:9 frame without overscan
10	18:11	480x576 4:3 frame with horizontal overscan
11	15:11	480x480 4:3 frame with horizontal overscan
12	64:33	540x576 16:9 frame with horizontal overscan
13	160:99	540x480 16:9 frame with horizontal overscan
14..254	Reserved	
255	Extended_SAR	

sar_width indicates the horizontal size of the sample aspect ratio (in arbitrary units).

sar_height indicates the vertical size of the sample aspect ratio (in the same arbitrary units as sar_width).

The sar_width and sar_height shall be relatively prime or zero. If aspect_ratio_idc is zero or if either of sar_width or sar_height are zero, the sample aspect ratio shall be considered unspecified by this Recommendation | International Standard.

overscan_info_present_flag: A flag that, when equal to 1, signals the presence of overscan_appropriate_flag. If overscan_info_present_flag is equal to 0 or is not present, the preferred display method for the video signal is unspecified.

overscan_appropriate_flag: A flag that, when equal to 1, indicates that the video signal is suitable for display using overscan. When overscan_appropriate_flag is equal to 0, it indicates that the video signal contains visually important information in the entire region out to the edges of the cropping rectangle of the picture, such that the video signal should not be displayed using overscan; instead, it should be displayed using either an exact match between the display area and the cropping rectangle, or using underscan. For example, overscan_appropriate_flag equal to 1 might be used for entertainment television programming, or for a live view of people in a videoconference, and overscan_appropriate_flag equal to 0 might be used for computer screen capture or security camera content.

video_signal_type_present_flag: A flag that, when 1, signals the presence of video signal information. If video_signal_type_present_flag is 0, then the following default values shall apply: video_format = 5, video_full_range_flag = 0, colour_description_present_flag = 0.

video_format: Indicates the representation of the pictures, as specified in Table E-2, before being coded in accordance with this Recommendation | International Standard.

Table E-2 – Meaning of video_format

video_format	Meaning
0	Component
1	PAL
2	NTSC
3	SECAM
4	MAC
5	Unspecified video format
6	Reserved
7	Reserved

video_full_range_flag indicates the nominal black level and range of the luma and chroma signals as derived from E'_Y, E'_{PB}, and E'_{PR} analogue component signals as follows:

If video_full_range_flag is equal to 0:

$$Y = Round(219 * E'_Y + 16) \quad \text{(E-1)}$$

$$Cb = Round(224 * E'_{PB} + 128) \quad \text{(E-2)}$$

$$Cr = Round(224 * E'_{PR} + 128) \quad \text{(E-3)}$$

If video_full_range_flag is equal to 1:

$$Y = Round(255 * E'_Y) \quad \text{(E-4)}$$

$$Cb = Round(255 * E'_{PB} + 128) \quad \text{(E-5)}$$

$$Cr = Round(255 * E'_{PR} + 128) \quad \text{(E-6)}$$

colour_description_present_flag indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

colour_primaries describes the chromaticity coordinates of the source primaries, and is specified in Table E-3 in terms of the CIE 1931 definition of x and y as specified by ISO/CIE 10527.

Table E-3 – Colour primaries

Value	Primaries		
0	Reserved		
1	ITU-R Recommendation BT.709		
	primary	x	y
	green	0.300	0.600
	blue	0.150	0.060
	red	0.640	0.330
	white D65	0.3127	0.3290
2	Unspecified Image characteristics are unknown or as determined by the application.		
3	Reserved		
4	ITU-R Recommendation BT.470-2 System M		
	primary	x	y
	green	0.21	0.71
	blue	0.14	0.08
	red	0.67	0.33
	white C	0.310	0.316
5	ITU-R Recommendation BT.470-2 System B, G		
	primary	x	y
	green	0.29	0.60
	blue	0.15	0.06
	red	0.64	0.33
	white D65	0.3127	0.3290
6	Society of Motion Picture and Television Engineers 170M		
	primary	x	y
	green	0.310	0.595
	blue	0.155	0.070
	red	0.630	0.340
	white D65	0.3127	0.3290
7	Society of Motion Picture and Television Engineers 240M (1987)		
	primary	x	y
	green	0.310	0.595
	blue	0.155	0.070
	red	0.630	0.340
	white D65	0.3127	0.3290
8	Generic film (colour filters using Illuminant C)		
	primary	x	y
	green	0.243	0.692 (Wratten 58)
	blue	0.145	0.049 (Wratten 47)
	red	0.681	0.319 (Wratten 25)
	white C	0.310	0.316
9-255	Reserved		

If video_signal_type_present_flag is 0 or colour_description_present_flag is 0, colour_primaries shall have an inferred value equal to 2 (the chromaticity is unspecified or is determined by the application).

transfer_characteristics: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is specified in Table E-4 as a function of a linear optical intensity input L_c with an analogue range from 0 to 1.

Table E-4 – Transfer characteristics

Value	Transfer Characteristic
0	Reserved
1	ITU-R Recommendation BT.709 $V = 1.099\ L_c^{0.45} - 0.099$ for $1 >= L_c >= 0.018$ $V = 4.500\ L_c$ for $0.018 > L_c$
2	Unspecified Image characteristics are unknown or are determined by the application.
3	Reserved
4	ITU-R Recommendation BT.470-2 System M Assumed display gamma 2.2
5	ITU-R Recommendation BT.470-2 System B, G Assumed display gamma 2.8
6	Society of Motion Picture and Television Engineers 170M $V = 1.099\ L_c^{0.45} - 0.099$ for $1 >= L_c >= 0.018$ $V = 4.500\ L_c$ for $0.018 > L_c$
7	Society of Motion Picture and Television Engineers 240M (1987) $V = 1.1115\ L_c^{0.45} - 0.1115$ for $L_c >= 0.0228$ $V = 4.0\ L_c$ for $0.0228 > L_c$
8	Linear transfer characteristics $V = L_c$
9	Logarithmic transfer characteristic (100:1 range) $V = 1.0 - \text{Log10}(L_c) \div 2$ for $1 >= L_c >= 0.01$ $V = 0.0$ for $0.01 > L_c$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0 - \text{Log10}(L_c) \div 2.5$ for $1 >= L_c >= 0.0031622777$ $V = 0.0$ for $0.0031622777 > L_c$
11..255	Reserved

If video_signal_type_present_flag is zero or colour_description_present_flag is 0, transfer_characteristics shall have an inferred value equal to 2 (the transfer characteristics are unspecified or determined by the application).

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luma and chroma signals from the green, blue, and red primaries, as specified in Table E-5.

Using the following definitions:

E'_R, E'_G, and E'_B are analogue with values between 0 and 1.

White is specified as having E'_R equal to 1, E'_G equal to 1, and E'_B equal to 1.

Then:

$$E'_Y = K_R * E'_R + (1 - K_R - K_B) * E'_G + K_B * E'_B \quad \text{(E-7)}$$

$$E'_{PB} = 0.5 * (E'_B - E'_Y) \div (1 - K_B) \quad \text{(E-8)}$$

$$E'_{PR} = 0.5 * (E'_R - E'_Y) \div (1 - K_R) \quad \text{(E-9)}$$

NOTE – Then E'_Y is analogue with values between 0 and 1, E'_{PB} and E'_{PR} are analogue with values between -0.5 and 0.5, and white is equivalently given by $E'_Y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$.

Table E-5 – Matrix coefficients

Value	Matrix
0	Reserved
1	ITU-R Recommendation BT.709 $K_R = 0.2126$; $K_B = 0.0722$
2	Unspecified Image characteristics are unknown or are determined by the application.
3	Reserved
4	Federal Communications Commission $K_R = 0.30$; $K_B = 0.11$
5	ITU-R Recommendation BT.470-2 System B, G: $K_R = 0.299$; $K_B = 0.114$
6	Society of Motion Picture and Television Engineers 170M $K_R = 0.299$; $K_B = 0.114$
7	Society of Motion Picture and Television Engineers 240M (1987) $K_R = 0.212$; $K_B = 0.087$
8-255	Reserved

If video_signal_type_present_flag is zero or colour_description_present_flag is 0, matrix_coefficients shall have an inferred value equal to 2 (the matrix coefficients shall be inferred to be unspecified or as determined by the application).

chroma_loc_info_present_flag: A flag that, when 1, signals the presence of the chroma location information. If the flag is 0, then the following default values shall apply: chroma_sample_loc_type_top_field = 0, chroma_sample_loc_type_bottom_field = 0

chroma_sample_loc_type_top_field and **chroma_sample_loc_type_bottom_field** specify the location of chroma samples for the top field and the bottom field as shown in Figure E-1. The value of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall not exceed 5.

NOTE – In progressive sequences, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should have the same value.

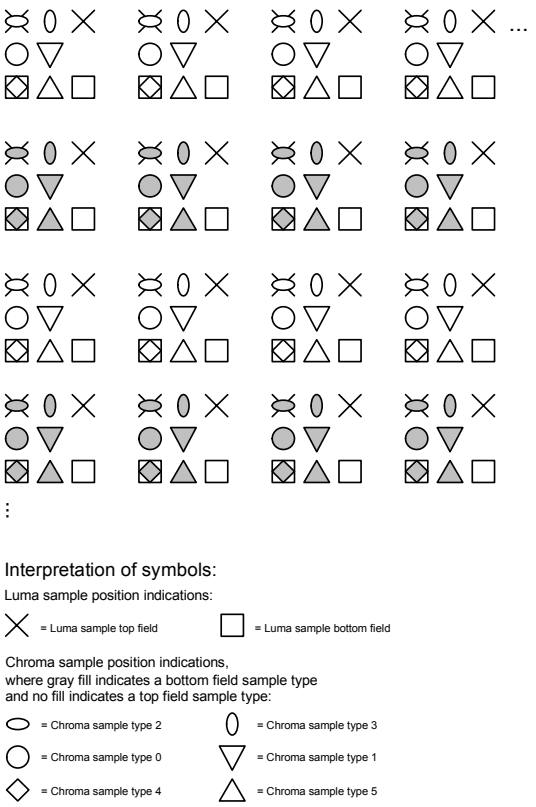


Figure E-1 – Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field

timing_info_present_flag: A flag that, when equal to 1, signals the presence of time unit information. If timing_info_present_flag is equal to 0, then the following default values shall be inferred: num_units_in_tick = 0, time_scale = 0, fixed_frame_rate_flag = 0.

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment of a clock tick counter. A clock tick is the minimum interval of time that can be represented in the coded data. For example, if the clock frequency of a video signal is 30000 ÷ 1001 Hz, time_scale may be 30 000 and num_units_in_tick may be 1001. If num_units_in_tick is 0, the duration of the clock tick is unspecified.

time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time_scale of 27 000 000. If time_scale is equal to 0, the duration of the clock tick specified above is unspecified.

fixed_frame_rate_flag is a bit that, if equal to 1, indicates that the temporal distance between the HRD output times of any two consecutive frames or fields in output order is constrained as follows.

If fixed_frame_rate_flag is equal to 1, for all n > 0 where n indicates the n-th picture in output order, the value of $\Delta t_{fi,dpb}(n)$ is computed as follows using $\Delta t_{o,dpb}(n)$ as specified in Equation C-10,

$$\Delta t_{fi,dpb}(n) = \Delta t_{o,dpb}(n) \div \text{DeltaTfiDivisor} \qquad \text{(E-10)}$$

where DeltaTfiDivisor is specified by Table E-6 based on the value of pic_struct_present_flag, field_pic_flag, and pic_struct for picture n - 1. Entries marked "-" in Table E-6 indicate a lack of dependence of DeltaTfiDivisor on the corresponding syntax element.

If fixed_frame_rate_flag is equal to 1, then the value computed for $\Delta t_{fi,dpb}(n)$ shall be the same for all n > 0. Moreover, if fixed_frame_rate_flag is equal to 1 and num_units_in_tick is not equal to 0 and time_scale is not equal to 0, then the value computed for $\Delta t_{fi,dpb}(n)$ shall be equal to num_units_in_tick ÷ time_scale for all n > 0.

Table E-6 – Divisor for computation of $\Delta t_{fi,dpb}(n)$

pic_struct_present_flag	field_pic_flag	pic_struct	DeltaTfiDivisor
0	1	-	1
1	-	1	1
1	-	2	1
0	0	-	2
1	-	0	2
1	-	3	2
1	-	4	2
1	-	5	3
1	-	6	3
1	-	7	4
1	-	8	6

nal_hrd_parameters_present_flag: If nal_hrd_parameters_present_flag is equal to 0, no HRD parameters pertaining to Type II bitstream conformance as specified in Annex C are present in the bitstream. In this case, the Type II bitstream conforms to an HRD whose buffer size and bit rate parameters are 20% higher than those given by the maxima specified in Table A-1 for the level specified by level_idc. [Ed. Note(GJS): Not true. There is not enough information in the bitstream if this data is not present.] If num_units_in_tick is equal to 0, nal_hrd_parameters_present_flag shall be equal to 0. If time_scale is equal to 0, nal_hrd_parameters_present_flag shall be equal to 0.

NOTE – When nal_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters, including the NAL sequence HRD parameter information and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

If nal_hrd_parameters_present_flag is equal to 1, then Type II HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag. [Ed. Note (AG/GJS): Remove remainder of paragraph if redundant.] At least one set of the Type II HRD parameters signalled in the sequence HRD parameters shall have CPB bit rate and buffer size values that are both less than values 20% higher than the maxima specified in Table A-1 for the level specified by level_idc. Some of the NAL HRD coded picture buffers signalled may have values outside the level limits.

Furthermore, if nal_hrd_parameters_present_flag is equal to 1, then a buffering period SEI message (subclauses D.1.1 and D.2.1) shall be present that is associated with the first picture of the sequence, and a picture timing SEI message (subclauses D.1.1 and D.2.1) shall precede each coded picture.

vcl_hrd_parameters_present_flag: If vcl_hrd_parameters_present_flag is equal to 0, no HRD parameters pertaining to VCL conformance as specified in Annex C are present in the bitstream. In this case, the VCL bitstream conforms to an HRD whose buffer size and bit rate parameters are given by the maxima specified in Table A-1 for the level specified by level_idc. If num_units_in_tick is equal to 0, vcl_hrd_parameters_present_flag shall be equal to 0. If time_scale is equal to 0, vcl_hrd_parameters_present_flag shall be equal to 0.

NOTE – When vcl_hrd_parameters_present_flag is equal to 0, the conformance of the VCL bitstream cannot be verified without provision of the VCL HRD parameters, including the VCL sequence HRD parameter information and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

If vcl_hrd_parameters_present_flag is equal to 1, then VCL HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag. [Ed. Note (AG/GJS): Remove remainder of paragraph if redundant.] At least one set of the HRD parameters signalled in the sequence HRD parameters shall have CPB bit rate and buffer size values that are both less than the maxima specified in Table A-1 for the level specified by level_idc. Some of the HRD coded picture buffers signalled may have values outside the level limits.

Furthermore, if vcl_hrd_parameters_present_flag is equal to 1, then a buffering period SEI message (subclauses D.1.1 and D.2.1) shall be associated with the first picture of the sequence, and a picture timing SEI message (subclauses D.1.2 and D.2.2) shall be associated with each subsequent coded picture.

low_delay_hrd_flag: If low_delay_hrd_flag is equal to 0, the HRD operates in delay-tolerant mode as specified in Annex C. If low_delay_hrd_flag is equal to 1, the HRD operates in low-delay mode. In low-delay mode, big pictures that violate the HRD removal time rules at the CPB are permitted.

NOTE - It is expected, but not required, that such big pictures occur only occasionally. [Ed. Note (AG/GJS): Move this note to Annex C]

[Ed. Note (GJS): Can low_delay_hrd_flag and fixed_frame_rate_flag both be equal to 1? What would that mean?]

bitstream_restriction_flag: A flag that, when 1, signals the presence of bitstream restriction indication information. If bitstream_restriction_flag is set to 0, then the following default values shall apply: motion_vectors_over_pic_boundaries_flag = 1, max_bytes_per_pic_denom = 2, max_bits_per_mb_denom = 1, log2_max_mv_length_horizontal = 16, log2_max_mv_length_vertical = 16, max_dec_frame_buffering = 16, num_reorder_frames = 16.

motion_vectors_over_pic_boundaries_flag equal to 0 indicates that no motion vector refers to samples outside the picture boundaries of the reference picture for all pictures of the sequence. motion_vectors_over_pic_boundaries_flag equal to 1 indicates that motion vectors in some pictures in the sequence may refer to samples outside the picture boundaries of a reference picture.

max_bytes_per_pic_denom indicates a number of bytes not exceeded by the sum of the sizes of the slice and slice data partition NAL units associated with any coded picture in the sequence.

The number of bytes that represent a picture in the NAL unit stream is specified for this purpose as the total number of bytes of NAL unit data (i.e., the total of the NumBytesInNALunit variables for the picture) that contain slices and slice data partitions for the picture. The value of max_bytes_per_pic_denom shall not exceed 16.

If max_bytes_per_pic_denom is equal to 0, no limits are specified.

If max_bytes_per_pic_denom is not equal to 0, no coded picture shall be represented in the sequence by more than

(PicSizeInMbs * 256 * ChromaFormatFactor) ÷ max_bytes_per_pic_denom (E-11)

bytes.

max_bits_per_mb_denom indicates the maximum number of coded bits that represent a macroblock in any picture of the sequence. The value of max_bits_per_mb_denom shall not exceed 16.

If max_bits_per_mb_denom is equal to 0, no limit is specified. If max_bits_per_mb_denom is not equal to 0, no coded macroblock shall be represented in the bitstream by more than

(2048 * ChromaFormatFactor + 128) ÷ max_bits_per_mb_denom (E-12)

bits.

[Ed. Note: Insert reference to method of counting bits for a particular MB, especially for CABAC.]

log2_max_mv_length_horizontal and **log2_max_mv_length_vertical** indicate the maximum absolute value of a decoded horizontal and vertical motion vector component, respectively, in ¼ luma sample units. A value of n asserts that no absolute value of a motion vector component is larger than 2^n units of ¼ luma sample displacement. The value of log2_max_mv_length_horizontal shall not exceed 16. The value of log2_max_mv_length_vertical shall not exceed 16.

NOTE - The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A.

num_reorder_frames indicates the maximum amount of frames, complementary field pairs, or non-paired fields that precede any frame, complementary field pair, or non-paired fields in the sequence in decoding order and follow it in output order. The value of num_reorder_frames shall not exceed the value of max_dec_frame_buffering.

max_dec_frame_buffering specifies minimum size of the decoded picture buffer in units of frames. The sequence shall not require a decoded picture buffer with capacity of more than max_dec_frame_buffering frames to enable the output of decoded pictures at the output times of the HRD. The value of max_dec_frame_buffering shall not exceed 16. [Ed. Note: (AG): What happens at the "seam" of two consecutive sequences with different values when no_output_of_prior_pics_flag is equal to 0?]

E.2.2 HRD parameters semantics

If multiple sequence parameter sets pertain to the bitstream, they shall contain consistent HRD information. [Ed. Note: What is the definition of consistent?]

cpb_cnt_minus1: This syntax element plus 1 specifies the number of CPB specifications in the bitstream. The value of cpb_cnt_minus1 shall not exceed 31. cpb_cnt_minus1 shall be equal to zero if low_delay_hrd_flag is equal to 1.

bit_rate_scale: Together with bit_rate_value[k], this syntax element specifies the maximum input bit rate of the k-th CPB in an HRD.

cpb_size_scale is used together with cpb_size_value[k] to define the CPB size of the k-th CPB in an HRD.

bit_rate_value[k **]**: Together with bit_rate_scale, this syntax element specifies the maximum input bit rate for the k-th CPB. bit_rate_value[k] shall not exceed 2^{32}-1. For any k>0, bit_rate_value[k] shall be greater than bit_rate_value[k - 1]. The actual bit rate in bits per second is given by [Ed. change to minus1 values and the upper value to 2^{32}-2]

$$bit\_rate[k] = bit\_rate\_value[k] * 2^{(6 + bit\_rate\_scale)} \quad \text{(E-13)}$$

cpb_size_value[k **]** is used together with cpb_size_scale to define the k-th CPB size. cpb_size_value[k] shall not exceed 2^{32}-1. [Ed. change to minus1 values and the upper value to 2^{32}-2]

The CPB size in bits is given by

$$cpb\_size[k] = cpb\_size\_value[k] * 2^{(4 + cpb\_size\_scale)} \quad \text{(E-14)}$$

For VCL HRD parameters, there shall be at least one value of k for which bit_rate[k] is within the maximum video bit rate, and cpb_size[k] is within the maximum CPB size specified in Table A-1. For NAL HRD parameters, there shall be at least one value of k for which bit_rate[k] is less than or equal to 120% of the maximum video bit rate, and cpb_size[k] is less than or equal to 120% of the maximum CPB size.

vbr_cbr_flag[k **]** equal to 0 specifies that to decode this bitstream by the HRD using the k-th CPB specification, the HRD operates in variable bit rate (VBR) mode. vbr_cbr_flag[k] equal to 1 specifies constant bit rate (CBR) operation. [Ed. Note (JVT): Bad syntax element name.]

initial_cpb_removal_delay_length_minus1 specifies the length in bits of the initial_cpb_removal_delay syntax element. The length of initial_cpb_removal_delay is initial_cpb_removal_delay_length_minus1+1.

cpb_removal_delay_length_minus1 specifies the length in bits of the cpb_removal_delay syntax element. The length of cpb_removal_delay is cpb_removal_delay_length_minus1+1.

dpb_output_delay_length_minus1 specifies the length in bits of the dpb_output_delay syntax element. The length of dpb_output_delay is dpb_output_delay_length_minus1+1.

time_offset_length specifies the length in bits of the time_offset syntax element.