Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG

5th Meeting: Geneva, Switzerland, 9-17 October, 2002

Document: **JVT-E022d7**
File: JVT-E022d7.doc
Generated: 2002-09-19

Title:	Editor's Proposed Draft Text Modifications for Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), Draft 7
Status:	Input Document to JVT
Contact:	Thomas Wiegand Heinrich Hertz Institute (HHI), Einsteinufer 37, D-10587 Berlin, Germany Tel: +49 - 30 - 31002 617, Fax: +49 - 030 - 392 72 00, wiegand@hhi.de
Purpose:	Report

[Ed. Notes:

Modifications in JVT-E022d1 relative to JVT-D157 J-FCD:

- Inclusion of editor's notes commenting on text (marked by "[Ed. Note: …]")

Modifications in JVT-E022d2 relative to JVT-E022d1:

- Automatic numbering inserted (switch field up-dates on/off using Ctrl Shift F11 / Ctrl F11)

Modifications in JVT-E022d3 relative to JVT-E022d2:

- CABAC related editorial changes suggested by AHG

Modifications in JVT-E022d4 relative to JVT-E022d3:

- editorial changes suggested on reflector
- changes received regarding CAVLC

Modifications in JVT-E022d5 relative to JVT-E022d4:

- editorial changes regarding automatic numbering and cross references for Figures, Tables, clauses, and subclauses.
- inserted Ed. Note on CABAC bug fix regarding MSBs and LSBs in Bytes.

Modifications in JVT-E022d6 relative to JVT-E022d5:

- editorial changes as suggested by Aharon Gill in clause 6.
- editorial changes added to the Introduction.

Modifications in JVT-E022d7 relative to JVT-E022d6:

- editorial changes as suggested by Ye-Kui Wang in clause 6.
- editorial change as suggested by Eric Viscito.
- editorial changes as suggested by Eric Viscito to luma 16x16 and chroma prediction process.
- editorial changes as suggested by Anthony Joch.
- general editorial changes.
- remark from Huifang Sun.
- editorial changes as suggested by Peter Borgwardt.
- editorial changes as suggested by Barry Haskell.
- editorial changes as suggested by Cristina Gomila.

End Ed. Notes]

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 Standardization 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 standardizing telecommunications on a worldwide basis. The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, 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 which 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 is being submitted for approval to the ITU-T and ISO/IEC JTC1/SC29. It 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 processing power and memory costs have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industy 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 ITU-T 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 video conferencing, 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 | Insternational 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 Application

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 they 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 "profile" and "level". These and other related terms are formally defined in clause 4.

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. 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 parameters in the bitstream. These constraints may be simple limits on numbers. Alternatively they may take the form of constraints on arithmetic combinations of the parameters (e.g. frame width multiplied by frame height multiplied by frame rate).

Coded video content conforming to this Specification 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 otherwise 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 while preserving image quality. The algorithm is not lossless as the exact sample values are not preserved through the encoding and decoding processes.

A number of techniques may be used to achieve high compression. The expected encoding algorithm (not specified in this Recommendation | International Standard) in inter coding first uses block-based motion compensation 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 error is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantised, producing an irreversible process that discards less important information while forming a close approximation to the source pictures. 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 Temporal processing

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 also shows only moderate compression. Inter coding (predictive or bipredictive) is more efficient using motion-compensated prediction of each block of sample values from some previously decoded picture selected by the encoder.

The application of the two 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

Each frame of interlaced video consists of two fields which 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 movement.

0.5.3 Macroblocks and motion segmentations

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

The selection of a motion compensation unit is a result of a trade-off between the coding gain provided by using motion information and the quantity of data needed to represent it. In this Recommendation | International Standard the motion compensation process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one quarter or one-eighth of a sample grid spacing displacement. The inter prediction process for motion-compensated 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.

In frame encoding, the prediction from the previous reference frame can itself be either frame-based or field-based. depending on the type of the motion vector information and other information that is encoded within the compressed picture representation. Motion vectors are encoded differentially with respect to predicted values formed from nearby encoded motion vectors.

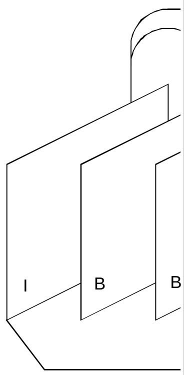
It is the responsibility of the encoder to calculate appropriate motion vectors and other data elements represented in the video data stream. This motion estimation process in the encoder and the selection of whether to use inter-picture prediction for the representation of each region of the video content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

0.5.4 Spatial redundancy reduction

Both source pictures and prediction errors have high spatial redundancy. This Recommendation | International Standard is based on the use of a block-based transform method for spatial redundancy removal. After motion-compensated

Deleted:
Deleted: Obtaining good image quality at the bit rates of interest demands very high compression, which is not achievable wit ... [1]
Deleted: reduce
Deleted: redundancy
Deleted: Motion compens ... [2]
Deleted: region
Deleted: three
Deleted: picture
Deleted: coded pictures (... [3]
Deleted: They
Deleted: are coded with
Deleted: -
Deleted: ed pictures (P-pictures
Deleted:)
Deleted: are coded more
Deleted: ly
Deleted: compensated
Deleted: Bi-predictive pi ... [4]
Deleted: organisation
Deleted: three
Deleted: picture
Deleted: Figure Intro-1 ... [5]

Bidire



Deleted: Fig ... [6]
Formatted: Bullets and Numbering ... [7]
Deleted: compensated
Deleted: or
Deleted:
Deleted: |
Deleted:
Deleted: standard
Deleted: motion

prediction or spatial-based prediction from previously-decoded samples within the current picture, the resulting prediction error is split into 4x4 blocks or up to 8x8 blocks if the adaptive block-size transform feature is used. 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 quantization 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). Then clause 6 (Source, coded, decoded, and output data formats) should be read for the relationship of the souce, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse the bitstream (see subclauses 7.1-7.3 for syntax) and the scope, restrictions, and conditions (see subclause 7.4 for semantics) that are imposed on the syntax elements. The actual entropy coding of each syntax element is specified in clause 9 (Entropy). 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.

Deleted: The

Deleted: should

Deleted: Source, coded, decoded and output data formats

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 which, 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 Standardization 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*

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 **B slice**: A bi-predictive *slice*; A *slice* that is coded in a manner in which a weighted average of two *inter prediction blocks* may be used for *inter prediction*.

3.3 **bitstream**: A sequence of bits that forms the representation of data and coded *fields* and *frames*.

3.4 **block**: An N-column by M-row array of samples, or NxM array of *transform coefficients*.

3.5 **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*.

3.6 **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.

3.7 **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*.

3.8 **byte stream format**: A *NAL unit stream* containing *start code prefixes* and *NAL units* as per Annex B.

3.9 **category**: For *slice layer* and lower layer *syntax elements*, specifies the allocation of syntax elements to data structures for *data partitioning*. It may also be used by the systems *layer* to refer to classes of syntax elements in a manner not specified in this Recommendation | International Standard.

3.10 **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 the chroma array or sample are Cr and Cb.

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

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

3.13 **coded pictures input buffer**: A first-in first-out (FIFO) buffer containing coded pictures in *decoding order* specified in the *video buffering verifier* in Annex C.

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

3.15 **common intermediate format (CIF)**: A video frame that is 22 macroblocks wide and 18 macroblocks high.

3.16 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame*. [Ed.Note: Used only in video attributes. suggest to avoid -> delete]

3.17 **context**: The numerical value of the *context variable* when decoding a *symbol*.

3.18 **context modelling**: The choice and specification of prior *decoded symbols* that are to be used in the *decoding* of a *symbol*.

3.19 context variable: Specified for each *symbol* by an equation containing the recently *decoded symbols* as defined by *context modelling*.

3.20 dangling field: A *field* for which there is no adjacent *field* carrying the same *frame number*. [Ed.Note: this is broken]

3.21 decoding order: The order in which the *coded pictures* are to be *decoded*. [Ed.Note: Main Profile further restricts decoding order within a coded picture, picture decoding order, NAL unit within a picture decoding order, decoding order within a coded slice: The order in which syntax elements of a coded slice are to be decoded.]

3.22 data partitioning: A method of *partitioning* selected *syntax elements* into syntactical structures based on a categorization of the *syntax elements*.

3.23 DC coefficient: The *transform coefficient* for which the frequency index is zero in both dimensions. [Ed.Note: define frequency index]

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

3.25 decoded pictures buffer: A buffer specified in the *video buffering verifier* in subclause C.1. The decoded picture buffer comprises the *reference picture buffer* and the *picture reordering buffer*. [Ed.Note: discussion ongoing: split into reference picture buffer and re-ordering buffer ?]

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

3.27 decoding process: The process specified in this Recommendation | International Standard that reads a *NAL unit stream* and produces decoded *fields* or *frames*.

3.28 direct prediction: An *inter prediction* for a *block* for which no *motion vector* is decoded.

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

3.30 emulation prevention byte: A byte having a fixed value present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive byte-aligned bytes in the *NAL unit* contains a *start code prefix*.

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

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

3.33 flag: A variable which can take one of only two possible values.

3.34 frame: A *frame* contains sampled and quantized luma and chroma data of all rows of a video signal *frame*. A *frame* consists of two *fields*, a *top field* and a *bottom field*. For interlaced video signal, one of these *fields* is sampled temporally later than the other.

Deleted: of a

3.35 intra prediction: A *prediction* derived from the decoded samples of the same *decoded picture*.

3.36 instantaneous decoder refresh (IDR) picture: A special *I picture* that causes the *decoder* to mark all *reference pictures* in the *decoded pictures buffer* [Ed.Note: see above] as un-used immediately before *decoding* the *IDR picture*, and to indicate that later *coded pictures* can be *decoded* without *inter prediction* from any *picture decoded* prior to the *IDR picture*.

3.37 inter coding: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses information from both, within the picture and from other pictures.

3.38 inter prediction: A *prediction* derived from decoded samples of *pictures* other than the current *decoded picture*. Inter prediction is a collective term for the prediction process in P, SP, and B macroblocks.

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

3.40 I picture: An *intra picture*; A *picture* that is coded using prediction only from decoded samples within the same *picture*.

3.41 inverse transform: A part of the *decoding process* by which a *block* of *scaled transform coefficient levels* is converted into a *block* of spatial-domain samples.

3.42 layer: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the picture, slice, reference picture selection, macroblock, 8x8 block and 4x4 block layers.

3.43 **level**: A defined set of constraints on the values which may be taken by the parameters of this Recommendation | International Standard. The same set of level definitions are used with all profiles, but individual implementations may support a different level for each supported profile. In a different context, *level* is the value of a *transform coefficient* prior to *scaling*.

3.44 **long SCP**: A *start code prefix* which is used in the construction of the *byte stream format* for *NAL unit streams*. It is mandatory at the start of a *coded picture* in *byte stream format*. Optionally it can be used instead of a *short SCP* at the start of a *coded slice* or lower *coding layers*.

3.45 **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.

3.46 **macroblock**: The 16x16 *luma* samples and the two corresponding blocks of *chroma* samples. [Ed.Note: There is a need to mention contiguity, non-overlapping and full coverage]

3.47 **macroblock address**: The *raster scan order* number of a *macroblock* starting with zero for the top left *macroblock* in a *picture*.

3.48 **macroblock allocation map**: A means of *partitioning* the *macroblocks* of a *picture* into *slice groups*. The *macroblock allocation map* consists of an array of numbers one for each *coded macroblock* indicating the *slice group* to which the *coded macroblock* belongs.

Formatted: Font: Italic

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

3.50 **macroblock pair**: A pair of vertically-contiguous macroblocks in a picture that is coupled for use in macroblock-adaptive frame/field decoder processing.

3.51 **Mbit**: 1 000 000 bits.

3.52 **motion compensation**: Part of the *inter prediction* process for sample values, using previously decoded samples that are spatially displaced as signalled by means of *motion vectors*.

3.53 **motion vector**: A two-dimensional vector used for *motion compensation* that provides an offset from the coordinate position in the *decoded picture* to the coordinates in a *reference picture*.

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

3.55 **NAL unit stream**: A sequence of *NAL units* containing the syntax structures associated with the coded video content.

3.56 **network abstraction layer (NAL)**: A definition of syntax structures and additional information including framing and timing that are supported by a system.

Deleted:

3.57 **non-reference picture**: a *decoded picture* that is marked as not used for *inter prediction*. [Ed.Note: this should be changed – How?]

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

3.59 **output order**: The order in which the *decoded pictures* are intended for output.

3.60 **output reordering delay**: A delay between decoding a *coded picture* and its output that is caused when the order of pictures specified for output is different from the order specified for decoding. [Ed.Note: is this really needed ?]

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

3.62 **partitioning**: The division of a set into sub-sets such that each element of the set is in exactly one of the sub-sets.

3.63 **P slice**: A *predictive slice*; A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference picture index* to *predict* the sample values of each *block*.

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

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

Formatted: Font: Italic

Formatted: Font: Italic

3.66 **picture reordering**: The process of re-ordering the decoded *pictures* when the *decoding order* is different from the *output order*.

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

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

3.69 prediction residual: The difference between the value of a source sample or data element and its *predictor*. [Ed.Note: This change is according to Gary. In the decoder side it is "residual". In the text all occurances of "prediction error" should be changed to "residual" (since prediction error is an encoder-only concept) and then the definition can be deleted]

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

3.71 probability model: The set of probability distributions used by the arithmetic decoding process when decoding a symbol. The *context* determines which probability distribution is to be used when decoding a particular symbol at a particular point, block, macroblock, etc. in the picture. For each symbol, the number of probability distributions in the set is equal to the number of possible values for the *context variable*, i.e., the number of *contexts*.[Ed.Note: remove this]

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

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

3.74 quarter common intermediate format (QCIF): A video frame that is 11 macroblocks wide and 9 macroblocks high.

3.75 random access: The ability to start the decoding of a coded NAL unit stream at a point other than the beginning of the stream and recover an exact or approximate representation of the decoded pictures represented by that NAL unit stream.

Deleted: d

3.76 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 row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc. rows of the pattern each scanned from left to right. [Ed.Note: inverse raster scan in decoding process]

3.77 reference field: A *reference field* is used for *inter prediction* when *P macroblocks*, *SP macroblocks*, and *B macroblocks* of a *coded field* or a *coded frame* are decoded.

3.78 reference frame: A *reference frame* is used for *inter prediction* when *P macroblocks*, *SP macroblocks*, and *B macroblocks* of a *coded frame* are decoded.

3.79 reference index list: A list of indices that is assigned to the *reference pictures* in the *reference picture buffer*.

3.80 reference index list 0: The list of *reference indices* for use in *reference list 0 prediction* for a *P*, *B*, or *SP slice*. All *inter prediction* used for *P* and *SP* slices is considered *reference list 0 prediction*. The *reference index list 0* is one of *two reference index lists* used for a *B slice*, with the other being the *reference index list 1*. [Ed.Note: Inter vs. P -> resolve]

3.81 reference list 0 motion vector: A *motion vector* associated with a *reference index* pointing into the *reference index list 0*.

3.82 reference list 0 prediction: *Inter prediction* of the content of a *slice* using a *reference index* into the *reference index list 0*.

3.83 reference index list 1: A list of *reference indices* defined for use in *inter prediction* for a *B slice*. The *reference index list1* is one of two lists of *reference indices* used by a *B slice*, with the other being the *reference index list 0*.

3.84 reference list 1 motion vector: A *motion vector* associated with a *reference index* pointing into the *reference index list 1*. [Ed. Note: Do we need the word "reference" in this term? How about just "list 1 prediction" ?]

3.85 reference list 1 prediction: *Inter prediction* of the content of a *B slice* using a *reference index* into the *reference index list 1*.

3.86 reference picture: A *picture* containing samples that are used for *inter prediction*.

3.87 reference picture buffer: A part of the decoded pictures buffer containing the *reference pictures*.

Deleted: (

Deleted: ?)

3.88 reference picture buffer management: specifies in the coded data, how the *decoding process* applies to the *decoded pictures buffer* and in particular to the *reference picture buffer*. [Ed.Note: revisit]

3.89 reserved: The term “reserved”, when used in the clauses defining 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 unless so specified.

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

3.91 **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 *block scan* order. In another context, the number of *skipped macroblocks*.

3.92 **sample aspect ratio (SAR)**: Specifies the ratio between the vertical displacement of the rows of *luma* samples in a *frame* and the horizontal displacement of the *luma* samples. Thus its units are (metres per row) ÷ (metres per sample).

Deleted: the distance between *luma* samples. It is defined as

Deleted: divided by

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

3.94 **short SCP**: A *start code prefix* which is used in the construction of a *byte stream format* of coded data. It can be used instead of a *long SCP* at the start of a *coded slice* or lower *coding layers*.

3.95 **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*.

3.96 **slice**: An integer number of *macroblocks* ordered contiguously in *raster scan order* within a particular *slice group*. Although a slice contains macroblocks that are contiguous in raster scan order within a slice group, these macroblocks are not necessarily contiguous within the picture. The addresses of the *macroblocks* are derived from the address of the first macroblock and the *slice group* parameters.

3.97 **slice group**: A sub-set of the *macroblocks* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture*. The partition is specified by the *slice group* parameters.

3.98 **slice header**: A part of a *coded slice* containing the *coded representation* of data elements pertaining to the *slice data* that follow the *slice header*.

3.99 **SI picture**: A *switching I picture*; A *picture* that is coded using prediction only from decoded samples within the same *picture*, encoded such that it can be reconstructed identically to another *SP picture* or *SI picture*, as specified in clause 11.

Deleted: *slice*

Deleted: *slice*

3.100 **SI slice**: A *switching I 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 another *SP slice* or *SI slice*, as specified in clause 11.

3.101 **source (input)**: Term used to describe the video material or some of its attributes before encoding.

3.102 **SP slice**: A *switching P slice*; A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference picture index* to *predict* the sample values of each *block*, encoded such that it can be reconstructed identically to another *SP slice* or *SI slice*, as specified in subclause 11.

3.103 **start code prefix (SCP)**: One of a set of unique codes embedded in the *byte-stream format* that are used for identifying the beginning of a coding layer. Emulation of *start code prefixes* is prohibited within NAL units.

3.104 **string of data bits (SODB):** An ordered sequence of some finite number of bits, in which the left-most bit is considered to be the first and most significant bit (MSB) and the right-most bit is considered to be the last and least significant bit (LSB).

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

3.106 **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*.

3.107 **transform coefficient**: A scalar considered to be in a frequency domain that is associated with a particular two-dimensional frequency index in the *inverse transform* of the decoding process.

3.108 **variable length coding (VLC)**: A reversible procedure for entropy coding that assigns shorter code-words to more frequent *symbols* and longer code-words to less frequent *symbols*.

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

3.110 **XYZ profile decoder**: A decoder able to decode coded data conforming to the specifications of the XYZ profile (with XYZ being any of the defined Profile names).

3.111 **zig-zag scan**: A specific sequential ordering of *transform coefficients* from (approximately) the lowest spatial frequency to the highest.

4 Abbreviations

4.1 **ABT**: Adaptive Block size Transform

4.2 **ASO**: Arbitrary Slice Order

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

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

4.5 **CIF**: Common Intermediate Format

4.6 **DPA**: Data Partition type A

4.7 **DPB**: Data Partition type B

4.8 **DPC**: Data Partition type C

4.9 **FCC**: Federal Communications Commission

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

4.11 **FMO**: Flexible Macroblock Ordering

4.12 **IDR**: Instantaneous Decoder Refresh

4.13 **LPS**: Least Probable Symbol

4.14 **LSB**: Least Significant Bit

4.15 **MB**: Macroblock

4.16 **MPS**: Most Probable Symbol

4.17 **MSB**: Most Significant Bit

4.18 **NAL**: Network Abstraction Layer

4.19 **QCIF**: Quarter Common Intermediate Format

4.20 **RBSP**: Raw Byte Sequence Payload

4.21 **SAR**: Sample Aspect Ratio

4.22 **SCP**: Start Code Prefix

4.23 **SEI**: Supplemental Enhancement Information

4.24 **SMPTE:** Society of Motion Picture and Television Engineers

4.25 **SODB**: String Of Data Bits

4.26 **VCL**: Video Coding Layer

4.27 **VBV**: Video Buffering Verifier

4.28 **VLC**: Variable Length Coding

Deleted: **4.1** . **ABT**: Adaptive Block size Transform¶
4.2 . **CABAC**: Context-based Adaptive Binary Arithmetic Coding¶
4.3 . **CAVLC**: Context-based Adaptive Variable Length Coding¶
4.4 . **CIF**: Common Intermediate Format¶
4.5 . **DPA**: Data Partition type A¶
4.6 . **DPB**: Data Partition type B¶
<#>**DPC**: Data Partition type C¶
4.8 . **FCC**: Federal Communications Commission¶
4.9 . **FIFO**: First-In, First-Out¶
4.10 . **IDR**: Instantaneous Decoder Refresh¶
4.11 . **LPS**: Least Probable Symbol¶
4.12 . **LSB**: Least Significant Bit¶
4.13 . **MB**: Macroblock¶
4.14 . **MPS**: Most Probable Symbol¶
4.15 . **MSB**: Most Significant Bit¶
4.16 . **NAL**: Network Abstraction Layer¶
4.17 . **QCIF**: Quarter Common Intermediate Format¶
4.18 . **RBSP**: Raw Byte Sequence Payload¶
4.19 . **SAR**: Sample Aspect Ratio¶
4.20 . **SCP**: Start Code Prefix¶
4.21 . **SEI**: Supplemental Enhancement Information¶
4.22 . **SMPTE:** Society of Motion Picture and Television Engineers¶
4.23 . **SODB**: String Of Data Bits¶
4.24 . **VCL**: Video Coding Layer¶
4.25 . **VBV**: Video Buffering Verifier¶
4.26 . **VLC**: Variable Length Coding

5 Conventions

The mathematical operators used to describe this Specification are similar to those used in the C programming language. However, integer divisions with truncation and rounding are specifically defined. Numbering and counting loops generally begin from zero.

5.1 Arithmetic operators

The following mathematical and logical operators are defined as follows

+	Addition
–	Subtraction (as a binary operator) or negation (as a unary operator)
++	Increment, i.e. $x{+}{+}$ is equivalent to $x = x + 1$
––	Decrement, i.e. $x{-}{-}$ is equivalent to $x = x - 1$

$\left.\begin{matrix}\times \\ *\end{matrix}\right\}$	Multiplication
^	Power
/	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.
DIV	Integer division with truncation of the result toward minus infinity. For example 3 DIV 2 is rounded to 1, and –3 DIV 2 is rounded to –2.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=a}^{b} f(i)$	The summation of the $f(i)$ with i taking all integer values from a up to and including b.
$a \% b$	Modulus operator. Remainder of a divided by b, defined only for a and b both positive integers

5.2 Logical operators

a && b	Boolean logical "and" of a and b
a || b	Boolean logical "or" of a and b
!	Logical NOT

5.3 Relational operators

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

5.4 Bit-wise operators

&	AND
|	OR
a >> b	Arithmetic right shift of a two's complement integer representation of a by b binary digits. This function is defined only for positive values of b. Bits shifted into the MSBs as a result of the right shift shall have a value equal to the MSB of a prior to the shift operation.
a << b	Arithmetic left shift of a two's complement integer representation of a by b binary digits. This function is defined only for positive values of b.

5.5 Assignment

=	Assignment operator
x = a..b	x takes on values starting from a to b inclusive, with x, a, and b being integer numbers.

Formatted: Bullets and Numbering

Deleted: =

Formatted: Indent: Left: 1,4 cm

5.6 Functions

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

$$\mathrm{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \tag{5-2}$$

$$\text{Clip3(a, b, c)} = \begin{cases} a & ; \quad c < a \\ b & ; \quad c > b \\ c & ; \quad \text{otherwise} \end{cases} \qquad (5\text{-}3)$$

$$\text{Clip1}(x) = \text{Clip3}(0, 255, x) \qquad (5\text{-}4)$$

Ceil(x) rounds x up to the nearest integer. Defined only for non-negative values of x. (5-5)

Log2(x) returns the base-2 logarithm of x. (5-6)

6 Source, coded, decoded, and output data formats

6.1 Bitstream formats

This subclause specifies the relationship between the NAL unit stream and Byte stream both referred to as the bitstream.

The bitstream can be in one of two formats: segmented or non-segmented. The segmented format is the more "basic" type and the non-segmented format can be constructed from the segmented format.

Each of the segments in the segmented format is called a NAL unit which is a byte aligned bitstream, and hence the segmented format is called NAL unit stream. There are constraints imposed on the order (and contents) of the NAL units in the NAL unit stream. Delimiting of the NAL units is outside the scope of this specification.

The non-segmented format called Byte stream format is constructed from the NAL unit stream as described 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.

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 decoder 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).

Deleted: are

[Ed. Note: There are two assumptions "hidden" in the above text. First, that both coded frames and coded fields can refer to the same sequence parameter set. Second, that the height specified is that of a frame and not a field. Are these assumptions correct? If yes, may be the parameter names should mention frames and not pictures.]

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).

Deleted: part of the sequence parameter set which is referred by the picture parameter set which is referred by the slice

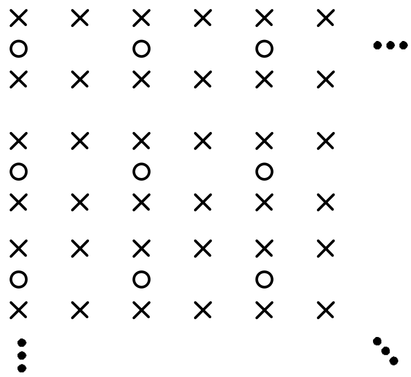


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

[EdNote: In Figure 6.1 change "sample" to "samples"]

This Recommendation | International Standard describes decoding of video that contains either progressive-scan [Ed. Note: add definition for progressive-scan] or interlaced-scan frames, which may be mixed together in the same sequence.

Source and decoded fields are one of two types: top field or bottom field. The nominal vertical and horizontal relative locations of luma and chroma samples in fields are shown in Figure 6-2. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

Deleted: part of the sequence parameter set which is referred by the picture parameter set which is referred by the slice

When two fields are output at the same time, or combined to be used as a reference frame (see below), the two fields (which has to 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 picture consists of only the top field rows of a decoded frame. When the top field or bottom field of a frame is used as a reference field (see below) only the odd rows (for a top field) or the even rows (for a bottom field) are used.

The two decoded fields of an interlaced frame are separated in time. They may be decoded separately as two fields or together as a frame.

NOTE - A progressive frame should always be coded as a single frame picture. However, a progressive frame is still considered to consist of two fields (at the same instant in time) and could be coded as an interlaced frame.

The nominal vertical and horizontal relative locations of luma and chroma samples in interlaced fields are shown in Figure 6-2. The vertical sampling relative locations of the chroma samples in a top field of an interlaced frame are specified as shifted up by 1/4 luma sample height relative to the field-sampling grid in order for these samples to align vertically to the usual location relative to the full-frame sampling grid. The vertical sampling locations of the chroma samples in a bottom field of an interlaced frame are specified as shifted down by 1/4 luma sample height relative to the field-sampling grid in order for these samples to align vertically to the usual location relative to the full-frame sampling grid. The horizontal sampling relative locations of the chroma samples are specified as unaffected by the application of interlaced field coding.

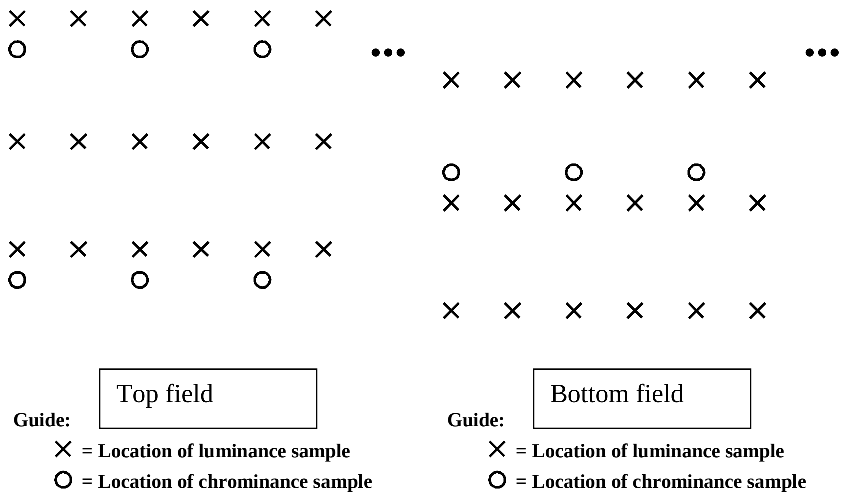


Figure 6-2 – Nominal vertical and temporal sampling locations of samples in 4:2:0 interlaced frames

6.3 Spatial subdivision of a picture into macroblocks

This subclause specifies how a picture is partitioned into macroblocks and the order in which macroblocks are scanned.

Pictures are divided into macroblocks. The division is done by partitioning the luma array into 16*16 sample arrays and partitioning each of the two chroma arrays into 8*8 sample arrays. Each macroblock is comprised of a one 16*16 luma and two 8*8 chroma arrays. For instance, a QCIF picture is divided into 99 macroblocks as indicated in Figure 6-3

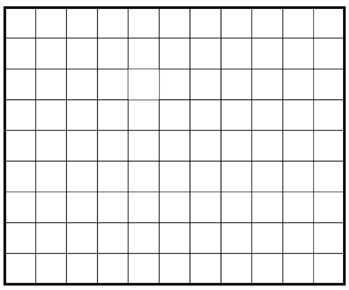


Figure 6-3 – A picture with 11 by 9 macroblocks (QCIF picture)

A macroblock location in the picture is given by a pair of indices where the first index is the macroblock column index starting with 0 for the leftmost column, and the second index is the macroblock row index starting with 0 for the topmost row.

When mb_frame_field_adaptive_flag (which can be found in the sequence parameter set) is equal to 0, the macroblock address is given by a single index allocated to the macroblock in raster scan order of macroblocks starting with 0 for the top leftmost macroblock. When mb_frame_field_adaptive_flag (which can be found in the sequence parameter set) is equal to 1, the macroblock address is given by a single index allocated to the macroblock in raster scan order of vertical macroblock pairs starting with 0 for the top macroblock of the top leftmost macroblock pair, continuing with 1 for the bottom macroblock of the top leftmost macroblock pair (see Figure Figure 6-4).

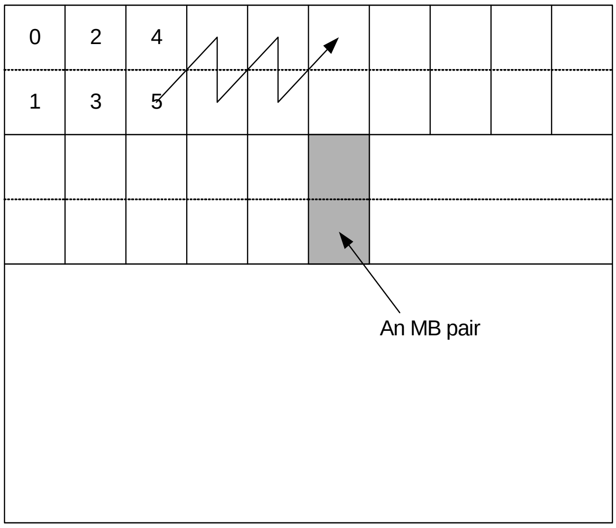


Figure 6-4 – Partitioning of the decoded frame into macroblock pairs. An macroblock pair can be coded as two frame macroblocks, or one top-field macroblock and one bottom-field macroblock. The numbers indicate the scanning order of coded macroblocks.

[Ed.Note: more explananation needed here]

When mb_adaptive_frame_field_flag in the picture parameter set is 0 [EdNote: There is no such parameter in the picture parameter set. In the text, mb_adaptive_frame_field_flag is always mentioned in a conditional sense. However mb_frame_field_adaptive_flag can be found in the sequence parameter set!], the macroblock address calculation is recursively specified as follows:

1. A coded slice contains in its slice header the macroblock address of the first macroblock in the coded slice. The macroblock allocation map conveys the slice group identifier of the first macroblock in a slice.

2. Let g be the slice group identifier of the most recently decoded macroblock of a given coded slice. The next macroblock address is found by searching the macroblock allocation map in scan order for the next macroblock that has the same slice group identifier g.

When mb_adaptive_frame_field_flag in the picture parameter set is 1, the macroblock pair address calculation is recursively specified as follows (see Figure 6-4):

1. A coded slice contains in its slice header the macroblock pair address of the first macroblock pair in the coded slice. The slice group index of the first macroblock pair in a coded slice may be found by referencing the macroblock allocation map.

2. Let g be the slice group identifier of the most recently decoded macroblock pair of a given coded slice. The next macroblock pair address is found by searching the macroblock allocation map in scan order for the next macroblock pair that has the same slice group identifier g.

NOTE - This note describes one of many possible implementations of the macroblock address calculation (the macroblock pair address could be calculated with a similar algorithm).

Assume the availability of a one-dimensional array m with as many entries as there are macroblocks or macroblock pairs in the picture, and that is initialized with a one-dimensional representation of the macroblock allocation map of the picture parameter set (in one of the several explicit or implicit forms specified there). Let n be the macroblock address of the last decoded macroblock of a given slice. The next macroblock address is calculated by the following three steps:

1. Identify the slice group of the macroblock n by using n as an index into m.
2. Search in m in ascending order, starting with n, for the next entry that has the same slice group identifier.
3. This is the macroblock address for the next macroblock or macroblock pair of the coded slice.

NOTE - a coded slice may consist of one slice NAL unit or, when data partitioning is used, of three NAL units DPA, DPB, and DPC.

6.4 Spatial subdivision of a macroblock and macroblock coefficients order assignment

This subclause specifies how the macroblock layer syntax elements in decoding order are mapped to samples inside a macroblock.

For some decoding operations a macroblock or sub-macroblock may be further divided into smaller blocks as shown in Figure 6-5.

The partitioning of a macroblock is indicated by its macroblock type in non-intra slices. The leftmost macroblock type refers to the 16x16 luma sample array and the corresponding two 8x8 chroma sample arrays. In this case, the motion-compensated samples are calculated maybe using at most two motion vectors and at most two reference picture indices. The motion compensation process is applied to all luma and chroma samples of this macroblock as specified in subclause 8.4. The three other types of macroblock partitioning are shown to the right of the leftmost macroblock type. Also shown are the assignment of the motion vectors and reference picture indices to blocks of luma and associated chroma samples. [Ed.Note: this needs further description …]

If the ABT feature is used, the transform for residual coding is adapted to the sub-macroblock partitioning pattern as well. [Ed.Note: this needs further description …]

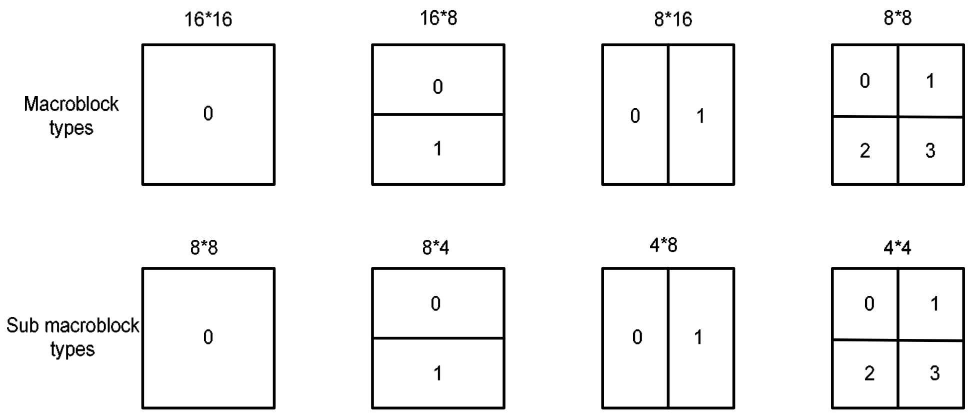


Figure 6-5 – Macroblock subdivision types and blocks order.

Figure 6-6 shows the order of the assignments of syntax elements resulting from coding a macroblock to sub-blocks of the macroblock if the ABT feature is not used. The assignment order if the ABT feature is used is specified in Figure 12-1. [Ed.Note: exchange CBPY by coded_block_patternY and this needs further description.]

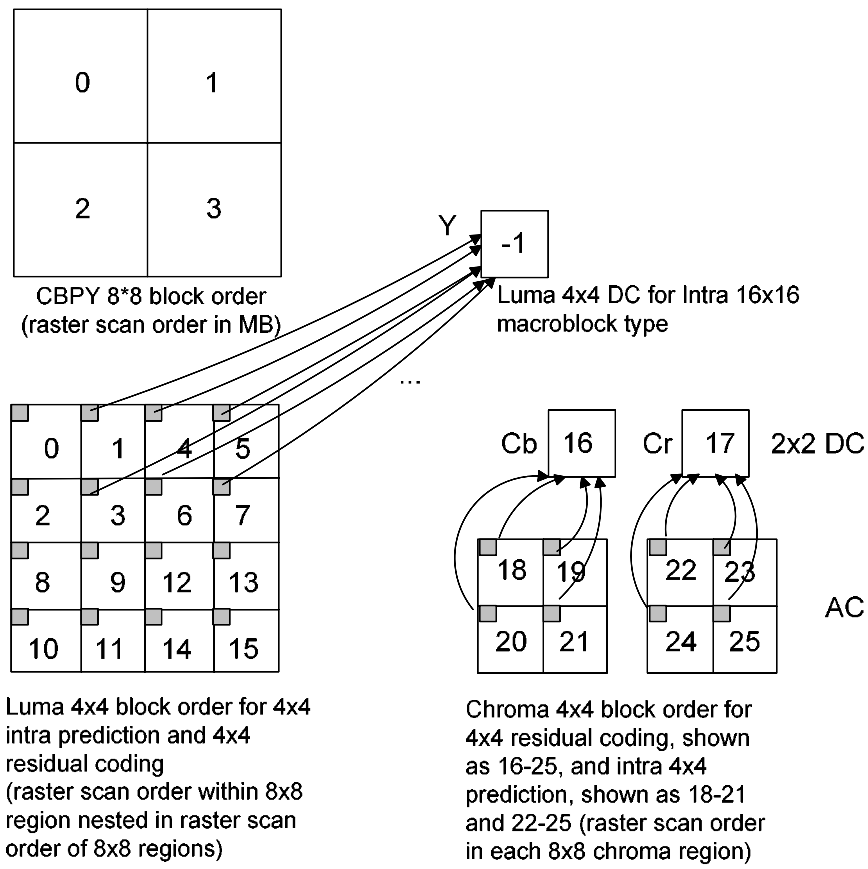


Figure 6-6 – Ordering of blocks for coded_block_patternY, 4x4 intra prediction, and 4x4 residual coding

7 Syntax and semantics

7.1 Method of describing the syntax in tabular form

The syntax is described in a manner that closely follows the C-language syntactic constructs. Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name, its syntax category and descriptor for its method of coded representation. A decoder behaves according to the value of the syntax element and on the values of previously decoded syntax elements.

The syntax tables describe a superset of the syntax of all correct and error-free input bitstreams. [Ed.Note: remove error or change to NOTE] Additional constraints on the syntax form may be specified in other clauses. An actual decoder must implement correct means for identifying entry points into the bitstream for proper decoding and to identify and handle errors in the bitstream [Ed.Note: remove error or change to NOTE]. The methods for identifying and handling errors and other such situations are not described here.

Following C-language conventions, a value of '0' represents a FALSE condition in a test statement. The value TRUE is represented by '1', but any other value different than zero is also understood as TRUE.

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

[Ed.Note: specify function of functions, specify passing of parameters, explain bold vs. not bold syntax elements, specify variables PayloadType …, mention meaning of vertical bar in category and descriptor]

	Category	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	e(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 indicates a test of whether a condition is true, and if true, indicates evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */		
while(condition)		
statement		
/* A "do … while" structure indicates evaluation of a statement once, followed by a test of whether a condition is true, and if true, indicates repeated evaluation of the statement until the condition is no longer true */		
do		
statement		
while(condition)		
/* An "if … else" structure indicates a test of whether a condition is true, and if the condition is true, indicates evaluation of a primary statement, otherwise indicates 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 indicates evaluation of an initial statement, followed by a test of a condition, and if the condition is true, indicates 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 Definitions of functions 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 decoder from the bitstream. [Ed.Note: fix bitstream vs. byte stream vs. coded stream vs. NAL unit stream]

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

first_non_skip_mb_in_pair( )

- Returns TRUE if the current macroblock is the first macroblock in a macroblock pair or if the previous macroblock in the macroblock pair was skipped. Used only in macroblock-adaptive frame/field coding.

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 an RBSP syntax structure or in a structure within an RBSP syntax structure, returns a non-matching value if fewer than *n* bits remain within the RBSP prior to the rbsp_trailing_bits( ). If used within the byte stream format syntax specified in Annex B, returns a non-matching value if fewer than *n* bits remain within the byte stream.

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 slices is specified by the system (or in Annex B for systems that use the byte stream format).

total_coeff( )

- Returns the number of coefficients from coeff_token. See subclause 7.3.5.3.1.

trailing_ones( )

- Returns the trailing ones from coeff_token. See subclause 7.3.5.3.1.

slice_type( )

- Returns the coding type of a slice.

The following descriptors are used to describe the type of each syntax element.

- **b(8):** byte having any value (8 bits).
- **ue(v)**: unsigned integer Exp-Golomb-coded syntax element with the left bit first.
- **se(v)**: signed integer Exp-Golomb-coded syntax element with the left bit first.
- **me(v)**: mapped Exp-Golomb-coded syntax element with the left bit first.
- **ce(v)**: context-adaptive variable-length entropy-coded syntax element with the left bit first.
- **f(n)**: fixed-value bit string using *n* bits written (from left to right) with the left bit first.
- **i(n)**: signed integer using *n* bits for a two's complement representation with most significant bit written first. If *n* is "v", the number of bits varies in a manner dependent on the value of other decoded data.
- **u(n)**: unsigned integer using *n* bits with most significant bit written first. If *n* is "v", the number of bits varies in a manner dependent on the value of other decoded data.
- **xe(v)**: extended Exp-Golomb-coded syntax element with left bit first. If indicating a selection from a list having only two alternatives, shall be interpreted as u(1). If indicating a selection from a list having more than two alternatives, shall be interpreted as ue(v).
- **ae(v)**: context-adaptive arithmetic entropy-coded syntax element. Some syntax elements are coded using CABAC when entropy_coding_mode = = 1. This is indicated by specifiying an alternative descriptor separated by a bar.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

nal_unit(NumBytesInNALunit) {	**Category**	**Descriptor**
forbidden_bit		u(1)
nal_storage_idc		u(2)
nal_unit_type		u(5)
NumBytesInRBSP = 0		
for(i = 0; i < NumBytesInNALunit-1; i++) {		
if(next_bits(16) = = 0x0003) {		
rbsp[NumBytesInRBSP++ **]**		b(8)
i++		
emulation_prevention_byte /* = = 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( ) {	**Category**	**Descriptor**
profile_idc	0	ue(v)
level_idc	0	ue(v)
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_minus4	0	ue(v)
else if(pic_order_cnt_type = = 1) {		
offset_for_non_stored_pic	0	se(v)
num_stored_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_stored_frames_in_pic_order_cnt_cycle; i++)		
offset_for_stored_frame[i **]**	0	se(v)
}		
num_of_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_mbs_minus1	0	ue(v)
filter_parameters_flag	0	u(1)
constrained_intra_pred_flag	0	u(1)
mb_frame_field_adaptive_flag	0	u(1)
vui_seq_parameters_flag	0	u(1)
if(vui_seq_parameters_flag)		
vui_seq_parameters( )	0	
rbsp_trailing_bits( )	0	
}		

Formatted: Font: Not Bold

Deleted: **required_frame_num_update_behaviour**

Deleted: **send_filter_parameters_flag**

[Ed.Note: Syntax structure functions should have category assignments, mention that parameters of parameter sets are global]

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	**Category**	**Descriptor**
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode	1	ue(v)
motion_resolution	1	ue(v)
adaptive_block_size_transform_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
mb_allocation_map_type	1	ue(v)
if(mb_allocation_map_type = = 0)		
for(i = 0; i <= num_slice_groups_minus1; i++)		
run_length	1	ue(v)
else if(mb_allocation_map_type = = 2)		
for(i = 0; i < num_mbs_in_pic; i++)		
slice_group_id [Ed. Note: how many bits ?]	1	u(v)
else if(mb_allocation_map_type = = 3)		
for(i = 0; i < num_slice_groups_minus1; i++) {		
top_left_mb	1	u(v)
bottom_right_mb	1	u(v)
}		
else if(mb_allocation_map_type = = 4 || mb_allocation_map_type = = 5 || mb_allocation_map_type = = 6) {		
slice_group_change_direction	1	u(1)
slice_group_change_rate_minus1	1	ue(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_explicit_flag	1	u(1)
weighted_bipred_implicit_flag	1	u(1)
slice_qp_minus26 /* relative to 26 */	1	se(v)
slice_qp_s_minus26 /* relative to 26 */	1	se(v)
redundant_slice_flag	1	u(1)
vui_pic_parameters_flag	1	u(1)
if(vui_pic_parameters_flag) {		
vui_pic_parameters( )	1	
}		
rbsp_trailing_bits( )	1	
}		

Formatted: Font: Bold

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	**Category**	**Descriptor**
do		
sei_message( )	7	
while(more_rbsp_data( ))		
rbsp_trailing_bits( )	7	
}		

7.3.2.3.1 Supplemental enhancement information message syntax

sei_message( ) {	**Category**	**Descriptor**
PayloadType = 0		
while(next_bits(8) = = 0xFF) {		
byte_ff /* equal to 0xFF */	7	u(8)
PayloadType += 255		
}		
last_payload_type_byte	7	u(8)
PayloadType += last_payload_type_byte		
PayloadSize = 0		
while(next_bits(8) = = 0xFF) {		
byte_ff	7	u(8)
PayloadSize += 255		
}		
last_payload_size_byte	7	u(8)
PayloadSize += last_payload_size_byte		
sei_payload(PayloadType, PayloadSize)	7	
}		

7.3.2.4 Picture delimiter RBSP syntax

pic_delimiter_rbsp( ) {	**Category**	**Descriptor**
three_reserved_bits	8	u(3)
pic_type	8	u(3)
non_stored_pic_flag	8	u(1)
rbsp_trailing_bits( )	8	
}		

7.3.2.5 Filler data RBSP syntax

filler_data_rbsp(NumBytesInRBSP) {	**Category**	**Descriptor**
while(next_bits(8) = = 0xFF)		
byte_ff	9	f(8)
rbsp_trailing_bits( )	9	
}		

7.3.2.6 Slice layer RBSP syntax

slice_layer_no_partitioning_rbsp( ) {	**Category**	**Descriptor**
slice_header( )	4	
slice_data( ) /* all categories of slice_data( ) syntax */	4 | 5 | 6	
rbsp_slice_trailing_bits( )	4	
}		

7.3.2.7 Data partition RBSP syntax

7.3.2.7.1 Data partition A RBSP syntax

dpa_layer_rbsp( ) {	**Category**	**Descriptor**
slice_header( )	4	
slice_id	4	ue(v)
slice_data( ) /* only the category 4 parts of slice_data( ) syntax */	4	
rbsp_slice_trailing_bits( )	4	
}		

7.3.2.7.2 Data partition B RBSP syntax

dpb_layer_rbsp( ) {	**Category**	**Descriptor**
slice_id	5	ue(v)
slice_data( ) /* only the category 5 parts of slice_data( ) syntax */	5	
rbsp_slice_trailing_bits( )	5	
}		

7.3.2.7.3 Data partition C RBSP syntax

dpc_layer_rbsp( ) {	**Category**	**Descriptor**
slice_id	6	ue(v)
slice_data( ) /* only the category 6 parts of slice_data( ) syntax */	6	
rbsp_slice_trailing_bits( )	6	
}		

7.3.2.8 RBSP trailing bits syntax

rsbp_trailing_bits( ) {	**Category**	**Descriptor**
rbsp_stop_bit /* equal to 1 */	All	f(1)
while(!byte_aligned( ))		
rbsp_alignment_bit /* equal to 0 */	All	f(1)
}		

7.3.2.9 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits( ) {	**Category**	**Descriptor**
rbsp_stop_bit /* equal to 1 */	All	f(1)
if(entropy_coding_mode = = 1)		
while(next_bits(1) = = '1')		
cabac_stuffing_bit /* equal to 1 */	All	f(1)
while(!byte_aligned( ))		
rbsp_alignment_bit /* equal to 0 */	All	f(1)
}		

7.3.3 Slice header syntax

slice_header( ) {	**Category**	**Descriptor**
pic_parameter_set_id	4	ue(v)
frame_num	4	u(v)
pic_structure	4	ue(v)
first_mb_in_slice	4	u(v)
slice_type_idc	4	ue(v)
if(pic_order_cnt_type = = 0)		
pic_order_cnt	4	u(v)
else if(pic_order_cnt_type = = 1)		
delta_pic_order_cnt	4	se(v)
if(redundant_slice_flag)		
redundant_pic_cnt	4	ue(v)
if(slice_type_idc = = BiPred)		
direct_spatial_mv_pred_flag	4	u(1)
num_ref_idx_active_override_flag	4	u(1)
if(num_ref_idx_active_override_flag) {		
if(slice_type_idc = = Pred || slice_type_idc = = SPred || slice_type_idc = = BiPred) {		
num_ref_idx_l0_active_minus1	4	ue(v)
if(slice_type_idc = = BiPred)		
num_ref_idx_l1_active_minus1	4	ue(v)
}		
}		
ref_idx_reordering( )	4	
if((weighted_pred_flag && ((slice_type_idc = = Pred) || (slice_type_idc = = SPred))) || (weighted_bipred_explicit_flag && (slice_type_idc = = BiPred)))		
pred_weight_table( )	4	
ref_pic_buffer_management( )	4	
slice_qp_delta	4	se(v)
if(slice_type_idc = = SPred || slice_type_idc = = SIntra) {		
if(slice_type_idc = = SPred)		
sp_for_switch_flag	4	u(1)
slice_qp_s_delta	4	se(v)
}		
if(filter_parameters_flag = = 1) {		
disable_deblocking_filter_flag	4	u(1)
if(!disable_deblocking_filter_flag) {		
slice_alpha_c0_offset_div2	4	se(v)
slice_beta_offset_div2	4	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && mb_allocation_map_type >= 4 && mb_allocation_map_type <= 6)		
slice_group_change_cycle	4	u(v)
}		

Deleted: send_filter_parameters_flag

7.3.3.1 Reference index reordering syntax

ref_idx_reordering( ) {	**Category**	**Descriptor**
if(slice_type( ) != Intra && slice_type( ) != SIntra) {		
ref_idx_reordering_flag_l0	4	u(1)
if(ref_idx_reordering_flag_l0) {		
do {		
remapping_of_pic_nums_idc	4	ue(v)
if(remapping_of_pic_nums_idc = = 0 || remapping_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	4	ue(v)
else if(remapping_of_pic_nums_idc = = 2)		
long_term_pic_idx	4	ue(v)
} while(remapping_of_pic_nums_idc != 3)		
}		
}		
if(slice_type( ) = = BiPred) {		
ref_idx_reordering_flag_l1	4	u(1)
if(ref_idx_reordering_flag_l1) {		
do {		
remapping_of_pic_nums_idc	4	ue(v)
if(remapping_of_pic_nums_idc = = 0 || remapping_of_pic_nums_idc = = 1)		
abs_diff_pic_num_minus1	4	ue(v)
else if(remapping_of_pic_nums_idc = = 2)		
long_term_pic_idx	4	ue(v)
} while(remapping_of_pic_nums_idc != 3)		
}		
}		
}		

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	**Category**	**Descriptor**
luma_log_weight_denom	4	ue(v)
chroma_log_weight_denom	4	ue(v)
for(i =0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_flag_l0	4	u(1)
if(luma_weight_flag_l0) {		
luma_weight_l0[i **]**	4	se(v)
luma_offset_l0[i **]**	4	se(v)
}		
chroma_weight_flag_l0	4	u(1)
if(chroma_weight_flag_l0)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i **][** j **]**	4	se(v)
chroma_offset_l0[i **][** j **]**	4	se(v)
}		
}		
if(slice_type( ) = = BiPred) {		
for(i = 0; i<= num_ref_idx_l1_active_minus1num; i++) {		
luma_weight_flag_l1	4	u(1)
if(luma_weight_flag_l1) {		
luma_weight_l1[i **]**	4	se(v)
luma_offset_l1[i **]**	4	se(v)
}		
chroma_weight_flag_l1	4	u(1)
if(chroma_weight_flag_l1)		
for(j = 0; j < 2; j++) {		
chroma_weight_l1[i **][** j **]**	4	se(v)
chroma_offset_l1[i **][** j **]**	4	se(v)
}		
}		
num_custom_bipred_weights	4	ue(v)
for(i=0; i < num_custom_bipred_weights; i++) {		
if(num_ref_idx_l0_active_minus1> 0)		
irp_l0	4	xe(v)
if(num_ref_idx_l1_active_minus1 > 0)		
irp_l1	4	xe(v)
luma_weight_bipred_l0[irp_l0 **][** irp_l1 **]**	4	se(v)
luma_weight_bipred_l1[irp_l0 **][** irp_l1 **]**	4	se(v)
luma_offset_bipred[irp_l0 **][** irp_l1 **]**	4	se(v)
chroma_weight_flag_bipred[irp_l0 **][** irp_l1 **]**	4	u(1)
if (chroma_weight_flag_bipred[irp_l0][irp_l1])		
for(j = 0; j < 2; j++) {		
chroma_weight_bipred_l0[irp_l0 **][** irp_l1 **][** j **]**	4	se(v)
chroma_weight_bipred_l1[irp_l0 **][** irp_l1 **][** j **]**	4	se(v)
chroma_offset_bipred[irp_l0 **][** irp_l1 **][** j **]**	4	se(v)
}		

}		
}		
}		

7.3.3.3 Reference picture buffer management syntax

ref_pic_buffer_management( ) {	**Category**	**Descriptor**
ref_pic_buffering_mode	4 | 7	u(1)
if(ref_pic_buffering_mode = = 1)		
do {		
memory_management_control_operation	4 | 7	ue(v)
if(memory_management_control_operation = = 1 || memory_management_control_operation = = 3)		
difference_of_pic_nums_minus1	4 | 7	ue(v)
if(memory_management_control_operation = = 2 || memory_management_control_operation = = 3)		
long_term_pic_idx	4 | 7	ue(v)
if(memory_management_control_operation = = 4)		
max_long_term_pic_idx_plus1	4 | 7	ue(v)
} while(memory_management_control_operation ! = 0 && memory_management_control_operation ! = 5)		
}		

7.3.4 Slice data syntax

slice_data( ) {	**Category**	**Descriptor**

if(mb_frame_field_adapative_flag &&
(pic_structure = = 0 ||
pic_structure = = 3 ||
pic_structure = = 4)) {		

MbPairY = first_mb_in_slice / (pic_width_in_mb_minus1 + 1)		
MbPairX = first_mb_in_slice % (pic_width_in_mb_minus1+1)		
MbNum = (MbPairY << 1) * (pic_width_in_mb_minus1 + 1) + MbPairX		
} else		
MbNum = first_mb_in_slice		
MoreDataFlag = 1		
do {		
if(slice_type( ) ! = Intra && slice_type( ) ! = SIntra)		
if(entropy_coding_mode = = 0) {		
mb_skip_run	4	ue(v)
MoreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	4	ae(v)
MoreDataFlag = !mb_skip_flag		
}		
if(MoreDataFlag) {		

if(mb_frame_field_adaptive_flag &&
(pic_structure = = 0 ||
pic_structure = = 3 || pic_structure = = 4) &&
(slice_type( ) ! = BiPred)) {		

if(MbNum % 2 = = 0)		
MbFieldDecodingFlag = 1		
if(MbFieldDecodingFlag) {		
mb_field_decoding_flag	4	u(1) | ae(v)
MbFieldDecodingFlag = 0		
} else		
mb_field_decoding_flag = 0		
}		
if(adaptive_block_size_transform_flag = = 0)		
macroblock_layer( )	4 | 5 | 6	
else		
macroblock_layer_abt( )	4 | 5 | 6	
if(entropy_coding_mode = = 0)		
MoreDataFlag = more_rbsp_data( )		
else {		
if(MbNum < MaxMbAddress) {		
if(mb_frame_field_adaptive_flag && (pic_structure = = 0 || pic_structure = = 3 || pic_structure = = 4) &&		
MbNum % 2 ! = 0)		
MoreDataFlag = 1		
else {		
end_of_slice_flag	4	ae(v)

Deleted: MAX_MB_ADDRESS

MoreDataFlag = !end_of_slice_flag		
} else		
MoreDataFlag = 0		
}		
MbNum++ [Ed. Note: is this correct with FMO ?]		
} while(MoreDataFlag)		
}		

[Ed.Note: restriction that for mb_frame_field_adaptive_flag = = 1, slice size must be even number of mbs must be worked in. AFF open issue.]

NOTE – macroblock_layer_abt( ) is specified in subclause 12.2.1.

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	**Category**	**Descriptor**
mb_type	4	ue(v) | ae(v)
if(num_mb_partition[mb_type] = = 4)		
sub_mb_pred(mb_type)	4	
else		
mb_pred(mb_type)	4	
SendResidual = 0		
if(mb_partition_pred_mode(mb_type, 1) = = Intra && mb_type ! = Intra_4x4) /* Intra_16x16_X_Y_Z mb_type */		
SendResidual = 1		
else {		
coded_block_pattern	4	me(v) | ae(v)
if(coded_block_pattern > 0)		
SendResidual = 1		
}		
if(SendResidual) {		
if(!mb_frame_field_adaptive_flag | | (mb_frame_field_adaptive_flag && (pic_structure = = 0 | | pic_structure = = 3 | | pic_structure = = 4) && first_non_skip_mb_in_pair( ))		
delta_qp	4	se(v) | ae(v)
residual( )	5 | 6	
}		
}		

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	**Category**	**Descriptor**
if(mb_partition_pred_mode(mb_type, 1) = = Intra) {		
if(mb_type = = Intra_4x4 | | mb_type = = SIntra_4x4)		
for(i = 0; i < num_mb_intra_partition(mb_type); i++) /* for each 4x4 luma block */		
intra_pred_mode	4	ce(v) | ae(v)
intra_chroma_pred_mode	4	ue(v) | ae(v)
} else if(mb_type ! = Direct_16x16) {		
for(i = 0; i < num_mb_partition(mb_type); i++)		
if(num_ref_idx_l0_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) ! = Pred_L1)		
ref_idx_l0 [Ed. Note: treat special case of 2 ref pics]	4	ue(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(num_ref_idx_l1_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) ! = Pred_L0)		
ref_idx_l1 [Ed. Note: treat special case of 2 ref pics]	4	ue(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode (mb_type, i) ! = Pred_L1)		
for(j = 0; j < 2; j++)		
mvd_l0[i **][** j **]**	4	se(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode(mb_type, i) ! = Pred_L0)		
for(j = 0; j < 2; j++)		
mvd_l1[i **][** j **]**	4	se(v) | ae(v)
}		
}		

7.3.5.2 Sub macroblock prediction syntax

sub_mb_pred(mb_type) {	Category	Descriptor
for(i = 0; i < 4; i++) /* for each sub macroblock */		
sub_mb_type[i **]**	4	ue(v) | ae(v)
IntraChromaPredModeFlag = 0		
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] = = Intra_8x8)		
for(j = 0; j < num_sub_mb_intra_partition(sub_mb_type[i]); j++) { /* num_sub_mb_intra_partition( ) = 4 */		
intra_pred_mode	4	ce(v) | ae(v)
IntraChromaPredModeFlag = 1		
}		
if(IntraChromaPredModeFlag)		
intra_chroma_pred_mode	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l0_active_minus1 > 0 && mb_type ! = Pred_8x8ref0 && sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L1)		
ref_idx_l0	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l1_active_minus1 > 0 && (sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L0)		
ref_idx_l1	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L1)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l0[i **][** j **][** k **]**	4	se(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L0)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l1[i **][** j **][** k **]**	4	se(v) | ae(v)
}		

7.3.5.3 Residual data syntax

residual(mb_type) {	**Category**	**Descriptor**
if(entropy_coding_mode = = 1)		
residual_4x4block = residual_4x4block_cabac( ) [Ed.Note: dirty trick, explain]	5 | 6	
else		
residual_4x4block = residual_4x4block_cavlc( )	5 | 6	
if(mb_type = = Intra_16x16)		
residual_4x4block(intra16x16DC, 16) [Ed.Note: define intra16x16DC]	5	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
for(i4x4 = 0; i4x4 < num_sub_blocks( ); i4x4++) /* each 4x4 sub-block of block */		
if(coded_block_pattern & (1 << i8x8))		
if(mb_type = = Intra_16x16)		
residual_4x4block(intra16x16AC, 16) [Ed.Note: define intra16x16AC]	5	
else		
residual_4x4block(luma, 16) [Ed.Note: define luma]	5 | 6	
if(coded_block_pattern & 0x30) /* chroma DC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
residual_4x4block(chromaDC, 4)) [Ed.Note: define chromaDC]	5 | 6	
if(coded_block_pattern & 0x20) /* chroma AC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
residual_4x4block(chromaAC, 16)	5 | 6	
}		

7.3.5.3.1 Residual 4x4 block CAVLC syntax

residual_4x4block_cavlc(block_mode, MaxNumCoeff) {	**Category**	**Descriptor**
coeff_token	5 | 6	ce(v)
NumCoeffs = total_coeff(coeff_token) - trailing_ones(coeff_token)		
if(trailing_ones(coeff_token) > 0)		
for(i = trailing_ones(coeff_token)-1; i >= 0; i--)		
trailing_ones_sign[i **]**	5 | 6	u(1)
if(total_coeff(coeff_token) > 0) {		
for(i = NumCoeffs-1; i >= 0; i--)		
coeff_level[i **]**	5 | 6	ce(v)
if(total_coeff(coeff_token) < MaxNumCoeff) {		
total_zeros	5 | 6	ce(v)
i = total_coeff(coeff_token) - 1		
ZerosLeft = total_zeros		
if(i>0 && ZerosLeft > 0) {		
do {		
run_before[i **]**	5 | 6	ce(v)
ZerosLeft -= run_before[i]		
i--		
} while(ZerosLeft > 0 && i > 0)		
run_before[i] = ZerosLeft		
}		
}		
}		
}		

7.3.5.3.2 Residual 4x4 block CABAC syntax

residual_4x4block_cabac(BlockType, MaxNumCoeff) {	**Category**	**Descriptor**
coded_block_flag	5 | 6	ae(v)
if(coded_block_flag) {		
for(i = 0; i < MaxNumCoeff - 1; i++) {		
significant_coeff_flag[i **]**	5 | 6	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i **]**	5 | 6	ae(v)
if(last_significant_coeff_flag[i])		
MaxNumCoeff = i + 1		
}		
}		
coeff_absolute_value_minus1[MaxNumCoeff-1 **]**	5 | 6	ae(v)
coeff_sign[MaxNumCoeff-1 **]**	5 | 6	ae(v)
for(i = MaxNumCoeff-2; i >= 0; i--)		
if(significant_coeff_flag[i]) {		
coeff_absolute_value_minus1[i **]**	5 | 6	ae(v)
coeff_sign[i **]**	5 | 6	ae(v)
}		
}		
}		

7.4 Semantics

7.4.1 NAL unit semantics

NOTE - The Video Coding Layer (VCL) is specified to efficiently represent the content of the video data. The Network Abstraction Layer (NAL) is specified to format that data and provide header information in a manner appropriate for conveyance by the transport layers 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 bitstream is identical except that each NAL unit can be preceded by a start code prefix in a bitstream-oriented transport layer.

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit and shall be conveyed by external means. Framing of NAL units is necessary to enable inference of NumBytesInNALunit. Such framing into a byte stream format is specified in Annex B and other methods for framing may be specified outside of this Recommendation | International Standard.

NOTE - Any sequence of bits can be formatted into a sequence of bytes in a manner specified as an RBSP by suffixing the data with rbsp_trailing_bits( ), and any RBSP can be encapsulated in a NAL unit in a manner that prevents emulation of byte stream start code prefixes within the NAL unit.

forbidden_bit shall be zero.

NOTE - The forbidden_bit may be used by external specifications to signal potentially corrupt NAL units.

nal_storage_idc equal to 0 signals that the content of the NAL unit belongs either to a picture that is not stored in the reference picture buffer, SEI data or Filler data. [Ed.Note: Not sure that the extension by SEI data or Filler Data here is right, but the point is these other types should be mentioned here.] nal_storage_idc shall not be 0 for sequence parameter set or picture parameter set NAL units. If nal_storage_idc is 0 for one slice or data partition NAL unit of a particular picture, it shall be 0 for all slice and data partition NAL units of the picture. nal_storage_idc greater than 0 signals that the content of the NAL unit belongs to a decoded picture that is stored in the reference picture buffer. [Ed.Note: resolve ambiguity between stored decoded and coded data.]

NOTE - In addition to signalling non-stored content, external specifications may use nal_storage_idc to indicate the relative transport priority of the NAL unit in a manner not specified in this Recommendation | International Standard. The value 0 should be used to signal the lowest transport priority and the priority should grow in ascending order of nal_storage_idc values.

nal_unit_type indicates the type of element contained in the NAL unit according to the types specified in Table 7-1.

Table 7-1 – NAL Unit Type Codes

Value of nal_unit_type	Content of NAL unit and RBSP syntax structure	Category
0x0 [Ed.Note: why hexadecimal ? make it decimal numbers ?]	Reserved for external use [Ed.Note: Why external ?]	
0x1	Coded slice slice_layer_no_partitioning_rbsp( )	4, 5, 6
0x2	Coded data partition A (DPA) dpa_layer_rbsp( )	4
0x3	Coded data partition B (DPB) dpb_layer_rbsp( )	5
0x4	Coded data partition C (DPC) dpc_layer_rbsp( )	6
0x5	Coded slice of an IDR picture slice_layer_no_partitioning_rbsp( )	4, 5
0x6	Supplemental Enhancement Information (SEI) sei_rbsp( )	7
0x7	Sequence Parameter Set (SPS) seq_parameter_set_rbsp( )	0
0x8	Picture Parameter Set (PPS) pic_parameter_set_rbsp( )	1
0x9	Picture Delimiter (PD) pic_delimiter_rbsp( )	8
0xA	Filler Data (FD) filler_data_rbsp( )	9
0xB – 0x17	Reserved	
0x18 – 0x1F	For external use [Ed.Note: Why external ?]	

[Ed.note: This paragraph should be revised and possibly moved?] An instantaneous decoder refresh picture (IDR picture) implies that all pictures in the multi-picture buffer are marked as "unused". Moreover, the maximum long-term index is reset to zero. An IDR picture contains only I or SI slices, and IDR slice type shall be used for all slices of an IDR picture.

Deleted: except the current picture

rbsp[i **]** a raw byte sequence payload is specified as an ordered sequence of bytes that contains an SODB. The RBSP contains the SODB in the following form:

a) If the SODB is null, the RBSP is also null.

b) Otherwise, the RBSP shall contain the SODB in the following form:

  1) The first byte of the RBSP shall contain 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) The final byte of the RBSP shall have the following form:

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

    ii) The next bit of the final RBSP byte shall consist of a single rbsp_stop_bit having the value one ('1'), and

    iii) Any remaining bits of the final RBSP byte, if any, shall consist of one or more rbsp_alignment_bit having the value zero ('0').

The last byte of a RBSP shall never have the value zero (0x00), because it contains the rbsp_stop_bit.

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_bit, which is the last (least significant, right-most) bit having the value one (‘1‘), and discarding any following (less significant, farther to the right) bits that follow it, which have the value zero (‘0‘).

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.

emulation_prevention_byte is a byte equal to 0x03.

Within the NAL unit, an **emulation_prevention_byte** shall be present after an rbsp[i] byte having the value zero (0x00) if and only if a next byte of RBSP data rbsp[i+1] follows that has one of the following four values:

– zero (0x00)
– one (0x01)
– two (0x02)
– three (0x03)

NOTE - Example encoder procedure. The encapsulation of an SODB within an RBSP and the encapsulation of an RBSP within a NAL unit is specified to prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit and enable identification of the end of the SODB within the NAL unit.

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

and a byte having the value three (0x03) is inserted to replace these bit patterns with the patterns

'00000000 00000011 000000xx'

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 process can allow any RBSP data to be sent in NAL unit while ensuring that no long SCP and no byte-aligned short SCP is emulated in 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 is called an active sequence parameter set when an IDR NAL unit refers to it. The parameters of an active sequence parameter set shall be replaced only when an IDR NAL unit refers to a different sequence parameter set.

A picture parameter set includes the parameters that remain unchanged within a coded picture. Every picture parameter set shall refer to the active sequence parameter set. A decoded picture parameter set is an active picture parameter set when the first slice NAL unit or first DPA NAL unit of a coded picture refers to it. The picture parameters of an active picture parameter set shall be replaced only when the first slice NAL unit or DPA NAL unit of a picture refers to a different picture parameter set.

NOTE - The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. The sequence parameter set, the picture parameter set, and the slice header contain all the parameters needed to decode the slice data. It is recommended to convey sequence and picture parameter sets out-of-band using a reliable transport mechanism. However, if an application requires a self-contained bitstream, in-band parameter set information units may be used. In error-prone transmission environments, in-band sequence and picture parameter set information units should be protected in a way that assures their successful reception. Synchronization between in-band and out-of-band transmission of the sequence and picture parameter set information is outside of the scope of this Recommendation | International Standard.

profile_idc and **level_idc** indicate profile and level as specified in Annex A.

seq_parameter_set_id identifies the sequence parameter set to be referred. The value of seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

log2_max_frame_num_minus4 specifies the MAX_FN used in frame number related arithmetic as follows:

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

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

pic_order_cnt_type equal to 0 or 1 indicates the method to code picture order count (see subclause 8.3.2). pic_order_cnt_type values greater than 1 are reserved.

log2_max_pic_order_cnt_minus4 is used to specify the MaxPicOrderCnt used in picture order count related arithmetic as follows:

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

Deleted: MAX_PIC_ORDER_CNT

Deleted: MAX_PIC_ORDER_CNT

The size of the pic_order_cnt parameter in the slice header is log2_max_pic_order_cnt_minus4 + 4 bits. The value of log2_max_pic_order_cnt_minus4 shall be in the range of 0 to 12, inclusive.

offset_for_non_stored_pic indicates an expected picture order count difference of a non-stored picture compared to the expected picture order count of the most recently decoded stored picture having a frame_num one less (modulo MAX_FN) than that of the non-stored picture.

Deleted: a stored picture having the same frame_num as the non-stored picture.

num_stored_frames_in_pic_order_cnt_cycle signals the number of frame numbers in a picture order count cycle. A picture order count cycle is a repetitive pattern of picture order count differences, each of which corresponds to a frame_num increment of one.

offset_for_stored_frame indicates an expected difference of picture order count corresponding to a frame_num increment of one. The notation offset_for_stored_frame[i] indicates the offset_for_stored_frame corresponding to index 'i' in the picture order count cycle.

Deleted: N

Deleted: n

Deleted: n

Deleted: offset_for_stored_frame_in_pic_order_cnt_cycle is the sum of all values of offset_for_stored_frame.

num_of_ref_frames specifies the total number of short- and long-term pictures in the reference picture buffer.

required_frame_num_update_behaviour_flag equal to 1 specifies a specific decoder behaviour in case of missing frame numbers. [Ed. Note: need to be more specific here].

Deleted: If **required_frame_num_update_behaviour**

Deleted: numbers

Deleted: .

pic_width_in_mbs_minus1 and **pic_height_in_mbs_minus1** specify the size of the luma picture array internal to the decoder in units of macroblocks. The picture width and height in units of macroblocks is computed by adding 1 to the decoded values of each of these parameters. The maximum macroblock address, MaxMbAddress, shall be calculated according to Equation 7-3.

$$\text{MaxMbAddress} = (\text{pic\_width\_in\_mbs\_minus1} + 1) \times (\text{pic\_height\_in\_mbs\_minus1} + 1) - 1 \qquad (7\text{-}3)$$

Deleted: MAX_MB_ADDRESS

Deleted: MAX_MB_ADDRESS

Deleted:

The NumBitsInMbAddress indicates the number of bits used to code a macroblock address with a fixed-length unsigned integer. It is calculated as follows.

Deleted: NUM_BITS_IN_MB_ADDRESS

If a picture is a field-structured picture or if macroblock-adaptive frame/field coding is not in use for the picture, then

$$\text{NumBitsInMbAddress} = \text{Ceil}(\text{Log2}(\text{MaxMbAddress} + 1)) \qquad (7\text{-}4)$$

Deleted: NUM_BITS_IN_MB_ADDRESS

Deleted: MAX_MB_ADDRESS

otherwise, NumBitsInMbAddress shall be

$$\text{NumBitsInMbAddress} = \text{Ceil}(\text{Log2}(\text{MaxMbAddress} + 1) - 1) \qquad (7\text{-}4a)$$

Deleted: NUM_BITS_IN_MB_ADDRESS

Deleted: MAX_MB_ADDRESS

Deleted: NUM_BITS_IN_MB_ADDRESS

filter_parameters_flag specifies whether a set of parameters controlling the characteristics of the deblocking filter is indicated in the slice header.

Deleted: **send_filter_parameters_flag**

constrained_intra_pred_flag equal to zero indicates normal intra prediction, whereas one indicates constrained intra prediction, where no intra prediction is done from macroblocks coded with mb_pred_type != Intra. [Ed.Note: SIntra ? MN: I think the text is OK as it is. SIntra_4x4 only ever occurs in SI slices, in macroblock locations where inter coding was used in the corresponding SP slice. Hence SIntra_4x4 is treated as inter, and so should not be used for prediction if inter is not used for prediction.].

mb_frame_field_adaptive_flag equal to zero indicates no switching between frame and field decoding mode at the macroblock layer, whereas one indicates the use of switching between frame and field decoding mode at the macroblock layer.

vui_seq_parameters_flag equal to zero specifies that default parameter values for the vui sequence parameters shall be applied. [Ed.Note: mention Annex E]

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id the picture parameter set identifier to be used for reference. The value of pic_parameter_set_id shall be in the range of 0 to 63, inclusive.

seq_parameter_set_id refers to the sequence parameter set that is used with this picture parameter set.

entropy_coding_mode equal to zero indicates VLC and CAVLC (see subclause 9.1), whereas value one indicates CABAC (see subclause 9.2). If CABAC is indicated, the ae(v) entropy coding is used for the assigned syntax elements.

motion_resolution equal to zero indicates 1/4 luma sample accurate motion resolution, and equal to one indicates 1/8 luma sample accurate motion resolution.

adaptive_block_size_transform_flag equal to zero indicates usage of 4x4 transforms for the luma residual, and equal one indicates usage of transforms of size 4x4, 4x8, 8x4, and 8x8 for the luma residual. Clause 12 specifies modifications as indicated in clause 7 that are related to syntax, semantics, and decoding process.

num_slice_groups_minus1 the number of slice groups is equal to num_slice_groups_minus1 + 1. If num_slice_groups_minus1 is zero, all slices of the picture belong to the same slice group.

NOTE – One slice group means that no flexible macroblock ordering is applied. If num_slice_groups_minus1 is greater than zero, flexible macroblock ordering is in use.

mb_allocation_map_type the macroblock allocation map type is present only if num_slice_groups_minus1 is greater than 0. This parameter indicates how the macroblock allocation map is coded. The value of this syntax element shall be in the range of 0 to 6, inclusive.

mb_allocation_map_type 0 is used to indicate interleaved slices.

mb_allocation_map_type 1 is used to indicate a dispersed macroblock allocation.

mb_allocation_map_type 2 is used to explicitly assign a slice group to each macroblock location in raster scan order.

mb_allocation_map_type 3 is used to indicate one or more “foreground” slice groups and a “leftover” slice group.

mb_allocation_map_types 4, 5 and 6 are used to indicate changing slice groups. num_slice_groups_minus1 shall be 1, when mb_allocation_map_type is 4, 5 or 6.

If mb_allocation_map_type is 0, the **run_length** syntax element follows for each slice group. It indicates the number of consecutive macroblocks that are assigned to the slice group in raster scan order. After the macroblocks of the last slice group have been assigned, the process begins again from the first slice group. The process ends when all the macroblocks of a picture have been assigned.

NOTE - Example: To signal macroblock row interleaving in a QCIF picture (where all even numbered macroblocks are in slice group 0, and all odd numbered macroblocks are in slice group 1), the number of slice groups is two and run_length is 11 for both slice groups.

If mb_allocation_map_type is 1, the macroblock allocation map is formed using the following formula, where n is the number of columns in the picture (in macroblocks) and p is the number of slice groups to be coded. Specifically, macroblock position x is assigned to slice group S according to Equation 7-5.

$$S(x) = ((x \% n) + ((\mathrm{floor}(x / n) * p) / 2)) \% p \tag{7-5}$$

If mb_allocation_map_type is 2, slice_group_id identifies a slice group of a macroblock. The size of the slice_group_id parameter shall be the minumum number of bits required for a fixed-length code to uniquely identify the number of slice groups. [Ed. Note: I’m sure this can be specified more clearly; I think this is the intention, but I do not see it specified anywhere.]

NOTE - slice_group_id is repeated as often as there are macroblocks in the picture.

If mb_allocation_map_type is 3, **top_left_mb** and **bottom_right_mb** are specified for each slice_group_id except for the last one. The top_left_mb specifies the top-left corner of a rectangle and bottom_right_mb specifies the bottom-right corner. top_left_mb and bottom_right_mb are indicated as macroblock addresses. The foreground slice group contains the macroblocks that are within the indicated rectangle and that do not belong to any slice group having a smaller slice_group_id. The last slice_group_id is dedicated for the leftover slice group, which contains the macroblocks that are not covered by the foreground slice groups. The leftover slice group shall not be empty. The size of the top_left_mb and bottom_right_mb parameters shall be NumBitsInMbAddress.

Deleted: NUM_BITS_IN_MB_ADDRESS

If mb_allocation_map_type is 4, 5 or 6, mb_allocation_map_type and **slice_group_change_direction** indicate the refined macroblock allocation map type according to Table 7-2. The macroblock allocation map is generated each time the decoder starts decoding of a new picture as described in subclause 8.3.4.

Table 7-2– Refined macroblock allocation map type

mb_allocation_map_type	slice_group_change_direction	refined macroblock allocation map type
4	0	Box-out clockwise
4	1	Box-out counter-clockwise

5	0	Raster scan
5	1	Reverse raster scan
6	0	Wipe right
6	1	Wipe left

slice_group_change_rate_minus1 is the minimum non-zero number of macroblocks by which the size a slice group can change from one picture to the next. The SliceGroupChangeRate variable is specified as follows:

$$\text{SliceGroupChangeRate} = \text{slice\_group\_change\_rate\_minus1} + 1 \quad (7\text{-}6)$$

The decoded value of slice_group_change_rate_minus1 shall be in the range of 0 to MaxMbAddress – 1, inclusive.

Deleted: SLICE_GROUP_CHANGE_RATE

Deleted: SLICE_GROUP_CHANGE_RATE

Deleted: MAX_MB_ADDRESS

num_ref_idx_l0_active_minus1 specifies the number of reference pictures minus 1 in the reference list 0 that are used to decode the picture.

num_ref_idx_l1_active_minus1 specifies the number of reference pictures minus 1 in the reference list 1 that are used to decode the picture.

weighted_pred_flag equal to zero indicates that weighted prediction is not applied to P and SP slices. weighted_pred_flag equal to one indicates that weighted prediction is applied to P and SP slices.

weighted_bipred_explicit_flag equal to zero indicates that explicit weighted prediction is not applied to B slices. weighted_bipred_explicit_flag equal to one indicates that explicit weighted prediction is applied to B slices. weighted_bipred_explicit_flag shall be zero if weighted_bipred_implicit_flag is one.

weighted_bipred_implicit_flag equal to zero indicates that implicit weighted prediction is not applied to B slices. weighted_bipred_implicit_flag equal to one indicates that implicit weighted prediction is applied to B slices.weighted_bipred_implicit_flag shall be zero if weighted_bipred_explicit_flag is one.

slice_qp_minus26 specifies the value of the default QP_Y for the macroblocks in an I, SI, P, SP, or B slice as specified in Equation 7-7. The value of this syntax element shall be in the range of -26 to +25, inclusive.

slice_qp_s_minus26 specifies the value of the default QS_Y for the macroblocks in a SP or SI slice as specified in Equation 7-8. The value of this syntax element shall be in the range of -26 to +25, inclusive.

Deleted: Eqatition

redundant_slice_flag indicates the presence of the redundant_pic_cnt [Ed.Note: pic or slice ?] parameter in all slice headers referencing the picture parameter set.

vui_pic_parameters_flag equal to zero specifies that default parameter values for the vui picture parameters shall be applied. [Ed.Note: mention Annex E]

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode VCL data correctly but is helpful for decoding or presentation purposes.

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 SEI header and SEI payload. The type and size of the SEI payload are coded using an extensible syntax. The SEI payload size is indicated in bytes. SEI payload types are specified in Annex C.

The SEI payload may have an SEI payload header. For example, a payload header may indicate to which picture the particular data belongs. The payload header shall be specified for each payload type separately.

An SEI message is associated with the next slice or data partition RBSP in decoding order. [Ed.Note: isn’t it associated with a complete picture ? Will it remain active for the next pictures ?]

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

last_payload_type_byte identifies the payload type of the last entry in an SEI message.

last_payload_size_byte identifies the size of the last entry in an SEI message.

7.4.2.4 Picture delimiter RBSP semantics

The picture delimiter may be used to signal the boundary between pictures, i.e., if present, it shall be inserted before the first NAL unit of a picture in decoding order. There is no normative decoder process associated with the picture delimiter.

pic_type signals which slice coding types are used in the following picture in decoding order. Table 7-3shows the slice coding types that can occur in a picture with a given pic_type.

Table 7-3– Meaning of pic_type

pic_type	Allowed slice_type_idc
000	Intra
001	Intra, Pred
010	Intra, Pred, BiPred
011	SIntra
100	SIntra, SPred
101	Intra, SIntra
110	Intra, SIntra, Pred, SPred
111	Intra, SIntra, Pred, SPred, BiPred

If adaptive_block_size_transform_flag = = 1, only pic_type = = '000', pic_type = = '001', and pic_type = = '010' are allowed.

non_stored_content_flag equal to 1 indicates that the picture is not stored in the reference picture buffer.

7.4.2.5 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF.

7.4.2.6 Slice layer RBSP semantics

The slice layer RBSP consists of a slice header and slice data.

7.4.2.7 Data partition RBSP semantics

7.4.2.7.1 Data partition A RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains all symbols of Category 4.

NOTE - Category 4 consists of the header symbols of all coded macroblocks.

Deleted: MB

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.4.2.7.2 Data partition B RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Data partition B contains all symbols of Category 5.

NOTE - Category 5 consists of the intra coded block patterns and coefficients.

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.4.2.7.3 Data partition C RBSP semantics

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Data partition C contains all symbols of Category 6.

NOTE - Category 6 consists of the inter coded block patterns and coefficients.

slice_id Each slice of a picture is associated a unique slice identifier within the picture. The first coded slice of the picture shall have identifier 0 and the identifier shall be incremented by one per each coded slice.

7.4.2.8 RBSP trailing bits semantics

rbsp_stop_bit is a single bit having the value one ('1').

rbsp_alignment_bit is a single bit having the value zero ('0').

7.4.2.9 RBSP slice trailing bits semantics

rbsp_stop_bit has the same semantics as in subclause 7.3.2.8.

rbsp_alignment_bit has the same semantics as in subclause 7.3.2.8.

cabac_stuffing_bit is a single bit having the value one ('1'). When entropy_coding_mode is equal to 1, the number of bins resulting from decoding the contents of the slice layer NAL unit shall not exceed 32*NumBytesInNALunit. A number of inserted cabac_stuffing_bit guarantee this condition.

NOTE – The method for determining the number of inserted cabac_stuffing_bit at the encoder is as follows: First, at the beginning of the slice encoding, the index CodeLength pointing to the current position of the bit stream is stored in a variable CodeLengthStored, i.e., CodeLength ← CodeLengthStored. In addition, two counters EBins and EBinsx8 are set to zero. Then, each time a symbol is encoded, EBins is incremented by one, i.e., EBins ← EBins+1. In the renormalization procedure, each time a new byte of compressed data is written, the following procedure is done:

```
while( EBins > 7 ) {
    EBins ← EBins-8
    EBinsx8 ← EBinsx8+1
}
```

After terminating the arithmetic encoding process such that the last pending bits have been written to the bitstream, the number of written bits is determined by CodeLength ← 8*(CodeLength – CodeLengthStored). Given the number of encoded bins by EBins ← 8*EBinsx8+EBins and the number NumMbSlice of macroblocks per slice, the following procedure is done:

```
EBins ← 0.25*EBins
if( CodeLength  >=  4*NumMbSlice )
{
    if( EBins > CodeLength )
        for( i = 0; i < EBins – CodeLength – 1; i++ )
            putbits( '1' ) /* writes cabac_stuffing_bit */
}
```

7.4.3 Slice header semantics

pic_parameter_set_id indicates the picture parameter set in use.

frame_num labels the frame. frame_num shall be incremented by 1 for each coded picture in decoding order, in modulo MAX_FN operation, relative to the frame_num of the previous stored frame in decoding order. An IDR picture shall have frame_num equal to 0. Both fields of a frame shall have the same frame number. The decoding order of primary coded pictures shall be non-decreasing in frame number order. For non-stored pictures with the same frame number, the decoding order shall be non-decreasing in picture order count. The frame_num serves as a unique ID for each frame stored in the reference picture buffer. Therefore, a frame cannot be kept in the buffer after its frame_num has been used by another frame unless it has been assigned a long-term frame index as specified below. No frame_num of a frame to be added to the reference picture buffer shall equal to any other among the short-term frames in the reference picture buffer. A decoder which encounters a frame number on a current frame having a value equal to the frame number of some other short-term stored frame in the reference picture buffer should treat this condition as an error.

pic_structure identifies the picture structure according to Table 7-4.

Table 7-4 – Meaning of pic_structure

Value of pic_structure	Meaning
0	Progressive frame picture
1	Top field picture
2	Bottom field picture
3	Interlaced frame picture, whose top field precedes its bottom field in time.
4	Interlaced frame picture, whose bottom field precedes its top field in time.

Note that when top field and bottom field pictures are coded for a frame, the one that is decoded first is the one that occurs first in time.

first_mb_in_slice specifies the macroblock address of the first macroblock contained in this slice. The size of the first_mb_in_slice parameter is NumBitsInMbAddress. The value of first_mb_in_slice shall be in the range of 0 to MaxMbAddress, inclusive.

If macroblock-adaptive frame/field decoding is in use, first_mb_in_slice contains a macroblock pair address rather than a macroblock address and the number of macroblocks included in a slice shall be an even number.

slice_type_idc indicates the coding type of the slice according to Table 7-5.

Deleted: NUM_BITS_IN_MB_ADDRESS

Deleted: MAX_MB_ADDRESS

Formatted: Font: Not Bold

Formatted: Font: Not Bold, Check spelling and grammar

Formatted: Font: Not Bold

Formatted: Font: Not Bold, Check spelling and grammar

Deleted: Table 7-5

Table 7-5 – Meaning of slice_type_idc

Value of slice_type_idc	Prediction type of slice (slice type)
0	Pred (P slice)
1	BiPred (B slice)
2	Intra (I slice)
3	SPred (SP slice)
4	SIntra (SI slice)

Table 7-6specifies, which macroblock prediction types are allowed when a slice type is decoded.

Table 7-6 – Allowed macroblock prediction types for slice_type_idc

Prediction type of slice (slice type)	Allowed macroblock prediction type
Pred (P slice)	Intra, Pred
BiPred (B slice)	Intra, Pred, BiPred
Intra (I slice)	Intra
SPred (SP slice)	SPred, Intra
SIntra (SI slice)	SIntra, Intra

If adaptive_block_size_transform_flag = = 1, the use of SI slices and SP slices is not allowed.

pic_order_cnt carries the picture order count coded in modulo MaxPicOrderCnt arithmetic. An IDR picture shall have pic_order_cnt equal to 0. The size of the pic_order_cnt parameter is log2_max_pic_order_cnt_minus4 + 4 bits.

Deleted: MAX_PIC_ORDER_CNT

delta_pic_order_cnt signals the picture order count difference compared to the expected picture order count as described in subclause 8.3.2.

Deleted: offset

redundant_pic_cnt is 0 for coded slices and data partitions belonging to the primary representation of the picture contents. The redundant_pic_cnt is greater than 0 for coded slices and data partitions that contain redundant coded representation of the picture contents. There should be no noticeable difference between the co-located areas of the decoded primary representation of the picture and any decoded redundant slices. Decoded slices having the same redundant_pic_cnt shall not overlap. Decoded slices having a redundant_pic_cnt greater than 0 may not cover the entire picture area.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to determine the prediction values direct prediction. If direct_spatial_mv_pred_flag is set to 0, then direct mode motion parameters are calculated from the picture order count as described in subclause 10.3.3.2. Otherwise, if this flag is set to 1, then direct mode motion parameters are calculated using the spatial motion vector prediction technique as described in subclause 10.3.3.1.

num_ref_idx_active_override_flag equal to zero indicates that the 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 one indicates that the num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 specified in the referred picture parameter set are overridden by the following values in the slice header.

num_ref_idx_l0_active_minus1 specifies the number of reference pictures minus 1 in the reference picture list 0 that are used to decode the slice.

Deleted: picture

num_ref_idx_l1_active_minus1 specifies the number of reference pictures minus 1 in the reference picture list 1 that are used to decode the slice.

Deleted: picture

slice_qp_delta specifies the value of the QP_Y for the macroblocks in the slice unless modified by the value of delta_qp in the macroblock layer. From this value, the initial QP_Y parameter for the slice is computed as:

Deleted: MB

$$QP_Y = 26 + \text{slice\_qp\_minus26} + \text{slice\_qp\_delta} \qquad (7\text{-}7)$$

The initial decoded QP_Y parameter shall be in the range of 0 to 51, inclusive. The value of QP_Y is initialised to the above result and this value is used for the decoding of each macroblock in the slice unless updated by a delta_qp sent in the macroblock layer.

sp_for_switch_flag indicates the decoding process to be used to decode the SP slice.

slice_qp_s_delta is signalled for SP and SI slices. The QS_Y parameter for the slice is computed as:

$$QS_Y = 26 + \text{slice\_qp\_s\_minus26} + \text{slice\_qp\_s\_delta} \quad (7\text{-}8)$$

The value of QS_Y shall be in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in the slice.

disable_deblocking_filter_flag equal to zero specifes that the deblocking filter shall be applied to the edges controlled by the macroblocks within the current slice. If disable_deblocking_filter_flag is 1, Filter_Offset_A and Filter_Offset_B shall both be inferred to be equal to -51. If not present in the slice header the value of this field shall be inferred to be zero.

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. The decoded value of this parameter shall be in the range from -6 to +6, inclusive. From this value, the offset that shall be applied when addressing these tables is computed as:

$$\text{Filter\_Offset\_A} = \text{slice\_alpha\_c0\_offset\_div2} << 1$$

If not present in the slice header the value of this field shall be inferred to be zero unless disable_deblocking_filter_flag is 1.

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. The decoded value of this parameter shall be in the range from -6 to +6, inclusive. From this value, the offset that is applied when addressing the β Table of the deblocking filter is computed as:

$$\text{Filter\_Offset\_B} = \text{slice\_beta\_offset\_div2} << 1$$

If not present in the slice header the value of this field shall be inferred to be zero unless disable_deblocking_filter_flag is 1.

slice_group_change_cycle × SliceGroupChangeRate indicates the number of macroblocks in slice group 0. The size of the slice_group_change_cycle field is Ceil(Log2(Ceil(MAX_MB_LOCATION ÷ SliceGroupChangeRate))). The maximum value of slice_group_change_cycle is Ceil(MAX_MB_LOCATION ÷ SliceGroupChangeRate).

Deleted: ALPHA

Deleted: BETA

Deleted: BETA

Deleted: SLICE_GROUP_CHANGE_RATE

Deleted: SLICE_GROUP_CHANGE_RATE

Deleted: SLICE_GROUP_CHANGE_RATE

7.4.3.1 Reference index reordering semantics

The syntax elements remapping_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_idx specify the change from the default reference index lists to the reference index lists used for decoding the slice.

ref_idx_reordering_flag_l0 indicates whether the syntax elements remapping_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_idx are present for specifying the reference index list 0.

ref_idx_reordering_flag_l1 has the same semantics as ref_idx_reordering_flag_l0 except that the reordering of the reference index list 1 is specified instead of the reference index list 0.

remapping_of_pic_nums_idc together with abs_diff_pic_num_minus1 and long_term_pic_idx indicates which of the reference pictures are re-mapped. The restrictions and the mapping to the code number are specified in Table 7-7. The number of signalling remapping_of_pic_nums_idc is limited to the num_ref_idx_l0_active_minus1 + 1.

Table 7-7 – remapping_of_pic_nums_idc operations for re-mapping of reference pictures

Value of remapping_of_pic_nums_idc	Re-mapping Specified
0	abs_diff_pic_num_minus1 is present and corresponds to a negative difference to add to a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a positive difference to add to a picture number prediction value
2	long_term_pic_idx is present and specifies the long-term index for a reference picture
3	End loop for re-mapping of reference picture set relative indexing default order

abs_diff_pic_num_minus1 plus 1 indicates the absolute difference between the picture number of the picture being remapped and the picture number prediction value.

long_term_pic_idx indicates the long-term picture index of the picture being remapped. In the case of frame-structured pictures, it shall be less than max_long_term_pic_idx_plus1; while in the case of field-structured pictures, it shall be less than 2 x max_long_term_pic_idx_plus1.

7.4.3.2 Reference picture buffer management semantics

The syntax elements ref_pic_buffering_mode, memory_management_control_operation, difference_of_pic_nums_minus1, long_term_pic_idx, and max_long_term_pic_idx_plus1 specify the buffering of a stored decoded picture into the reference picture buffer. Further, the reference picture buffer can be modified by marking pictures as unused, by assigning long-term pictures indices and by resetting of the reference picture buffer. The syntax elements ref_pic_buffering_mode, memory_management_control_operation, difference_of_pic_nums_minus1, long_term_pic_idx, and max_long_term_pic_idx_plus1 shall be identical for all coded slices of a coded picture.

ref_pic_buffering_mode specifies the buffering mode of the currently decoded picture and specifies how the reference picture buffer is modified after the current picture is decoded. The values for ref_pic_buffering_mode are specified in Table 7-8.

Table 7-8 – Interpretation of ref_pic_buffering_mode

Value of ref_pic_buffering_mode	Reference picture buffering mode specified
0	Sliding window buffering mode; A simple buffering mode providing a first-in first-out mechanism for pictures that are not assigned a long-term index
1	Adaptive buffering mode; A more flexible buffering mode than sliding window buffering mode providing syntax elements to specify marking pictures as unused, to assign long-term pictures indices, and to reset the reference picture buffer.

memory_management_control_operation specifies a control operation to be applied to manage the reference picture buffer. The memory_management_control_operation parameter is followed by data necessary for the operation specified by the value of memory_management_control_operation. memory_management_control_operation commands do not affect the buffer contents or the decoding process for the decoding of the current frame. They specify the necessary buffer status for the decoding of subsequent coded pictures. The values and control operations associated with memory_management_control_operation are specified in Table 7-9.

If memory_management_control_operation is Reset, all frames and fields in the reference picture buffer (but not the current picture unless specified separately) shall be marked “unused” (including both short-term and long-term pictures). Moreover, the maximum long-term picture index shall be reset to zero.

The frame height and width shall not change within the bitstream except within a picture containing a Reset memory_management_control_operation command.

A “stored picture” shall not contain any memory_management_control_operation command which marks that (entire) picture as “unused”. If the current picture is non-stored picture, the value of the memory_management_control_operation shall not contain any of the following types of memory_management_control_operation commands:

a) A Reset memory_management_control_operation command,

b) Any memory_management_control_operation command which marks any other picture (other than the current picture) as “unused” that has not also been marked as “unused” in the RPS layer of a prior stored picture, or

c) Any memory_management_control_operation command which assigns a long-term index to a picture that has not also been assigned the same long-term index in the RPS layer of a prior stored picture.

Table 7-9 – Memory management control operation (memory_management_control_operation) values

Value of memory_management_control_operation	Memory Management Control Operation	Associated Data Fields Following

0	End memory_management_control_operation Loop	None (end of RPS layer)
1	Mark a short-term picture as "Unused"	difference_of_pic_nums_minus1
2	Mark a long-term picture as "Unused"	long_term_pic_idx
3	Assign a long-term index to a picture	difference_of_pic_nums and long_term_pic_idx
4	Specify the maximum long-lerm picture index	max_long_term_pic_idx_plus1
5	Reset	None

Deleted: prame

difference_of_pic_nums_minus1 is used to assign a long-term index to a picture or to mark a short-term picture as "unused".

long_term_pic_idx is used to assign a long-term index to a picture or to mark a long-term picture as "unused".

max_long_term_pic_idx_plus1 indicates the maximum index allowed for long-term reference frames (until receipt of another value of max_long_term_pic_idx_plus1). The decoder shall initially infer that max_long_term_pic_idx_plus1 is 0 until some other value has been received.

7.4.3.3 Prediction weight table semantics

luma_log_weight_denom is the binary logarithm of the denominator for all luma weighting factors.

chroma_log_weight_denom is the binary logarithm of the denominator for all chroma weighting factors.

luma_weight_flag_l0 indicates whether weighting factors are present for the luma component of the list 0 prediction.

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for reference index i in list 0 prediction.

luma_offset_l0[i **]** is the additive offset applied to the luma prediction value for reference index i in list 0 prediction.

If luma_weight_flag_l0 is equal to zero, luma_weight_l0[i] shall be interpreted as equal to $2^{\text{luma\_log\_weight\_denom}}$ and luma_offset_l0[i] shall be interpreted as equal to zero.

chroma_weight_flag_l0 indicates whether weighting factors are present for the Cb and Cr components of the list 0 prediction.

chroma_weight_l0[i **][** 0 **]** is the weighting factor applied to the Cb prediction values for reference index i in list 0 prediction.

chroma_offset_l0[i **][** 0 **]** is the additive offset applied to the Cb prediction values for reference index i in list 0 prediction.

chroma_weight_l0[i **][** 1 **]** is the weighting factor applied to the Cr prediction values for reference index i in list 0 prediction.

chroma_offset_l0[i **][** 1 **]** is the additive offset applied to the Cr prediction values for reference index i in list 0 prediction.

If chroma_weight_flag_l0 is equal to zero**,** chroma_weight_l0[i] shall be interpreted as equal to $2^{\text{chroma\_log\_weight\_denom}}$ and chroma_offset_l0[i] shall be interpreted as equal to zero.

luma_weight_flag_l1 indicates whether weighting factors are present for the luma component of the list 1 prediction.

luma_weight_l1[i **]** is the weighting factor applied to the luma prediction values for reference index i in list 1 prediction.

luma_offset_l1[i **]** is the additive offset applied to the luma prediction value for reference index i in list 1 prediction.

If luma_weight_flag_l1 is equal to zero, luma_weight_l1[i] shall be interpreted as equal to $2^{\text{luma\_log\_weight\_denom}}$ and luma_offset_l1[i] shall be interpreted as equal to zero.

chroma_weight_flag_l1 indicates whether weighting factors are present for the chroma component of the list 1 prediction.

chroma_weight_l1[i **][** 0 **]** is the weighting factor applied to the Cb prediction values for reference index i in list 1 prediction.

chroma_offset_l1[i **][** 0 **]** is the additive offset applied to the Cb prediction values for reference index i in list 1 prediction.

chroma_weight_l1[i **][** 1 **]** is the weighting factor applied to the Cr prediction values for reference index i in list 1 prediction.

chroma_offset_l1[i **][** 1 **]** is the additive offset applied to the Cr prediction values for reference index i in list 1 prediction.

If chroma_weight_flag_l1 is equal to zero, chroma_weight_l1[i][j] shall be interpreted as equal to $2^{\text{chroma\_log\_weight\_denom}}$ and chroma_offset_l1[i][j] shall be interpreted as equal to zero.

num_custom_bipred_weights is the number of custom weight and offset combinations sent for bi-predictive weighting.

irp_l0 is the index of the reference picture in list 0 for which a custom weight and offset combination is indicated for custom bi-predictive weighting.

irp_l1 is the index of the reference picture in list 1 for which a custom weight and offset combination is indicated for custom bi-predictive weighting.

luma_weight_bipred_l0[irp_l0 **][** irp_l1 **]** is the weighting factor applied to the luma prediction value for reference index irp_l0 in list 0 when used with reference index irp_l1 in list 1 for bi-prediction.

luma_weight_bipred_l1[irp_l0 **][** irp_l1 **]** is the weighting factor applied to the luma prediction value for reference index irp_l1 in list 1 when used with reference index irp_l0 in list 0 for bi-prediction.

luma_offset_bipred[irp_l0 **][** irp_l1 **]** is the additive offset applied to the luma prediction values when reference index irp_l0 in list 0 is used with reference index irp_l1 in list 1 for bi-prediction.

chroma_weight_flag_bipred[irp_l0 **][** irp_l1 **]** indicates whether custom weight and offset combinations are sent for Cb and Cr prediction when reference index irp_l0 in list 0 is used with reference index irp_l1 in list 1 for bi-prediction.

chroma_weight_bipred_l0[irp_l0 **][** irp_l1 **][** 0 **]** is the weighting factor applied to the Cb prediction value for reference index irp_l0 in list 0 when used with reference index irp_l1 in list 1 for bi-prediction.

chroma_weight_bipred_l1[irp_l0 **][** irp_l1 **][** 0 **]** is the weighting factor applied to the Cb prediction value for reference index irp_l1 in list 1 when used with reference index irp_l0 in list 0 for bi-prediction.

chroma_offset_bipred[irp_l0 **][** irp_l1 **][** 0 **]** is the additive offset applied to the Cb prediction values when reference index irp_l0 in list 0 is used with reference index irp_l1 in list 1 for bi-prediction.

chroma_weight_bipred_l0[irp_l0 **][** irp_l1 **][** 1 **]** is the weighting factor applied to the Cr prediction value for reference index irp_l0 in list 0 when used with reference index irp_l1 in list 1 for bi-prediction.

chroma_weight_bipred_l1[irp_l0 **][** irp_l1 **][** 1 **]** is the weighting factor applied to the Cr prediction value for reference index irp_l1 in list 1 when used with reference index irp_l0 in list 0 for bi-prediction.

chroma_offset_bipred[irp_l0 **][** irp_l1 **][** 1 **]** is the additive offset applied to the Cr prediction values when reference index irp_l0 in list 0 is used with reference index irp_l1 in list 1 for bi-prediction.

If chroma_weight_flag_bipred[irp_l0][irp_l1] is zero, chroma_weight_bipred_l0[irp_l0][irp_l1][j] and chroma_weight_bipred[irp_l0][irp_l1][j] shall be interpreted as equal to $2^{\text{chroma\_log\_weight\_denom}}$ and chroma_offset_bipred[irp_l0][irp_l1][j] shall be interpreted as equal to zero.

For any combination of irp_l0 and irp_l1 that is not sent, the following values shall be inferred:

- luma_weight_bipred_l0[irp_l0][irp_l1] = luma_weight_l0[irp_l0],
- luma_weight_bipred_l1[irp_l0][irp_l1] = luma_weight_l1[irp_l1],
- luma_offset_bipred[irp_l0][irp_l1] =
  (luma_offset_l0[irp_l0] + luma_offset_l1[irp_l1] + 1) >> 1
- chroma_weight_bipred_l0[irp_l0][irp_l1][j] = chroma_weight_l0[irp_l0][j],
- chroma_weight_bipred_l1[irp_l0][irp_l1][j] = chroma_weight_l1[irp_l1][j],
- chroma_offset_bipred[irp_l0][irp_l1][j] =
  (chroma_offset_l0[irp_l0][j] + chroma_offset_l1[irp_l1][j] + 1) >> 1

7.4.4 Slice data semantics

mb_skip_run indicates a number of consecutive macroblocks decoded as MbSkip macroblock type in P slices or Direct_16x16 macroblock type and no additional transform coefficients in B slices when entropy_coding_mode = = 0.

For mb_frame_field_adaptive_flag == 1, refer to the macroblock scanning order in Figure 6-4 (subclause 6.3).

Deleted: MB

mb_skip_flag indicates that the macroblock is decoded as MbSkip macroblock type in P slices or Direct_16x16 macroblock type and no additional transform coefficients in B slices when entropy_coding_mode == 1.

For the MbSkip macroblock type no further information about the macroblock is decoded. The motion vector for a MbSkip macroblock type shall be obtained as described in subclause 8.4.1.3. If pic_structure indicates a frame, then the decoded frame with ref_idx_l0 == 0, which either was decoded from a frame picture or is the union of two decoded field pictures shall be used as reference in motion compensation. If pic_structure indicates a field, then the decoded field of the same parity (top or bottom) with ref_idx_l0 == 0, which was either decoded from a field picture or is part of the most recently decoded frame picture shall be used as reference in motion compensation. For mb_frame_field_adaptive_flag == 1, a pair of macroblocks will be in frame mode only. [Ed.Note: I don't really understand the following two sentences] If one of a pair of macroblocks is skipped, it should be in the same frame/field coding mode as another macroblock of the same pair. In field coding, the skipped macroblock is decoded by copying the co-located macroblock from the most recently decoded I or P field of the same field parity.

mb_field_decoding_flag equal to zero indicates that the macroblock pair is decoded in frame decoding mode and equal to one indicates that the macroblock pair is decoded in field decoding mode. [Ed.Note: This should only be sent per macroblock pair. Moreover, MbSkip macroblock type should be permitted for each of the macroblocks separately.]

end_of_slice_flag equal to 0 indicates that another macroblock is following, whereas equal to 1 indicates the end of the slice and that no further macroblock follows. end_of_slice_flag is only present if entropy_coding_mode == 1 for each macroblock except for the last macroblock of the picture.

If the current picture is a frame-structured picture and mb_adaptive_frame_field_flag is 1, the number of coded macroblocks in the slice shall be an even number.

7.4.5 Macroblock layer semantics

mb_type indicates the macroblock types. The semantics of mb_type depend on the slice type. In the following, for I slices, SI slices, P slices, SP slices, and B slices, tables and semantics are specified for the various macroblock types.

The macroblock types for I slices are specified in Table 7-10. If adaptive_block_size_transform_flag == 1, the macroblock types for I slices are specified in Table 12-1.

Table 7-10 – Macroblock types for I slices

Value of mb_type	Name of mb_type in I slices	mb_partition_pred_mode(, 1)	num_sub_blocks( )	num_mb_intra_partition( )
0	Intra_4x4	Intra	4	16
1	Intra_16x16_0_0_0	Intra	4	na
2	Intra_16x16_1_0_0	Intra	4	na
3	Intra_16x16_2_0_0	Intra	4	na
4	Intra_16x16_3_0_0	Intra	4	na
5	Intra_16x16_0_1_0	Intra	4	na
6	Intra_16x16_1_1_0	Intra	4	na
7	Intra_16x16_2_1_0	Intra	4	na
8	Intra_16x16_3_1_0	Intra	4	na
9	Intra_16x16_0_2_0	Intra	4	na
10	Intra_16x16_1_2_0	Intra	4	na
11	Intra_16x16_2_2_0	Intra	4	na
12	Intra_16x16_3_2_0	Intra	4	na
13	Intra_16x16_0_0_1	Intra	4	na
14	Intra_16x16_1_0_1	Intra	4	na

15	Intra_16x16_2_0_1	Intra	4	na
16	Intra_16x16_3_0_1	Intra	4	na
17	Intra_16x16_0_1_1	Intra	4	na
18	Intra_16x16_1_1_1	Intra	4	na
19	Intra_16x16_2_1_1	Intra	4	na
20	Intra_16x16_3_1_1	Intra	4	na
21	Intra_16x16_0_2_1	Intra	4	na
22	Intra_16x16_1_2_1	Intra	4	na
23	Intra_16x16_2_2_1	Intra	4	na
24	Intra_16x16_3_2_1	Intra	4	na

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

- Intra_4x4: the macroblock is coded as Intra prediction type.
- Intra_16x16_x_y_z: x: Imode, y: nc, z: AC these modes refer to 16x16 Intra prediction type. Imode numbers from 6 and upwards represent 16x16 intra coding. [Ed. Note: this sentence appears to be wrong.]

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Tab after: 1,27 cm + Indent at: 1,27 cm

The macroblock types for SI slices are specified in Table 7-11 and Table 7-10. The mb_type value 0 is specified in Table 7-11 and the mb_type values 1 to 25 are specified in Table 7-10 by adding 1 to the value of mb_type in Table 7-10.

NOTE - If adaptive_block_size_transform_flag = = 1, the use of SI slices is not allowed.

Table 7-11 – Macroblock type with value 0 for SI slices

Value of mb_type	Name of mb_type SI slices	mb_partition_pred_mode(, 1)	num_sub_blocks( )
0	SIntra_4x4	SIntra	4

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

- SIntra_4x4: the macroblock is coded as SIntra prediction type.

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Tab after: 1,27 cm + Indent at: 1,27 cm

The macroblock types for P and SP slices are specified in Table 7-12 and Table 7-10. The mb_type values 0 to 4 are specified in Table 7-12 and the mb_type values 5 to 29 are specified in Table 7-10 by adding 5 to the value of mb_type in Table 7-10.

NOTE - If adaptive_block_size_transform_flag = = 1, the use of SP slices is not allowed.

Table 7-12 – Macroblock type values 0 to 4 for P and SP slices

Value of mb_type	Name of mb_type	num_mb_partition ( )	mb_partition_pred_mode(, 1)	mb_partition_pred_mode(, 2)	num_sub_blocks( )
0	Pred_L0_16x16	1	Pred_L0		4
1	Pred_L0_L0_16x8	2	Pred_L0	Pred_L0	4
2	Pred_L0_L0_8x16	2	Pred_L0	Pred_L0	4
3	Pred_8x8	4	Na	na	4
4	Pred_8x8ref0	4	Na	na	na

Formatted Table

... [8]

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

- Pred_L0_16x16, Pred_L0_L0_16x8, Pred_L0_L0_8x16, and Pred_8x8: the macroblock is predicted from a past picture with luma block sizes 16x16, 16x8, 8x16, and 8x8, respectively, and the associated chroma blocks. For the macroblock types NxM=16x16, 16x8, and 8x16, a motion vector is provided for each NxM luma block and the associated chroma blocks. If N=M=8, for each 8x8 sub macroblock an additional syntax element is decoded which indicates in which type the corresponding sub macroblock is decoded (see subclause 7.4.5.2). Depending on N,M and the sub macroblock types modes there may be 1 to 16 sets of motion data elements for a macroblock.

- Pred_8x8ref0: same as Pred_8x8 but ref_idx_l0 is not sent and set to 0 for all sub macroblocks.

The macroblock types for B slices are specified in Table 7-13 and Table 7-10. The mb_type values 0 to 22 are specified in Table 7-13 and the mb_type values 23 to 47 are specified in Table 7-10 by adding 23 to the value of mb_type in Table 7-10.

Table 7-13 – Macroblock type values 0 to 22 for B slices

Value of mb_type	Macroblock type mb_type name	num_mb_partition( )	mb_partition_pred_mode(, 1)	mb_partition_pred_mode(, 2)	num_sub_blocks( )
0	Direct_16x16	1	Direct		4
1	Pred_L0_16x16	1	Pred_L0		4
2	BiPred_L1_16x16	1	Pred_L1		4
3	BiPred_Bi_16x16	1	BiPred		4
4	Pred_L0_L0_16x8	2	Pred_L0	Pred_L0	4
5	Pred_L0_L0_8x16	2	Pred_L0	Pred_L0	4
6	BiPred_L1_L1_16x8	2	Pred_L1	Pred_L1	4
7	BiPred_L1_L1_8x16	2	Pred_L1	Pred_L1	4
8	BiPred_L0_L1_16x8	2	Pred_L0	Pred_L1	4
9	BiPred_L0_L1_8x16	2	Pred_L0	Pred_L1	4
10	BiPred_L1_L0_16x8	2	Pred_L1	Pred_L0	4
11	BiPred_L1_L0_8x16	2	Pred_L1	Pred_L0	4
12	BiPred_L0_Bi_16x8	2	Pred_L0	BiPred	4
13	BiPred_L0_Bi_8x16	2	Pred_L0	BiPred	4
14	BiPred_L1_Bi_16x8	2	Pred_L1	BiPred	4
15	BiPred_L1_Bi_8x16	2	Pred_L1	BiPred	4
16	BiPred_Bi_L0_16x8	2	BiPred	Pred_L0	4
17	BiPred_Bi_L0_8x16	2	BiPred	Pred_L0	4
18	BiPred_Bi_L1_16x8	2	BiPred	Pred_L1	4
19	BiPred_Bi_L1_8x16	2	BiPred	Pred_L1	4
20	BiPred_Bi_Bi_16x8	2	BiPred	BiPred	4
21	BiPred_Bi_Bi_8x16	2	BiPred	BiPred	4
22	BiPred_8x8	4	na		na

The following semantics are assigned to the macroblock types in Table 7-13:

- Direct_16x16 type: no motion vector data is transmitted.

- BiPred_x_y_NxM, x,y=L0,L1,Bi: each NxM block of a macroblock is predicted by using distinct motion vectors, reference pictures, and prediction directions. As indicated in Table 11-11???, three different macroblock types that differ in their prediction methods exist for the 16x16 mode. For the 16x8 and 8x16 macroblock types, nine different combinations of the prediction methods are possible. If a macroblock is coded in 8x8 mode, an additional codeword for each 8x8 sub-partition indicates the decomposition of the 8x8 block as well as the chosen prediction direction (see Table 11-2???).

- BiPred_8x8: the macroblock is partitioned into sub macroblocks. The coding of each sub macroblock is specified using sub_mb_type.

coded_block_pattern specifies which of the 8x8 blocks - luma and chroma - contain transform coefficients. An 8x8 block contains four 4x4 blocks. An indication that an 8x8 block contains coefficients means that one or more of the four 4x4 blocks within the 8x8 block contains coefficients'. The four least significant bits of coded_block_pattern contain information on which of four 8x8 luma blocks in the macroblock contains nonzero coefficients. These four bits are denoted as coded_block_patternY. The ordering of 8x8 blocks is indicated in Figure 6-6. A bit equal to zero in position n of coded_block_pattern (binary representation) indicates that the corresponding 8x8 block has no coefficients and a bit equal to 1 indicates that the 8x8 block has one or more non-zero coefficients. For chroma, 3 possibilities are specified in Table 7-14.

Deleted: contains information

Deleted: Figure 6-5

Table 7-14 – Specification of nc values

Value of nc	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.

Deleted: e

Deleted: e

The value of coded_block_pattern for a macroblock is given by coded_block_pattern = coded_block_patternY + 16 x nc

The coded_block_pattern is indicated with a different codeword for macroblocks in P, SP, and B slices compared to macroblocks in I and SI slices.

If adaptive_block_size_transform_flag = = 1, the semantics for coded_block_pattern are specified in subclause 12.3.

delta_qp the value of QP_Y can be changed in the macroblock layer by the parameter delta_qp. The delta_qp parameter is present only for non-skipped macroblocks, as specified by:

- If coded_block_pattern indicates that there are nonzero transform coefficients in the macroblock or
- If the macroblock is 16x16 based intra coded

Furthermore, when macroblock-level adaptive frame/field coding is in use, delta_qp is present only for the first non-skipped macroblock of each macroblock pair.

The decoded value of delta_qp shall be in the range from -26 to +25, inclusive. This specifies the value of QP_Y in the range [0..51], as given by

$$QP_Y = (QP_Y + delta\_qp + 52) \% 52 \tag{7-9}$$

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction mode is signalled using the mb_type syntax element. Depending on the mb_type syntax element, intra prediction information is signalled using intra_pred_mode and intra_chroma_pred_mode or inter prediction information is signalled using the syntax elements ref_idx_l0, ref_idx_l1, mvd_l0, and mvd_l1.

If adaptive_block_size_transform_flag = = 1, modifications to the macroblock prediction semantics are specified in subclause 12.2.1.1.

The return value of num_mb_intra_partition( ) is always equal to 16 if adaptive_block_size_transform_flag = = 0.

NOTE - If adaptive_block_size_transform_flag = = 1 the semantics regarding the function num_mb_intra_partition( ) are specified in subclause 12.2.1.1.

intra_pred_mode indicates the intra prediction mode.

intra_chroma_pred_mode indicates the type of spatial prediction used for chroma whenever any part of the luma macroblock is intra coded. This is shown in Table 7-15.

Table 7-15 – Relationship between intra_chroma_pred_mode and spatial prediction modes

Value of intra_chroma_pred_mode	Prediction Mode
0	Vertical
1	Horizontal
2	DC
3	Plane

Deleted: DC

Deleted: Vertical

ref_idx_l0, when present, indicates the reference picture to be used for prediction.

If pic_structure indicates that the current picture is a frame picture, then the reference picture is a previous frame in list 0 that was either indicated as a single frame picture or a frame that was indicated as two field pictures and has been reconstructed as a frame. Thus for frames the following Table gives the reference frame:

Code_num	Reference frame
0	The first frame in list 0
1	The second frame in list 0
2	The third frame in list 0
..	..

If pic_structure indicates that the current picture is a field picture, then the reference picture is a previous field in list 0 that was either coded as part of a frame-structured picture or coded as a field-structured picture. Thus for fields the following Table gives the reference field:

Code_num	Reference field
0	The first field in list 0
1	The second field in list 0
2	The third field in list 0
..	..

If num_ref_idx_l0_active_minus1 is equal to 0, ref_idx_l0 is not present. If num_ref_idx_l0_active_minus1 is equal to 1, only a single encoded bit is used to represent ref_idx_l0. If num_ref_idx_l0_active_minus1 is greater than 1, the value of ref_idx_l0 is represented by a decoded index.

ref_idx_l1 has the same semantics as ref_idx_l0, except that it is applied to the reference index list 1.

mvd_l0 indicates the difference between a vector component to be used and its prediction. If so indicated by mb_type, vector data for 1-16 blocks are transmitted. For the coded motion vector, a prediction is formed for the horizontal and vertical components of the motion vector.

Deleted: ¶

Deleted: if

Deleted: every block

Deleted:

Deleted: mvd_l0 signals the difference between the vector component to be used and this prediction. Motion vectors are allowed to point to samples outside the reference picture. If a sample outside the reference picture is referred to in the prediction process, the nearest sample belonging to the picture (an edge or corner sample) shall be used. All fractional sample positions shall be interpolated as described in subclause 8.3.2. If a sample referred in the interpolation process (necessarily integer accuracy) is outside of the reference picture it shall be replaced by the nearest sample belonging to the picture (an edge or corner sample). Reconstructed motion vectors shall be clipped to ±19 integer samples outside of the picture. [Ed. Note: Is that an encoder or decoder clipping process? Does it affect the values of the prediction of subsequent MVs?]¶

mvd_l1 has the same semantics as mvd_l0, except that it is applied to the reference index list 1.

7.4.5.2 Sub macroblock prediction semantics

sub_mb_type indicates the sub macroblock types.

If adaptive_block_size_transform_flag = = 1, modifications to the sub macroblock prediction semantics are specified in subclause 12.2.1.2.

The return value of num_sub_mb_intra_partition( ) is always equal to 4 if adaptive_block_size_transform_flag = = 0.

NOTE - If adaptive_block_size_transform_flag = = 1 the semantics regarding the function num_sub_mb_intra_partition( ) are specified in subclause 12.2.1.1.

The sub macroblock types for P macroblocks are specified in Table 7-16.

Table 7-16 – Sub macroblock types in P macroblocks

Value of sub_mb_type	Name of sub_mb_type	num_sub_mb_ partition( )	sub_mb_pred_ mode( )	num_sub_mb_intra _partition( )	num_sub_blocks ( )
0	Pred_L0_8x8	1	Pred_L0	na	4
1	Pred_L0_8x4	2	Pred_L0	na	4
2	Pred_L0_4x8	2	Pred_L0	na	4
3	Pred_L0_4x4	4	Pred_L0	4	4
4	Intra_8x8	na	Intra		

The following semantics are assigned to the sub macroblock types in Table 7-16:

- Pred_L0_ XxY, X,Y=4,8 the corresponding partition of the sub macroblock is predicted from a past picture with luma block size 8x4, 4x8, and 4x4, respectively, and the associated chroma blocks. A motion vector is transmitted for each NxM=8x4, 4x8, and 4x4 block. Depending on N and M, up to 4 motion vectors may be decoded for a sub macroblock, and thus up to 16 motion vectors maybe dedoced for a macroblock.
- Intra_8x8 the 8x8 sub-partition is coded in intra mode. Intra_8x8 *shall not* be present in SPred slices.

The sub macroblock types for B macroblocks are specified in Table 7-17.

Table 7-17 – Sub macroblock types in B macroblocks

Value of sub_mb_type	Name of sub_mb_type	num_sub_ mb_partition( )	sub_mb_pred_mode( )	num_sub_mb_intra _partition( )	num_sub_blocks( )
0	Direct_8x8	1	Direct	na	4
1	Pred_L0_8x8	1	Pred_L0	na	4
2	BiPred_L1_8x8	1	Pred_L1	na	4
3	BiPred_Bi_8x8	1	BiPred	na	4
4	Pred_L0_8x4	2	Pred_L0	na	4
5	Pred_L0_4x8	2	Pred_L0	na	4
6	BiPred_L1_8x4	2	Pred_L1	na	4
7	BiPred_L1_4x8	2	Pred_L1	na	4
8	BiPred_Bi_8x4	2	BiPred	na	4
9	BiPred_Bi_4x8	2	BiPred	na	4
10	Pred_L0_4x4	4	Pred_L0	na	4
11	BiPred_L1_4x4	4	Pred_L1	na	4
12	BiPred_Bi_4x4	4	BiPred	na	4
13	Intra_8x8	1	Intra	4	4

The following semantics are assigned to the sub macroblock types in Table 7-17:

- Pred_L0_XxY, X,Y=4,8, have the same semantics as in Table 7-16.
- BiPred_Z_X_Y, Z=L1,Bi, X,Y=4,8 [Ed.Note: I need input here !!!]
- Intra_8x8 has the same semantics as in Table 7-16.

7.4.5.3 Residual data semantics

[Ed.Note: I need input here !!!]

If adaptive_block_size_transform_flag = = 1, modifications to the residual data semantics are specified in subclause 12.2.1.3.

The return value of num_sub_blocks( ) is always equal to 4 if adaptive_block_size_transform_flag = = 0.

NOTE - If adaptive_block_size_transform_flag = = 1 the semantics regarding the function num_sub_blocks( ) are specified in subclause 12.2.1.3.

7.4.5.3.1 Residual 4x4 block CAVLC semantics

[Ed.Note: I need input here !!!]

7.4.5.3.2 Residual 4x4 block CABAC semantics

coded_block_flag indicates 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] indicates 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 position i has a non-zero value.

last_significant_coeff_flag[i] indicates for the scanning position i whether there are non-zero transform coefficients for subsequent scanning positions i+1 to max_numcoeff-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_absolute_value_minus_1 is the absolute value of the transform coefficient minus 1.

coeff_sign is the sign of the transform coefficient. A coeff_sign equal to 0 indicates a positive transform coefficient; a coeff_sign equal to 1 indicates a negative transform coefficient.

8 Decoding process

8.1 Ordering of decoding process

[Ed.note: High-level decoding steps have to be added.]

A macroblock or sub-partition is decoded in the following order.

1. Parsing of syntax elements using VLC/CAVLC (see subclause 9.1) or CABAC (see subclause 9.2)
2. Motion compensation (see subclause 8.4) or Intra prediction (see subclause 8.5)
3. Transform coefficient decoding (see subclause 8.6)
4. Deblocking Filter (see subclause 8.7)

8.2 NAL unit decoding

8.2.1 NAL unit delivery and decoding order

This subclause presents the requirements for the NAL unit deliver and decoding order.

Decoders conforming to this Recommendation | International Standard shall be capable of receiving NAL units in decoding order.

Systems conveying NAL unit streams conforming to this Recommendation | International Standard shall either

– Present NAL unit streams to the decoder in decoding order, or
– Provide a means to indicate the NAL unit decoding order to the decoder in the case of enhanced-capability decoders which may be capable of receiving or processing some NAL units in an out-of-order fashion. No such enhanced capability is specified or required herein for decoders conforming to this Recommendation | International Standard.

The decoding order of a sequence parameter set shall precede the decoding order of other NAL units that refer to that sequence parameter set.

The decoding order of a picture parameter set shall precede the decoding order of other NAL units that refer to that picture parameter set.

A coded picture is called a primary coded picture if the redundant_slice_flag is 0 or if its slice headers contain redundant_pic_cnt equal to 0.

The decoding order of coded slices and data partitions of a primary coded picture shall be contiguous relative to the decoding order of coded slices and data partitions of other primary coded pictures.

The decoding order of coded slices and data partitions of a primary or redundant coded picture shall precede the decoding order of coded slices and data partitions of any other coded picture that uses the primary coded picture as a reference for inter-picture prediction.

The decoding order of any coded slices or data partitions of a primary coded picture shall precede the decoding order of any redundant slices or data partitions containing coded data for the same macroblock locations represented in the slice or data partition for the primary decoded picture.

The decoding order of any redundant coded slice or data partition shall precede the decoding order of the coded slices and data partitions of any other coded picture that uses the primary coded picture corresponding to the coded slice or data partition of the redundant coded slice or data partition as a reference for inter-picture prediction.

The decoding order of slices and data partitions of primary coded pictures shall be non-decreasing in frame number order. The decoding order of the slices and data partitions of non-stored primary coded pictures shall be subsequent to the decoding order of the the slices and data partitions of the stored picture with the same frame number. If multiple primary coded pictures share the same frame number, the decoding order of the slices and data partitions of the non-stored pictures shall be in ascending order of redundant_pic_cnt.

Depending on the profile in use, arbitrary slice ordering may or may not be allowed. If arbitrary slice ordering is allowed, the slices and data partitions of a primary coded picture may follow any decoding order relative to each other. If arbitrary slice ordering is not allowed, the decoding order of slices and data partitions of a primary coded picture shall be increasing in the raster scan order of the first macroblock of each slice, and the decoding order of data partition A of a coded slice shall precede the decoding order of data partition B of the same coded slice, and the decoding of data partition B of a coded slice shall precede the decoding order of data partition C of the same coded slice, and data partitions A, B, and C of a coded slice shall be contiguous in decoding order relative to the decoding order of any data partitions or non-partitioned slice data NAL units of other coded slices.

The decoding order of SEI NAL units shall precede the decoding order of the slices and the slices and data partitions of the corresponding slice or data partition or coded picture or sequence of pictures to which the SEI NAL unit corresponds, and shall be subsequent to the decoding order of any SEI NAL units, slices, and data partitions of pictures that preced the corresponding coded picture in decoding order.

The decoding order of a picture delimiter, if present, shall precede the decoding order of the SEI NAL units, slices and data partitions of the corresponding coded picture, and shall be subsequent to the decoding order of any SEI NAL units, slices, and data partitions of pictures that precede the corresponding coded picture in decoding order.

8.2.2 Parameter set decoding

The decoder maintains 16 sequence parameter set locations. For each of the 16 possible values of seq_parameter_set_id, the most recent decoded sequence parameter set is copied into the location referenced by seq_parameter_set_id immediately before the decoding of the next IDR picture.

The decoder maintains 64 picture parameter set locations. For each of the 64 possible values of pic_parameter_set_id, the most recent picture parameter set is copied into the location referenced by pic_parameter_set_id immediately before the decoding of a slice or DPA belonging to the next coded picture.

8.3 Slice decoding

8.3.1 Detection of coded picture boundaries

Decoding of a new picture is started from the slice to be decoded, herein called the current slice, if the slice is not a redundant slice and if one of the following conditions is true:

1. The frame number of the current slice is different from the frame number of the previously decoded slice.
2. The frame number of the current slice is the same as frame number of the previously decoded slice, the nal_storage_idc of the previously decoded slice is zero, and the nal_storage_idc of the current slice is non-zero.
3. The frame number of the current slice is the same as frame number of the previously decoded slice, pic_order_cnt_type is 0, and pic_order_cnt is different from the pic_order_cnt in the previously decoded slice.
4. The frame number of the current slice is the same as the previously decoded slice, pic_order_cnt_type is 1 and delta_pic_order_cnt is different from the delta_pic_order_cnt in the previously decoded slice.

8.3.2 Picture order count

Each coded picture is associated with a picture order count, called PicOrderCnt, which indicates the output order of each picture relative to the last IDR picture in decoding order.

NOTE – Picture order counts are used to determine the intended output order of coded pictures, to determine default index orderings for reference pictures (see 8.3.6.3) and to represent temporal distances between pictures for motion vector prediction (see 10.3.2).

The value of called PicOrderCnt shall be a 32-bit signed integer. An IDR picture shall have PicOrderCnt equal to 0. The PicOrderCnt of each stored picture shall be stored as long as the picture stays in the reference picture buffer. When a picture coded as a frame is used as a field reference, the picture order count of the top and bottom fields shall be the same as the picture order count of the frame.

There are two modes for coding picture order counts. The first mode, mode 0, uses a fixed size field to code the low order bits of the picture order count of each picture. The second mode, mode 1, uses values in the sequence parameter set to predict the picture order count for each picture and encodes only the difference between the predicted value and the true picture order count in the slice header.

The decoder should treat a wraparound, underflow or overflow of pic_order_cnt, PicOrderCntOffset, FrameNumOffset and AbsFrameNum specified in subclauses 8.3.2.1 and 8.3.2.2 as an error.

8.3.2.1 Picture order count type 0

If pic_order_cnt_type is 0, the decoder shall maintain a picture order count offset, called PicOrderCntOffset, which is a 32-bit signed integer. The PicOrderCntOffset shall be zero for an IDR picture.

If pic_order_cnt_type is 0 and if the decoding of a new picture is started, the PicOrderCntOffset is updated and the pic_order_cnt of the picture is calculated. The pic_order_cnt of the previous picture in decoding order is herein called PreviousPicOrderCnt. If pic_order_cnt is smaller than PreviousPicOrderCnt and if (PreviousPicOrderCnt – pic_order_cnt) is greater than or equal to (MaxPicOrderCnt / 2), PicOrderCntOffset is calculated according to Equation 8-1.

$$\text{PicOrderCntOffset} = \text{PicOrderCntOffset} + \text{MaxPicOrderCnt} \quad (8\text{-}1)$$

If pic_order_cnt is greater than PreviousPicOrderCnt and if (pic_order_cnt – PreviousPicOrderCnt) is greater than or equal to (MaxPicOrderCnt / 2), PicOrderCntOffset is calculated according to Equation 8-2.

$$\text{PicOrderCntOffset} = \text{PicOrderCntOffset} - \text{MaxPicOrderCnt} \quad (8\text{-}2)$$

Otherwise, the value of PicOrderCntOffset is not changed.

If pic_order_cnt_type is 0 and if the decoding of a new picture is started, the PicOrderCnt of the picture is calculated according to Equation 8-3.

$$\text{PicOrderCnt} = \text{PicOrderCntOffset} + \text{pic\_order\_cnt} \quad (8\text{-}3)$$

8.3.2.2 Picture order count type 1

If pic_order_cnt_type is 1, the decoder shall maintain a FrameNumOffset that is a 32-bit unsigned integer. The FrameNumOffset of an IDR picture shall be set to zero. For an IDR picture the frame_num and the PicOrderCnt are always zero, and the calculations below are not carried out.

If pic_order_cnt_type is 1 and if the decoding of a new picture is started, the PicOrderCnt of the picture is calculated. In the following, AbsFrameNum and PicOrderCntCycleCnt are 32-bit unsigned integers, and PreviousFrameNum is the frame_num of the previous picture in decoding order. First, the FrameNumOffset is updated as follows. If frame_num is greater than or equal to the PreviousFrameNum, FrameNumOffset is unchanged. Otherwise, when frame_num is smaller than PreviousFrameNum, FrameNumOffset is calculated according to Equation 8-4.

$$\text{FrameNumOffset} = \text{FrameNumOffset} + \text{MAX\_FN} \quad (8\text{-}4)$$

Second, the AbsFrameNum (the frame number relative to the last IDR picture) is calculated according to Equation 8-5.

$$\text{AbsFrameNum} = \text{FrameNumOffset} + \text{frame\_num} \quad (8\text{-}5)$$

Deleted: , which is

Deleted: MAX_PIC_ORDER_CNT

Deleted: MAX_PIC_ORDER_CNT

Deleted: MAX_PIC_ORDER_CNT

Deleted: MAX_PIC_ORDER_CNT

Formatted: Normal

Deleted: If pic_order_cnt_type is 1, the decoder shall maintain a FrameNumOffset that is a 32-bit unsigned integer. The FrameNumOffset of an IDR picture shall be zero.
If pic_order_cnt_type is 1 and if the decoding of a new picture is started, the PicOrderCnt of the picture is calculated. In the following, AbsFrameNum and PicOrderCntCycleCount are 32-bit unsigned integers, and the frame_num of the previous picture in decoding order is called PreviousFrameNum. First, the FrameNumOffset is updated. If frame_num is greater than or equal to the PreviousFrameNum, FrameNumOffset is unchanged. Otherwise, when frame_num is smaller than PreviousFrameNum, FrameNumOffset is calculated according to Equation 8-4.

```
if( nal_storage_idc  = =  0  &&  AbsFrameNum > 0)                    (8-6)
   AbsFrameNum = AbsFrameNum – 1
```

Third, if AbsFrameNum > 0, the PicOrderCntCycleCnt and FrameNumInPicOrderCntCycle are calculated according to Equation 8-7.

```
if(AbsFrameNum > 0) {                                                 (8-7)
   PicOrderCntCycleCnt = AbsFrameNum / num_stored_frames_in_pic_order_cnt_cycle
   FrameNumInPicOrderCntCycle = (AbsFrameNum –1)%
                    num_stored_frames_in_pic_order_cnt_cycle
}
```

In the following, the ExpectedDeltaPerPicOrderCntCycle is the sum of offset_for_stored_frame values. Finally, the PicOrderCnt of the picture is computed using the algorithm expressed in Equation 8-8.

```
if( AbsFrameNum > 0 ){
   ExpectedPicOrderCnt = PicOrderCntCycleCnt × ExpectedDeltaPerPicOrderCntCycle
   for( i = 0; i <= FrameNumInPicOrderCntCycle; i++ )
         ExpectedPicOrderCnt = ExpectedPicOrderCnt + offset_for_stored_frame[ i ]
} else {
   ExpectedPicOrderCnt = 0
   if( nal_storage_idc  = =  0 )
   ExpectedPicOrderCnt = ExpectedPicOrderCnt + offset_for_non_stored_pic
   PicOrderCnt = ExpectedPicOrderCnt + delta_pic_order_cnt
}                                                                     (8-8)
```

[Ed. Note: something is wrong here …]

8.3.3 Decoder process for redundant slices

If the redundant_pic_cnt in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. If some of the samples in the decoded primary picture are incorrect and if the coded redundant slice can be correctly decoded, the decoder should replace the incorrect samples with the corresponding samples of the redundant decoded slice.

8.3.4 Specification of macroblock allocation map

If decoding of a new picture is started, if num_slice_groups_minus1 is greater than 0 and if mb_allocation_map_type is 4, 5 or 6, a macroblock allocation map is generated. Slice group 0 has a growth order specified in ubclauses 8.3.4.1-8.3.4.4. The number of macroblocks in slice group 0 is equal to slice_group_change_cycle × SliceGroupChangeRate. This number of macroblock locations in the specified growth order is allocated for slice group 0. The rest of the macroblocks of the picture are allocated for slice group 1.

8.3.4.1 Allocation order for box-out

Let H denote the number of coded macroblock rows of the picture and W denote the number of coded macroblocks columns of the picture. Macroblock locations are indicated with coordinates (x, y), where the top-left macroblock location of the picture has coordinates (0, 0) and the bottom-right macroblock location has coordinates (W – 1, H – 1). The allocation order is created using a AllocationDirection variable that indicates the next macroblock location relative to the current one. AllocationDirection can have four values: (-1, 0), (1, 0), (0, -1) and (0, 1). Furthermore, the left-most and right-most macroblock columns allocated in the allocation order and the top-most and bottom-most macroblock rows allocated in the allocation order are stored in the variables Left, Right, Top, and Bottom respectively. For the box-out clockwise macroblock allocation map type, the first macroblock location in the allocation order is (x, y) = (W/2, H/2) and the initial AllocationDirection is (-1, 0). For the counter-clockwise macroblock allocation map type, the first macroblock location in allocation order is (x, y) = ((W – 1)/2, (H – 1)/2) and the initial AllocationDirection is (0, 1). At the beginning, Left = Right = x, and Top = Bottom = y. A subsequent macroblock location (x, y) in allocation order is allocated by searching the first row from top to bottom in Table 8-1for the same value of AllocationDirection and where the given condition is true. Then, the x, y, AllocationDirection, Left, Right, Top and Bottom variables are updated according to the refined macroblock allocation map type. If Left >= 0, Right < W, Top >= 0 and Bottom < H, the next macroblock location (x, y) in allocation order is allocated as described above. Otherwise, all macroblock locations have been allocated.

Formatted: Equation, Indent: Left: 2,54 cm, Keep with next, Keep lines together

Formatted: Font:

Formatted: Font:

Deleted: ou

Deleted: is calculated

Deleted: 6

Deleted: ou

Deleted: . . . . (8-6)¶ Fourth, the FrameNumInPicOrderCntCycle is calculated according to Equation 8-7.¶ FrameNumInPicOrderCntCycle = AbsFrameNum % num_stored_frames_in_pic_order_cnt_cycle . . . . (8-7)¶

Deleted: EXPECTED_DELTA_PER_PIC_ORDER_CNT_CYCLE

Formatted: Tabs: 4 cm, Left

Deleted: ou

Deleted: . . .

Deleted: EXPECTED_DELTA_PER_PIC_ORDER_CNT_CYCLE¶

Deleted: i¶ PicOrderCnt = PicOrderCnt + delta_pic_order_cnt¶ if(nal_storage_idc = = 0) . . PicOrderCnt = PicOrderCnt + offset_for_non_stored_pic

Formatted: Not Superscript/ Subscript

Formatted: Normal, Indent: Left: 0 cm

Deleted: SLICE_GROUP_CHANGE_RATE

Table 8-1 – Allocation order for the box-out macroblock map allocation type

AllocationDirection	Condition	Box-out clockwise	Box-out counter-clockwise
(-1, 0)	x > Left	x = x – 1	x = x – 1
(-1, 0)	x = = 0	y = Top – 1 Top = Top – 1 AllocationDirection = (1, 0)	y = Bottom + 1 Bottom = Bottom + 1 AllocationDirection = (1, 0)
(-1, 0)	x = = Left	x = x – 1 Left = Left – 1 AllocationDirection = (0, -1)	x = x –1 Left = Left – 1 AllocationDirection = (0, 1)
(1, 0)	x < Right	x = x + 1	x = x + 1
(1, 0)	x = = W – 1	y = Bottom + 1 Bottom = Bottom + 1 AllocationDirection = (-1, 0)	y = Top – 1 Top = Top – 1 AllocationDirection = (-1, 0)
(1, 0)	x = = Right	x = x + 1 Right = Right + 1 AllocationDirection = (0, 1)	x = x + 1 Right = Right + 1 AllocationDirection = (0, -1)
(0, -1)	y > Top	y = y – 1	y = y – 1
(0, -1)	y = = 0	x = Right + 1 Right = Right + 1 AllocationDirection = (0, 1)	x = Left – 1 Left = Left – 1 AllocationDirection = (0, 1)
(0, -1)	y = = Top	y = y – 1 Top = Top –1 AllocationDirection = (1, 0)	y = y –1 Top = Top –1 AllocationDirection = (-1, 0)
(0, 1)	y < Bottom	y = y + 1	y = y + 1
(0, 1)	y = = H – 1	x = Left – 1 Left = Left – 1 AllocationDirection = (0, -1)	x = Right + 1 Right = Right + 1 AllocationDirection = (0, -1)
(0, 1)	y = = Bottom	y = y + 1 Bottom = Bottom + 1 AllocationDirection = (-1 ,0)	y = y + 1 Bottom = Bottom + 1 AllocationDirection = (1 ,0)

8.3.4.2 Allocation order for raster scan

For the raster scan slice group macroblock allocation map type, the first macroblock in the allocation order is the top-left one of the picture, and the allocation order follows the raster scan order.

For the reverse raster scan slice group macroblock allocation map type, the first macroblock in the allocation order is the bottom-right one of the picture, and the allocation order follows the reverse raster scan order.

8.3.4.3 Allocation order for wipe

For the wipe right slice group macroblock allocation map type, the first macroblock in the allocation order is the top-left one of the picture. The allocation order runs from top to bottom. The next macroblock after the bottom macroblock of a column is the top macroblock of the column to the right of the previous column.

For the wipe left slice group macroblock allocation map type, the first macroblock in the allocation order is the bottom-right one of the picture. The allocation order runs from bottom to top. The next macroblock after the top macroblock of a column is the bottom macroblock of the column to the left of the previous column.

8.3.4.4 Allocation order for macroblock level adaptive frame and field coding

Allocation order follows Figure 6-4 in subclause 6.3, instead of raster scan. [Ed.Note: more text need here !!!]

8.3.5 Data partitioning

When data partitioning is not used, coded slices start with a slice header and are followed by all the entropy coded symbols of Categories 4, 5, and 6 (see Category column in clause 7) of the macroblock data for the macroblocks of the slice.

When Data Partitioning is used, the macroblock data of a Slice is partitioned in three partitions. Partition A contains a partition A header and all entropy coded symbols of Category 4. Partition B contains a partition B header all symbols of Category 5. Partition C contains a partition C header and all symbols of Category 6. When data partitioning is used, each partition is conveyed in its own NAL unit, which may be empty if no symbols of the respective Category.

NOTE - Symbols of Category 5 are relevant to the decoding of intra coded texture information. Symbols of Category 6 are relevant to the decoding of residual data in Inter slices. Category 4 encompasses all other symbol types related to the decoding of macroblocks, and their information is often denoted as header information. The Partition A header contains all the symbols of the slice header, and additionally a slice number that is used to associate the partitions B and C with the partition A. The partition B and C headers contain only the slice number which allows their association with the partition A of the slice

8.3.6 Decoder process for management and use of the reference picture buffer

8.3.6.1 General

Intro to multi picture buffer. [Ed. Note: To be written properly]

Decoder stores reference pictures as indicated in the bitstream. These are used for prediction. The buffer is divided into two independent buffers: the short term buffer and the long term buffer. Pictures can only remain in the short term buffer for a finite duration, given by MAX_FN. Pictures can remain in the long term buffer until the next IDR picture. mmco commands are used to control the contents of these buffers.

The decoder employs indices when referencing a picture for motion compensation on the macroblock layer. Default indices are specified. These indices of pictures in the reference picture buffer are re-mapped onto newly specified indices according to the remapping_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_idx fields.

8.3.6.2 Picture Numbering

Picture numbers are used in the decoding process for management and use of the reference picture buffer for both changing the default indices and for controlling the contents of the reference picture buffer using memory management control operations.

In frame structured pictures, the picture number, PN, of a frame that has frame number FN, is given by PN = FN

In field structured pictures, the picture number, PN, of a field that has frame number FN, is given by PN = 2 x FN if the field is a top field, and is given by PN = 2 x FN + 1 if the field is a bottom field.

Long term picture numbers are also used in the decoding process for management and use of the reference picture buffer. Long term picture numbers are used for both changing the default indices of pictures in the long term buffer, and are used for transferring pictures from short term buffer to the long term buffer and for removing pictures from the long term buffer.

In frame structured pictures, the long term picture number, LTPN, of a frame that has long-term frame index LTFI, is given by LTPN = LTFI

In field structured pictures, the long term picture number, LTPN, of a field that has long-term frame index LTFI, is given by LTPN = 2 x LTFI if the field is a top field, and is given by LTPN = 2 x LTFI + 1 if the field is a bottom field.

In frame-structured pictures, the parameter MAX_PN is specified to equal MAX_FN, and in field-structured pictures , the parameter MAX_PN is specified to equal 2 x MAX_FN.

8.3.6.3 Default index orders

8.3.6.3.1 General

A reference index is a relative index into a list of reference indices to indicate which reference picture out of the reference picture buffer is used for motion compensation. When decoding a P or SP slice, there is one such list of reference indices, called the first reference index list. When decoding a B slice, there may be two reference indices used per block each pointing into a separate lists of reference indices which are called the first reference index list and second reference index list.

The first reference index list and the second reference index list have default mappings to the pictures numbers in the reference picture buffer as specified below.

8.3.6.3.2 Default index order for P and SP slices in frames

Deleted: -structured pictures

The default index order for list 0 prediction of P and SP slices in frames (i.e., frames which have not been given a long-term index) is for the short-term frames to precede the long-term frames in the reference indexing order. Within the set of short-term frames, the default order is for the frames to be ordered starting with the most recently-decoded reference frame and proceeding through to the reference frame in the short-term buffer that was decoded first (i.e., in decreasing order of frame_num in the absence of wrapping of the frame_num value). Within the set of long-term frames, the default

Deleted: -structured pictures

order is for the frames to be ordered starting with the frame with the smallest long-term index and proceeding up to the frame with largest long-term index.

A field that is stored in the short term or long term buffer for which the opposite parity field is not stored in the same buffer shall not be included in the default index order, shall not be remapped, and shall not be used for prediction in frames.

For example, assuming no wrap of the frame_num field, if the buffer contains three short-term frames with frame_num equal to 300, 302, and 303 and two long-term frames with long-term frame indices 0 and 3, the default index order is:

default relative index 0 refers to the short-term frame with frame_num 303,

default relative index 1 refers to the short-term frame with frame_num 302,

default relative index 2 refers to the short-term frame with frame_num 300,

default relative index 3 refers to the long-term frame with long-term frame index 0, and

default relative index 4 refers to the long-term frame with long-term frame index 3.

Deleted: , are not

Deleted: -structured pictures

Deleted: A field that is stored in the short term or long term buffer for which the opposite parity field is not stored in the same buffer, are not included in the default index order, shall not be remapped, and shall not be used for prediction in frame-structured pictures.¶

8.3.6.3.3 Default index order for P and SP slices in fields

Deleted: -structured pictures

In the case that the current picture is field-structured, each field of the stored reference pictures is identified as a separate reference picture with a unique index. Thus field structured pictures effectively have at least twice the number of pictures available for referencing. The calculated decoding order of reference fields alternates between reference pictures of the same and opposite parity, starting with fields that have the same parity as the current field-structured picture. Figure 8-6 shows the case of the first field in a field-structured picture pair, while Figure 8-7 shows the case of the second field. If one field of a reference frame was neither decoded nor stored, the decoding order calculation shall ignore the missing field and instead index the next available stored reference field of the respective parity in decoding order. When there are no more fields of the respective parity in the short term buffer, default indices shall be allocated to the not yet indexed fields of the other parity starting with the most recently decoded such field and progressing to the first decoded such field.

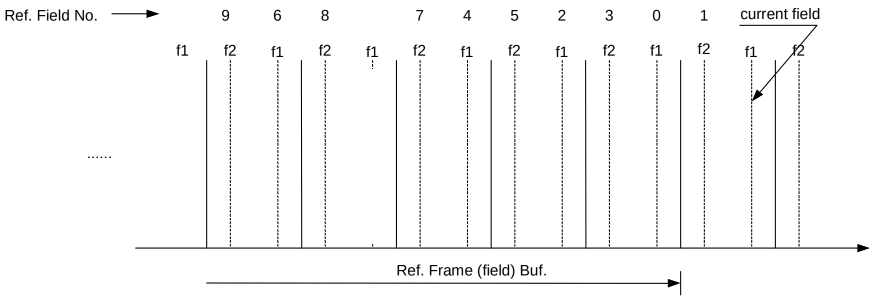


Figure 8-1 – Default reference field number assignment when the current picture is the first field coded in a frame

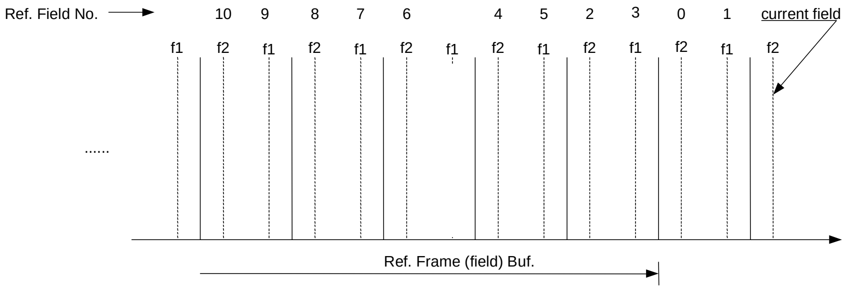


Figure 8-2 – Default reference field number assignment when the current picture is the second field coded in a frame

8.3.6.3.4 Default index order for B slices in frames

Deleted: -structured pictures

The organisation of short term pictures in the default order for B slices depends on output order, as given by PicOrderCnt.

The default index order for list 0 prediction of B slices in frame-structured pictures is for the short-term frames (i.e., frames which have not been given a long-term index) to precede the long-term frames in the reference indexing order. Within the set of short-term frames, the default order is for the frames to be ordered starting with the decoded reference frame with the largest value of PicOrderCnt less than the value of PicOrderCnt of the current frame and proceeding through to the reference frame in the short-term buffer that has the smallest value of PicOrderCnt; and then the frame with the largest value of PicOrderCnt greater than the value of PicOrderCnt of the current frame and proceeding through to the reference frame in the short-term buffer that has the smallest value of PicOrderCnt greater than the value of PicOrderCnt of the current frame. Within the set of long-term frames, the default order is for the frames to be ordered starting with the frame with the smallest long-term index and proceeding up to the frame with the largest long-term index.

The default index order for list 1 prediction of B slices in frame-structured pictures is for the short-term frames (i.e., frames which have not been given a long-term index) to precede the long-term frames in the reference indexing order. Within the set of short-term frames, the default order is for the frames to be ordered starting with the decoded reference frame with the largest value of PicOrderCnt and proceeding through to the reference frame in the short-term buffer that has the smallest value of PicOrderCnt. Within the set of long-term frames, the default order is for the frames to be ordered starting with the frame with the smallest long-term index and proceeding up to the frame with the largest long-term index.

The ordinary default order specified in the previous paragraph shall be used as the default index order for list 1 prediction unless there is more than one reference picture in the set and the ordinary default index order for list 1 prediction is the same as the default index order for list 0 prediction. In this exceptional case, the default index order for list 1 prediction shall be the ordinary default index order with the order of the first two pictures switched.

A field that is stored in the short term or long term buffer for which the opposite parity field is not stored in the same buffer shall not be included in the default index order, shall not be remapped, and shall not be used for prediction in frame-structured pictures.

Deleted: , are not

Deleted: -structured pictures

8.3.6.3.5 Default index order for B slices in fields

The default index order for list 0 and list 1 prediction of B slices in field-structured pictures is as for frame-structured pictures except that it is split between even indices for same-parity fields and odd indices for opposite-parity fields.

[Ed. This needs further work to allow for the case of dangling fields in the reference buffer.]

8.3.6.4 Changing the default index orders

8.3.6.4.1 General

The syntax elements remapping_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_idx fields allow indexing into the reference picture buffer to be temporarily altered from the default index order for the decoding of the current slice. A remapping_of_pic_nums_idc "end loop" indication indicates the end of a list of re-ordering commands.

The indices are assigned starting at zero and increasing by one for each remapping_of_pic_nums_idc field. Pictures that are not re-mapped to a specific order by remapping_of_pic_nums_idc, shall follow after any pictures having a re-mapped order in the indexing scheme, following the default order amongst these non-re-mapped pictures.

8.3.6.4.2 Changing the default index orders for short term pictures

abs_diff_pic_num_minus1 plus one indicates the absolute difference between the picture number of the picture being re-mapped and the prediction value. For the first occurence of the abs_diff_pic_num_minus1 field in ref_idx_reordering(), the prediction value is the picture number of the current picture. For subsequent occurences of the abs_diff_pic_num_minus1 field in ref_idx_reordering(), the prediction value is the picture number of the picture that was re-mapped most recently using abs_diff_pic_num_minus1.

The decoder shall determine the picture number of the picture being re-mapped, PNQ, in a manner mathematically equivalent to the following, where the picture number prediction is PNP.

```
if(remapping_of_pic_nums_idc = = 0)
{  /* a negative difference */
  if(PNP – abs_diff_pic_num_minus1 < 0)
    PNQ = PNP – abs_diff_pic_num_minus1 + MAX_PN;
  else
    PNQ = PNP – abs_diff_pic_num_minus1;
}
else
{  /* a positive difference */
  if(PNP + abs_diff_pic_num_minus1 > MAX_PN-1)
    PNQ  = PNP + abs_diff_pic_num_minus1 – MAX_PN;
  else
    PNQ  = PNP + abs_diff_pic_num_minus1;
}
```

The encoder shall control remapping_of_pic_nums_idc and abs_diff_pic_num_minus1 such that the decoded value of abs_diff_pic_num_minus1 shall not be greater than or equal to MAX_PN.

As an example implementation, the encoder may use the following process to determine values of abs_diff_pic_num_minus1 and remapping_of_pic_nums_idc to specify a re-mapped picture number in question, PN:

```
if(remapping_of_pic_nums_idc = = 0)
{  /* a negative difference */
  if(PNP – abs_diff_pic_num_minus1 < 0)
    PNQ = PNP – abs_diff_pic_num_minus1 + MAX_PN;
  else
    PNQ = PNP – abs_diff_pic_num_minus1;
}
```

```
else
{  /* a positive difference */
  if(PNP + abs_diff_pic_num_minus1 > MAX_PN-1)
    PNQ  = PNP + abs_diff_pic_num_minus1 – MAX_PN;
  else
    PNQ  = PNP + abs_diff_pic_num_minus1;
}
```

where abs( ) indicates an absolute value operation.

remapping_of_pic_nums_idc is then determined by the sign of MDELTA.

The prediction value used by any subsequent abs_diff_pic_num_minus1 re-mappings is not affected by long_term_pic_idx.

8.3.6.4.3 Changing the default index orders for long term pictures

The long_term_pic_idx field indicates the long term picture number of the long term picture being remapped.

8.3.6.5 Overview of decoder process for reference picture buffer management

The reference picture buffer consists of two independent parts: a short term buffer and a long term buffer. The decoder shall assume the initial size of the long term buffer to be 0, that is, it shall assume that max_long_term_pic_idx_plus1 is set to zero.

The long term buffer has capacity to store max_long_term_idx_plus1 frames. The usage of the long term buffer is constrained so that it has capacity for no more than max_long_term_idx_plus1 top fields and no more than max_long_term_idx_plus1 bottom fields.

The remainder of the reference picture buffer is allocated to the short term buffer, which has capacity to store (num_of_ref_frames - max_long_term_idx_plus1) frames. There is no further constraint on its capacity to store top and bottom fields. For example, the whole of the short term buffer could be used to store top fields.

nal_storage_idc indicates whether the current picture is stored in the reference picture buffer. When nal_storage_idc is equal to 0, the current picture is not stored in the reference picture buffer, otherwise it is stored in the reference picture buffer.

If the current picture is stored in the reference picture buffer, the process used for storing is indicated by ref_pic_buffering_mode, which indicates either "Sliding Window", a first-in, first-out mechanism, or "Adaptive Memory Control", a customised adaptive buffering operation specified with memory_management_control_operation commands.

In frame structured pictures, memory_management_control_operation commands apply to both fields of the frame.

In field structured pictures, memory_management_control_operation commands apply to individual fields.

8.3.6.6 Sliding window reference picture buffer management

The "Sliding Window" buffering mode operates as follows.

If there is sufficient “unused” capacity in the short term buffer to store the current picture, the current picture is stored in the short term buffer and no pictures are removed from the short term buffer.

Otherwise if the current picture is a field-structured picture, the short-term field with the largest default index, that is, field that has been in the short term buffer for the longest time, is marked “unused”, thus creating sufficient capacity to store the current picture. The current picture is then stored in the short term buffer.

Otherwise if the current picture is a frame-structured picture, default indices are calculated as done when decoding a field-structured picture, and the short-term field with the largest default index is marked “unused”. If there is still insufficient “unused” capacity in the short term buffer to store the current picture, the short-term field which now has the largest default index is also marked “unused”. The current picture is then stored in the short term buffer.

8.3.6.7 Adaptive Memory Control reference picture buffer management

8.3.6.7.1 General

The "Adaptive Memory Control" buffering mode allows specified pictures to be removed from either or both of the short and long term buffers, allows specified pictures to be moved from the short term buffer to the long term buffer, allows specified pictures to be removed from the long term buffer, allows the number of long term pictures to be modified, and allows the whole buffer to be reset, by use of memory_management_control_operation commands.

memory_management_control_operation commands are processed in the order they occur in the bitstream, and are processed after the whole picture has been decoded. When all commands have been processed, storage of the current picture is considered. When nal_storage_idc is equal to 0, the current picture is not stored in the reference picture buffer, otherwise it is stored in the short term buffer. memory_management_control_operation commands in the bitstream shall be such that when nal_storage_idc indicates that the current picture is to be stored, that there shall be sufficient "unused" capacity in the short term buffer to store the current picture.

8.3.6.7.2 Removal of short term pictures

If memory_management_control_operation equals 1 (Mark a Short-Term Picture as "Unused"), a specified short term picture in the short term buffer is marked as "unused", if that picture has not already been marked as "unused".

If the current decoded picture number is PNC, difference_of_pic_nums_minus1 is used in an operation mathematically equivalent to the following equations, to calculate, PNQ, the picture number of the short term picture to be marked as "unused".

```
if(PNC < difference_of_pic_nums_minus1)
  PNQ = PNC – difference_of_pic_nums_minus1 – 1 + MAX_PN;
else
  PNQ = PNC – difference_of_pic_nums_minus1 - 1;
```

Similarly, the encoder may compute the difference_of_pic_nums_minus1 value to encode using the following relation:

```
if(PNC < PNQ)
  difference_of_pic_nums_minus1 = PNC – PNQ – 1 + MAX_PN;
else
  difference_of_pic_nums_minus1 = PNC – PNQ - 1;
```

8.3.6.7.3 Removal of long term pictures

If memory_management_control_operation equals 2 (Mark a Long-Term Picture as "Unused"), a specified long term picture in the long term buffer is marked as "unused", if that picture has not already been marked as "unused".

The field long_term_pic_idx indicates the the long term picture number, LTPN, of the picture to be marked as "unused".

NOTE: this use of long_term_pic_idx is different to its use when transferring short term pictures to the long term buffer.

8.3.6.7.4 Transfer of short term pictures to the long term buffer

If memory_management_control_operation equals 3 (Assign a Long-Term Index to a Picture), a specified short term picture in the short term buffer is transferred to the long term buffer with a specified long-term index, if that picture has not already been transferred to the long term buffer. If the picture specified in a long-term assignment operation is already associated with the required long_term_pic_idx, no action shall be taken by the decoder. The specified short term picture is no longer in the short term buffer following the processing of this command, and shall not be referenced at a later point in the bitstream by reference to its picture number.

If another picture is already present in the long term buffer with the same long-term index as the specified long-term index, the other picture is marked as "unused".

The picture in the short term buffer to be transferred is identified by its picture number, which is derived from difference_of_pic_nums_minus1 as in subclause **Error! Reference source not found.**

Deleted: 9.2.6.6.1

A top field in the short term buffer can only be transferred to the top field of a long term frame, and a bottom field in the short term buffer can only be transferred to the bottom field of a long term frame. The long term frame number of the frame into which the short term picture is transferred is given by long_term_pic_idx.

long_term_pic_idx shall not be greater than max_long_term_idx_plus1–1. If long_term_pic_idx does not satisfy this constraint, this condition should be treated by the decoder as an error.

For error resilience, the bitstream may contain the same long-term index assignment operation or max_long_term_idx_plus1 specification message repeatedly.

A bitstream shall not assign a long-term index to a short-term picture that has been marked as "unused" by the decoding process prior to the first such assignment message in the bitstream. A bitstream shall not assign a long-term index to a picture number that has not been sent.

Once a long-term picture index has been assigned to a picture, the only potential subsequent use of the long term picture's picture number within the bitstream shall be in a repetition of the long-term index assignment. long_term_pic_idx becomes the unique ID for the life of a long term picture.

8.3.6.7.5 Modification of the size of the long term buffer

If memory_management_control_operation equals 4 (Specify the Maximum Long-Term Frame Index), max_long_term_pic_idx_plus1 indicates the maximum index allowed for long-term reference frames (until receipt of another value of max_long_term_pic_idx_plus1).

If max_long_term_pic_idx_plus1 is smaller than its previous value, all frames in the long term buffer having indices greater than max_long_term_pic_idx_plus1 – 1 shall be marked "unused".

If max_long_term_pic_idx_plus1 is greater than its previous value, the capacity of the short term buffer is reduced by the same amount as the capacity of the long term buffer is increased. The memory_management_control_operation commands in the bitstream shall be such that at the time of processing this command, the reduced capacity of the short term buffer shall be sufficient for the contents of the short term buffer.

NOTE: max_long_term_pic_idx_plus1 can therefore be used to remove long term pictures from the long term buffer but can not be used to remove short term pictures from the short term buffer.

The frequency of transmitting max_long_term_idx_plus1 is out of the scope of this Recommendation. However, the encoder should send an max_long_term_idx_plus1 parameter upon receiving an error message, such as an Intra request message.

8.3.6.7.6 Buffer reset

If memory_management_control_operation equals 5 (Reset), or the current picture is an IDR picture, all pictures in the short and long term buffers are marked as "unused", and max_long_term_pic_idx_plus1 is set to zero.

8.3.6.8 Error resilience with reference picture buffer management

If required_frame_num_update_behaviour_flag equals 1 the following picture buffer management behaviour shall be used.

Deleted: required_frame_num_update_behaviour

If the decoder identifies that pictures that should have been stored have not been decoded, by a gap in frame numbers, the decoder shall act as if the missing pictures had been inserted into the reference picture buffer using the "Sliding Window" buffering mode. An index for a missing picture is called an "invalid" index. The decoder should infer an unintentional picture loss if any "invalid" index is referred to in motion compensation or if an "invalid" index is re-mapped.

If required_frame_num_update_behaviour_flag equals 0, the decoder should infer an unintentional picture loss if one or several frame numbers are missing or if a picture not stored in the reference picture buffer is indicated in an abs_diff_pic_num_minus1 or long_term_pic_idx field.

Deleted: required_frame_num_update_behaviour

Note: In case of an unintentional picture loss, the decoder may invoke some concealment process. If required_frame_num_update_behaviour_flag equals 1, the decoder may replace the picture corresponding to an "invalid" index with an error-concealed one and remove the "invalid" indication. Otherwise, the decoder may insert an error-concealed picture into the reference picture buffer assuming the "Sliding Window" buffering mode. Concealment may be conducted by copying the closest temporally preceding picture that is available in the reference picture buffer into the position of the missing picture. The temporal order of the short-term pictures in the reference picture buffer can be inferred from their picture numbers. In addition or instead, the decoder may send a forced intra update signal to the encoder by external means (for example, Recommendation H.245) if such external means is available, or the decoder may use external means or back-channel messages (for example, Recommendation H.245) to indicate the loss of pictures to the encoder if such external means is available.

Deleted: required_frame_num_update_behaviour

[Ed. Note: This section requires further editting by someone who understands the intentions.]

8.3.7 Decoding process for macroblock level frame/field adaptive coding

[Ed.Note: this should go somewhere else in the spec]

When mb_ frame_field_adaptive_flag = = 1, the decoded frame is scanned on a macroblock pair by macroblock pair basis, as shown in Figure 6-4 (subclause 6.3). A macroblock pair can be decoded in either frame or field decoding mode. For frame decoding mode, a macroblock pair is decoded as two frame macroblocks, and each can be further divided into one of the block patterns shown in Figure 6-5. For field coding mode, a macroblock pair is first split into one top-field macroblock and one bottom-field macroblock, as shown in Figure 8-3. The top-field macroblock and the bottom-field macroblock are further divided into block patterns shown in Figure 6-5. Each macroblock in either frame or field decoding mode can have a different mb_type described in subclause 7.4.6???

Deleted: Figure 6-4

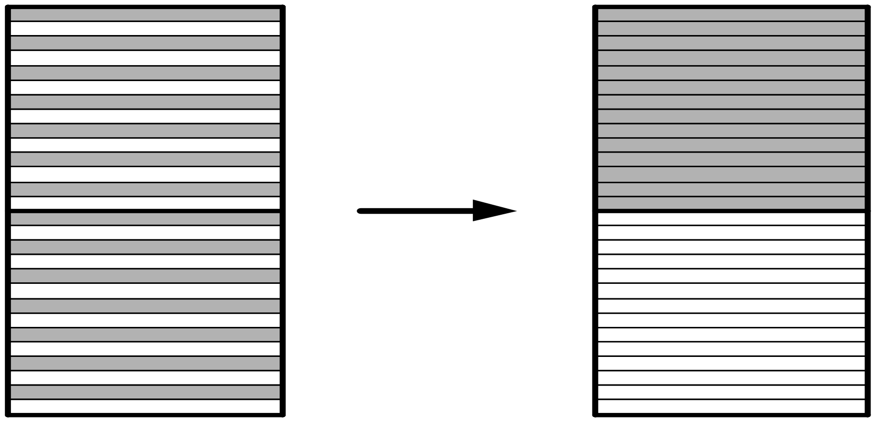


Figure 8-3 – Split of a pair of macroblocks into one top-field macroblock and one bottom-field macroblock.

When mb_field_decoding_flag == 0, the top macroblock of a macroblock pair is decoded first, followed by the bottom macroblock, as shown in Figure 6-4 (subclause 6.3). When mb_field_decoding_flag == 1, the top-field macroblock is decoded first, followed by the bottom-field macroblock (see Figure 6-4). A few specific rules/conventions are specified as follows.

For intra prediction, if a block/macroblock is in field decoding mode, its neighbouring samples in calculating the prediction shall be the neighbouring samples of the same field.

As in frame decoding mode, the prediction mode of a 4x4 field block is decoded based upon the prediction modes of the (above and left) neighbouring blocks. For interior blocks of a field macroblock pair, the neighbouring blocks used in coding of intra prediction mode are the blocks above and left of the current block. For boundary blocks of a field macroblock pair, the above or left neighbouring block may be in different macroblock pair that can be of either frame or field coding mode. The neighbouring blocks for these boundary blocks shall be as follows:

– If the above or the left macroblock pair is also in field decoding mode, the neighbours of the boundary blocks in the current macroblock pair are in the same field of the above or the left macroblock pair.
– If the above or the left macroblock pair is in frame decoding mode, the neighbours of the boundary blocks in the top (bottom) field macroblock are specified to be the corresponding blocks in the top (bottom) macroblock in the frame macroblock pair.
– For macroblock pairs on the upper boundary of a slice, if the left macroblock pair is in frame decoding mode, then the intra mode prediction value used to predict a field macroblock shall be set to DC prediction.

8.4 Motion compensation

The motion compensation process generates motion-compensated predictions for picture blocks using previously decoded reference pictures. The reference picture selection is described in subclauses 7.3.3, 7.3.5.1-7.3.5.2, 7.4.6.1???, and 8.3.6-8.3.7. If pic_structure indicates a field picture, only the reference field indicated by the ref_idx_l0 or ref_idx_l1 is used in the motion compensation. The motion vectors to be used are described in subclauses 7.3.5.1-7.3.5.2, 7.4.5, 7.4.6.1??? and 8.4.1.

Deleted: compensated

If the current macroblock pair is in frame mode and one or more neighbouring blocks is in field mode, the reference frame index used in MV prediction from the field coded neighbours is obtained by dividing the reference field list index by 2 and truncation of any fractional result toward zero to obtain the effective frame index for prediction. If the macroblock pair is in frame mode, the reference field number for any frame coded neighbour is obtained by multiplying the reference frame index by 2. [Ed.Note: Some workding problems here]

Deleted: MB

Deleted: MB

When a macroblock pairis in field mode, each field macroblock may refer to any (top or bottom) field in the reference picture buffer. [Ed.Note: Some wording problems]

8.4.1 Prediction of vector components

No vector component prediction takes place across macroblock boundaries of macroblocks that do not belong to the same slice. For the purpose of vector component prediction, macroblocks that do not belong to the same slice are treated as outside the picture.

With exception of the 16x8 and 8x16 block shapes, "median prediction" (see subclause 8.4.1.1) is used. In case the macroblock may be classified to have directional segmentation the prediction is specified in subclause 8.4.1.2. The motion vector for a Skip mode macroblock shall be obtained as described in subclause 8.4.1.3.

Deleted: Motion

8.4.1.1 Median prediction

The prediction of the components of the motion vector value for a block E is formed based on the parameters of neighbouring blocks A, B, C, and D as shown in Figure 8-4. This process is referred to as median prediction.

A The block containing the sample to the left of the upper left sample in E

B The block containing the sample just above the upper left sample in E

C The block containing the sample above and to the right of the upper right sample in E

D The block containing the sample above and to the left of the upper left sample in E

NOTE - The prediction of A, B, C, D and E may use different indices into the reference picture list.

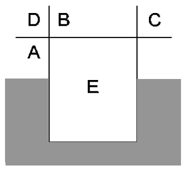


Figure 8-4 – Median prediction of motion vectors

The following rules specify the predicted motion vector value resulting from the median prediction process for block E:

– If block C is outside the current picture or slice or is not available due to the decoding order within a macroblock as specified in Figure 6-5, its motion vector and reference picture index shall be considered equal to the motion vector and reference picture index for block D.

Deleted: Figure 6-4

– If blocks B, C, and D are all outside the current picture or slice, their motion vector values and reference picture indices shall be considered as equal to the motion vector value and reference picture index for block A.

– If any predictor not specified by the first or second rules above is coded as intra or is outside the current picture or slice, its motion vector value shall be considered equal to zero and it shall be considered to have a different reference picture than block E.

– If only one of the three blocks A, B and C has the same reference picture as block E, then the predicted motion vector for block E shall be equal to the motion vector of the A, B, or C block with the same reference picture as block E; otherwise, each component of the predicted motion vector value for block E shall be the median of the corresponding motion vector component values for blocks A, B, and C.

The following additional considerations apply in the case of macroblock-adaptive frame/field coding: [Ed. Note: Significant rework of wording needed below]

– If a block A is field type then it is assigned two frame MV's for the purpose of motion vector prediction. The first frame MV is the field MV of the block with vertical motion vector component multiplied by 2, and the second MV is the field MV of the block in same geometric location as A in the second macroblock of the macroblock pair (vertical motion vector component multiplied by two). If a block A is frame type then it is assigned two field MV's for the purpose of motion vector prediction. The first field MV is the frame MV of the block with vertical motion vector component divided by 2, and the second MV is the frame MV of the block in same geometric location as A in the second macroblock of the macroblock pair (vertical motion vector component divided by two). Similar rules are used to determine the two reference frames (fields) of a field (frame) block.

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

- If E is in frame coding mode, the MVs of A, B, C and D used in calculating PMV are also frame-based. If block A, B, C, or D is coded in field coding mode, its two frame-based MVs are averaged. In that case, the two reference field numbers of A, B, C or D shall be the same, and they shall be equal to the reference frame number of E multplied by 2.
- If E is in field coding mode, the MVs of A, B, C and D used in calculating PMV are also field-based in the same field parity. If block A, B, C, or D is frame coded, the field-based motion vector is obtained by averaging the two field MVs of the block. In that case, two frame reference numbers of A, B, C or D shall be the same, and they shall be equal to the reference field number of E divided by 2 with truncation of fractional values toward zero.

8.4.1.2 Directional segmentation prediction

If the macroblock where the block to be predicted is coded in 16x8 or 8x16 mode, the prediction is generated as follows (refer to Figure 8-5 and the definitions of A, B, C, E above):

a) Vector block size 8x16:

1) Left block: A is used as prediction if it has the same reference picture as E, otherwise "median prediction" is used

2) Right block: C is used as prediction if it has the same reference picture as E, otherwise "median prediction" is used

b) Vector block size 16x8:

1) Upper block: B is used as prediction if it has the same reference picture as E, otherwise "median prediction" is used

2) Lower block: A is used as prediction if it has the same reference picture as E, otherwise "median prediction" is used

If the indicated prediction block is outside the picture, the same substitution rules are applied as in the case of median prediction.

[Ed.Note: rewrite] For field-coded macroblocks, the directional segmentation follow the same conventions as the above, but the neighbouring blocks are constructed from samples of the macroblock pair having the same field parity.

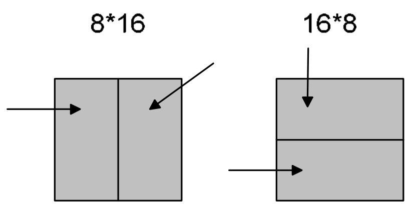


Figure 8-5 – Directional segmentation prediction

8.4.1.3 Motion vector for a Skip macroblock type

The motion vector for a Skip macroblock type shall be obtained identically to the prediction motion vector for the 16x16 macroblock type. However, if any of the conditions below hold, a zero motion vector shall be used instead:

a) The macroblock immediately above or to the left is not available (that is, is outside of the picture or belongs to a different slice)

b) Either one of the motion vectors applying to samples A or B (as described in subclause 8.4.1.1) uses the last decoded picture as reference and has zero magnitude.

Deleted: skip mode

Deleted: Motion

Deleted: mode

Deleted: Macroblock

8.4.1.4 Chroma vectors

Chroma vectors are derived from the luma vectors. Since chroma has half resolution compared to luma, the chroma vectors are obtained by dividing the corresponding luma motion vectors by two. [Ed. Note: (AJ) I'm not sure that this is true. chroma vectors are used with eighth-pel accuracy. Thus, they are not divided by 2, but re-mapped to eighth-pel chroma grid].

Due to the lower resolution of the chroma array relative to the luma array, a chroma vector applies to 1/4 as many samples as the luma vector.

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.

8.4.2 Fractional sample accuracy

Fractional sample accuracy is indicated by motion_resolution. If motion_resolution has the value 0**,** quarter-sample interpolation with a 6-tap filter is applied to the luma samples in the block. If motion_resolution has the value 1, eighth-sample interpolation with an 8-tap filter is used. All fractional luma samples shall be interpolated as described in subclauses 8.4.2.1 for quarter-pel accuracy and subclause 8.4.2.2 for eighth-pel accuracy. The interpolation process for fractional chroma samples in both cases is described in subclause 8.4.2.3.

Deleted: The prediction process for

If a sample referred to in the interpolation process (necessarily integer accuracy) is outside of the reference picture it shall be replaced by the nearest sample belonging to the picture (an edge or corner sample). Motion vectors are allowed to point to samples outside the reference picture. If a sample outside the reference picture is referred to in the prediction process, the nearest sample belonging to the picture (an edge or corner sample) shall be used. Reconstructed motion vectors shall be clipped to ±19 integer samples outside of the picture. [Ed. Note: Is that an encoder or decoder clipping process? Does it affect the values of the prediction of subsequent MVs? Description must be improved.]

8.4.2.1 Quarter sample luma interpolation

In Figure 8-6, the positions labelled with upper-case letters within shaded blocks represent reference picture samples at integer sample positions, and the positions labelled with lower-case letters within un-shaded blocks represent reference picture samples at fractional sample positions.

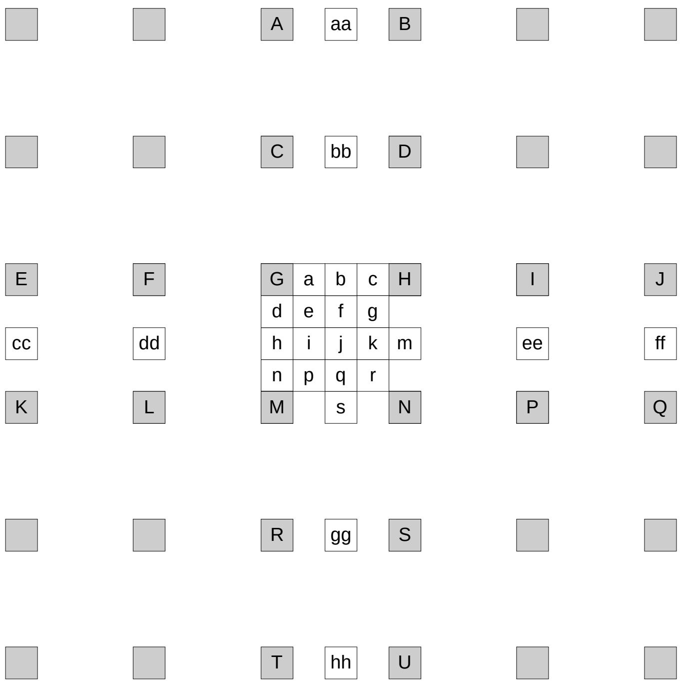


Figure 8-6 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation.

The luma prediction values at half sample positions shall be obtained 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 obtained by averaging samples at integer and half sample positions. The process for each fractional position is described below.

- The samples at half sample positions labelled 'b' shall be obtained by first calculating intermediate values denoted as 'b'' 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 obtained by first calculating intermediate values denoted as 'h'' by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

  b=(E-5F+20G+20H-5I+J),

  h=(A-5C+20G+20M-5R+T).

  The final prediction values shall be calculated using:

  b=Clip1((b+16)>>5),

  h= Clip1((h+16)>>5).

- The samples at half sample position labelled as 'j' shall be obtained by first calculating intermediate value denoted as 'j' 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 equivalent result.

  $j=cc-5dd+20h+20m-5ee+ff$, or

  $j=aa-5bb+20b+20s-5gg+hh$,

  where intermediate values denoted as 'aa', 'bb', 'gg', 's' and 'hh' shall be obtained by applying the 6-tap filter horizontally in an equivalent manner to 'b' and intermediate values denoted as 'cc', 'dd', 'ee', 'm' and 'ff' shall be obtained by applying the 6-tap filter vertically in an equivalent manner to 'h'. The final prediction value shall be calculated using: j=Clip1((j+512)>>10).

- The samples at quarter sample positions labelled as 'a', 'c', 'd', 'n', 'f', 'i', 'k' and 'q' shall be obtained by averaging with truncation the two nearest samples at integer and half sample positions using: a=(G+b)>>1, c=(H+b)>>1, d=(G+h)>>1, n=(M+h)>>1, f=(b+j)>>1, i=(h+j)>>1, k=(j+m)>>1 and q=(j+s)>>1.
- The samples at quarter sample positions labelled as 'e', 'g' and 'p' shall be obtained by averaging with truncation the two nearest samples at half sample positions in the diagonal direction using e=(b+h)>>1, g=(b+m)>>1, p=(h+s)>>1.
- The sample at quarter sample position labelled as 'r' shall be obtained by averaging with rounding using the four nearest samples at integer positions using r=(G+H+M+N+2)>>2.

8.4.2.2 One eighth sample luma interpolation

The positions labelled 'A' in Figure 8-7 represent reference picture samples in integer positions. Other symbols represent interpolated values at fractional sample positions.

A	d	b^h	d	b^h	d	b^h	d	A
d	e	d	f^h	d	f^h	d	e	
b^v	d	c^q	d	c^q	d	c^q	d	b^v
d	f^v	f	g	d	g	d	f^v	
b^v	d	c^q	d	c^m	d	c^q	d	b^v
d	f^v	d	g	d	g	d	f^v	
b^v	d	c^q	d	c^q	d	c^q	r	b^v
d	e	d	f^h	d	f^h	d	e	
A		b^h		b^h		b^h		A

Figure 8-7 – Integer samples (‘A‘) and fractional sample locations for one eighth sample luma interpolation

The samples at half and quarter sample positions shall be obtained by applying 8-tap filters with following coefficients:

– coeff1 for sample values at 1/4 positions: (-3, 12, -37, 229, 71, -21, 6, -1),

– coeff2 for sample values at 2/4 positions: (-3, 12, -39, 158, 158, -39, 12, -3),

– coeff3 for sample values at 3/4 positions: (-1, 6, -21, 71, 229, -37, 12, -3).

The samples at one eighth sample positions are specified as weighted averages of reference picture samples at integer, half and quarter sample positions. The process for each position is described below.

– The samples at half and quarter sample positions denoted as ‘b^h’ shall be obtained by first calculating intermediate values *b*, by applying 8-tap filter to the nearest samples ‘A’ at integer positions in a horizontal direction. For left ‘b^h’, middle ‘b^h’ and right ‘b^h’ in Figure 8-7, coefficients coeff1, coeff2 and coeff3 are used, respectively. The final value of ‘b^h’ shall be obtained using b^h = Clip1((*b*+128)>>8). The samples at half and quarter sample positions labelled as ‘b^v’ shall be obtained equivalently with the filter applied in vertical direction. For upper ‘b^v’, middle ‘b^v’ and bottom ‘b^v’ coefficients coeff1, coeff2 and coeff3 are used, respectively.

– The samples at half and quarter sample positions labelled as ‘c^m’ and ‘c^q’ shall be obtained by 8-tap filtering of the closest intermediate values *b* in either horizontal or vertical direction to obtain value *c*, and then the final result shall be obtained using c^m=Clip1((*c*+32768)>>16) or c^q = Clip1((*c*+32768)>>16). Filtering in horizontal and vertical direction gives identical results. When filtering in horizontal direction is applied, for left ‘c^q’, middle ‘c^q’ and right ‘c^q’, coefficients coeff1, coeff2 and coeff3 are used, respectively. When filtering in vertical direction is applied, for upper ‘c^q’, middle ‘c^q’ and bottom ‘c^q’ coefficients coeff1, coeff2 and coeff3 are used, respectively. For ‘c^m’ coefficients coeff2 are used.

– The samples at one eighth sample positions labelled as ‘d’ shall be obtained by averaging with truncation of the two closest samples at half and quarter sample positions using d = (A+b^h)>>1, d = (b^h+b^h)>>1, d = (A+b^v)>>1, d = (b^h+c^q)>>1, d = (b^v+c^q)>>1, d = (c^q+c^q)>>1, d = (b^v+b^v)>>1, or d = (c^q+c^m)>>1.

– The samples at one eighth sample positions labelled as ‘e’ shall be obtained by averaging with truncation the closest ‘b^h’ and ‘b^v’ samples in diagonal direction using e = (b^h+b^v)>>1.

– The samples at one eighth sample positions labelled as ‘g’ shall be obtained from the closest integer samples ‘A’ and the ‘c^m’ samples using g = (A+3c^m+2)>>2.

– The samples at one eighth sample positions labelled as ‘f^h’ and ‘f^v’ shall be calculated as f^h = (3b^h+ b^v+2)>>2 and f^v = (3b^v+ b^h+2)>>2.

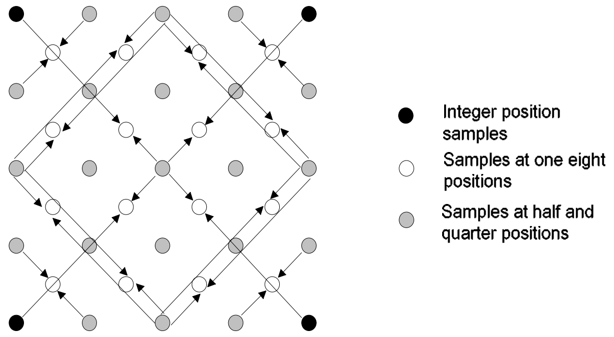


Figure 8-8 – Diagonal interpolation for one eighth sample luma interpolation

8.4.2.3 Chroma interpolation

Motion-compensated chroma prediction values at fractional sample positions shall be obtained using Equation 8-13.

Deleted: compensated

Deleted: prediction fractional chroma samples

$$v = ((s - d^x)(s - d^y)A + d^x(s - d^y)B + (s - d^x)d^yC + d^xd^yD + s^2/2)/s^2 \qquad (8\text{-}13)$$

where A, B, C and D are the integer position reference picture samples surrounding the fractional sample location; d^x and d^y are the fractional parts of the sample position in units of one eighth samples for quarter sample interpolation or one sixteenth samples for one eighth sample interpolation; and s is 8 for quarter sample interpolation and s is 16 for one eighth sample interpolation. The relationships between the variables in Equation 8-13 and reference picture positions are illustrated in Figure 8-9.

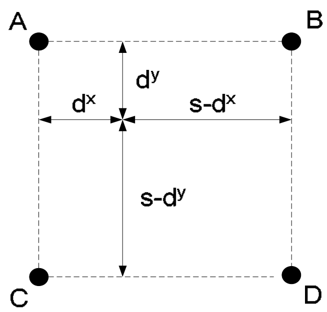


Figure 8-9 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D.

8.5 Intra Prediction

Two Intra coding modes for macroblocks are described below. Sample values used in the prediction process for intra sample prediction shall be sample values prior to alteration by any deblocking filter operations.

8.5.1 Intra Prediction for 4x4 luma block in Intra_4x4 macroblock type

Figure 8-10 illustrates the Intra prediction for a 4x4 luma block. The samples of a 4x4 luma block containing samples a to p in Figure 8-10 are predicted using samples A to Q in Figure 8-10 from neighbouring blocks. Samples A to Q may already be decoded and then be used for prediction. Any sample A-Q shall be considered not available and not used for prediction under the following circumstances:

- if they are outside the picture or outside the current slice,
- if they belong to a macroblock that is subsequent to the current macroblock in raster scan order,

– if they are sent later than the current 4x4 block in the order shown in Figure 6-5, or
– if they are in a non-intra macroblock and constrained_intra_pred is 1.

When samples E-H are not available, the sample value of D is substituted for samples E-H. When samples M-P are not available, the sample value of L is substituted for samples M-P.

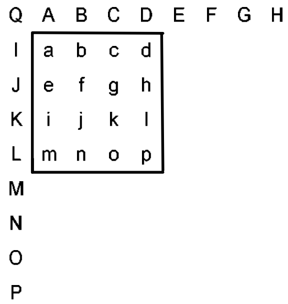


Figure 8-10 – Identification of samples used for intra spatial prediction

For the luma signal, there are nine intra prediction modes labelled 0, 1, 3, 4, 5, 6, 7, and 8. Mode 2 is 'DC-prediction' (see below). The other modes represent directions of predictions as indicated in Figure 8-11.

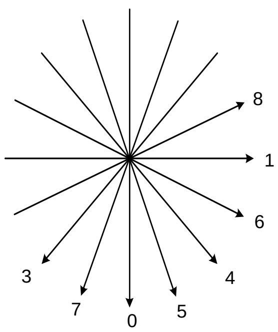


Figure 8-11 – Intra prediction directions

If adaptive_block_size_transform_flag = = 1, the intra prediction modes for 4x8, 8x4, and 8x8 luma blocks as specified in subclause 12.4.1.

8.5.1.1 Mode 0: vertical Prediction

This mode shall be used only if A, B, C, D are available. The prediction in this mode shall be as follows:

– a, e, i, m are predicted by A,
– b, f, j, n are predicted by B,
– c, g, k, o are predicted by C,
– d, h, l, p are predicted by D.

8.5.1.2 Mode 1: horizontal prediction

This mode shall be used only if I, J, K, L are available. The prediction in this mode shall be as follows:

– a, b, c, d are predicted by I,

– e, f, g, h are predicted by J,

– i, j, k, l are predicted by K,

– m, n, o, p are predicted by L.

8.5.1.3 Mode 2: DC prediction

If all samples A, B, C, D, I, J, K, L, are available, all samples are predicted by (A+B+C+D+I+J+K+L+4)>>3. If A, B, C, and D are not available and I, J, K, and L are available, all samples shall be predicted by (I+J+K+L+2)>>2. If I, J, K, and L are not available and A, B, C, and D are available, all samples shall be predicted by (A+B+C+D+2)>>2. If all eight samples are not available, the prediction for all luma samples in the 4x4 block shall be 128. A block may therefore always be predicted in this mode.

8.5.1.4 Mode 3: diagonal down/left prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are available. This is a 'diagonal' prediction. The prediction in this mode shall be as follows:

– a is predicted by (A + 2B + C + I + 2J + K + 4) >> 3

– b, e are predicted by (B + 2C + D + J + 2K + L + 4) >> 3

– c, f, i are predicted by (C + 2D + E + K + 2L + M + 4) >> 3

– d, g, j, m are predicted by (D + 2E + F + L + 2M + N + 4) >> 3

– h, k, n are predicted by (E + 2F + G + M + 2N + O + 4) >> 3

– l, o are predicted by (F + 2G + H + N + 2O + P + 4) >> 3

– p is predicted by (G + H + O + P + 2) >> 2

8.5.1.5 Mode 4: diagonal down/right prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are available. This is a 'diagonal' prediction. The prediction in this mode shall be as follows:

– m is predicted by: (J + 2K + L + 2) >> 2

– i, n are predicted by (I + 2J + K + 2) >> 2

– e, j, o are predicted by (Q + 2I + J + 2) >> 2

– a, f, k, p are predicted by (A + 2Q + I + 2) >> 2

– b, g, l are predicted by (Q + 2A + B + 2) >> 2

– c, h are predicted by (A + 2B + C + 2) >> 2

– d is predicted by (B + 2C + D + 2) >> 2

8.5.1.6 Mode 5: vertical-left [Ed. Note: right ?] prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are inside the slice. This is a 'diagonal' prediction.

– a, j are predicted by (Q + A + 1) >> 1

– b, k are predicted by (A + B + 1) >> 1

– c, l are predicted by (B + C + 1) >> 1

– d is predicted by (C + D + 1) >> 1

– e, n are predicted by (I + 2Q + A + 2) >> 2

– f, o are predicted by (Q + 2A + B + 2) >> 2

– g, p are predicted by (A + 2B + C + 2) >> 2

– h is predicted by (B + 2C + D + 2) >> 2

– i is predicted by (Q + 2I + J + 2) >> 2

– m is predicted by (I + 2J + K + 2) >> 2

8.5.1.7 Mode 6: horizontal-down prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are available. This is a 'diagonal' prediction. The prediction in this mode shall be as follows:

– a, g are predicted by (Q + I + 1) >> 1

- b, h are predicted by (I + 2Q + A+ 2) >> 2
- c is predicted by (Q + 2A + B+ 2) >> 2
- d is predicted by (A + 2B + C+ 2) >> 2
- e, k are predicted by (I + J + 1) >> 1
- f, l are predicted by (Q + 2I + J+ 2) >> 2
- i, o are predicted by (J + K + 1) >> 1
- j, p are predicted by (I + 2J + K+ 2) >> 2
- m is predicted by (K + L + 1) >> 1
- n is predicted by (J + 2K + L + 2) >> 2

8.5.1.8 Mode 7: vertical-right [Ed. Note: left ?] prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are available. This is a 'diagonal' prediction. The prediction in this mode shall be as follows:

- a is predicted by (2A + 2B + J + 2K + L + 4) >> 3
- b, i are predicted by (B + C + 1) >> 1
- c, j are predicted by (C + D + 1) >> 1
- d, k are predicted by (D + E + 1) >> 1
- l is predicted by (E + F + 1) >> 1
- e is predicted by (A + 2B + C + K + 2L + M + 4) >> 3
- f, m are predicted by (B + 2C + D + 2) >> 2
- g, n are predicted by (C + 2D + E + 2) >> 2
- h, o are predicted by (D + 2E + F + 2) >> 2
- p is predicted by (E + 2F + G + 2) >> 2

8.5.1.9 Mode 8: horizontal-up prediction

This mode shall be used only if all A, B, C, D, I, J, K, L, Q are available. This is a 'diagonal' prediction. The prediction in this mode shall be as follows:

- a is predicted by (B + 2C + D + 2I + 2J + 4) >> 3
- b is predicted by (C + 2D + E + I + 2J + K + 4) >> 3
- c, e are predicted by (J + K + 1) >> 1
- d, f are predicted by (J + 2K + L + 2) >> 2
- g, i are predicted by (K + L + 1) >> 1
- h, j are predicted by (K + 2L + M + 2) >> 2
- l, n are predicted by (L + 2M + N + 2) >> 2
- k, m are predicted by (L + M + 1) >> 1
- o is predicted by (M + N + 1) >> 1
- p is predicted by (M + 2N + O + 2) >> 2

8.5.2 Intra prediction for 16x16 luma block in Intra_16x16 macroblock type

Let P(-1,y), with y=-1,0,...,15 and P(x,-1), with x=0,…,15 denote the 33 neighbouring samples immediately above and/or to the left of the 16x16 luma block. Let Pred(x,y) x,y = 0..15 denote the prediction for the 16x16 luma block samples. There are 4 different prediction modes as specified in subclauses 8.5.2.1 to 8.5.2.4. Only Mode 3 uses P(-1,-1).

Samples P(x,-1) or P(-1,y) shall be considered not available if they are outside the picture or outside the current slice, or if they are in a non-intra macroblock and constrained_intra_pred is 1. If none of the neighbouring samples are available, Mode 2 (DC prediction) is the only possible 16x16 luma prediction mode.

8.5.2.1 Mode 0: vertical prediction

This mode shall be used only if all neighbouring samples P(x, -1), with x = 0..15 are available.

$$\mathrm{Pred}(x, y) = P(x, -1), \text{ with } x, y = 0..15 \qquad (8\text{-}14)$$

Deleted: Denote the block to be predicted as having sample locations 0 to 15 horizontally and 0 to 15 vertically. The notation P(x,y) is used, where x = 0..15 corresponds to horizontal positions and y = 0..15 corresponds to vertical positions. P(x,-1), x=0..15 are the neighbouring samples above the block and P(-1,y), y=0..15 are the neighbouring samples to the left of the block. Pred(x,y) x,y = 0..15 is the prediction for the luma macroblock samples. There are 4 different prediction modes as specified in subclauses 8.5.2.1to 8.5.2.4.[Ed. Note: Should this description indicate that P(-1,-1) is also used, in the case of plane prediction?]¶
Samples P(x,-1) or P(-1,y) shall be considered not available under the following circumstances:¶
– if they are outside the picture or outside the current slice, or¶
– if they are in a non-intra macroblock and constrained_intra_pred is 1.¶

Formatted: Bullets and Numbering

8.5.2.2 Mode 1: horizontal prediction

This mode shall be used only if all neighbouring samples P(-1, y) , y=0..15 are available.

$$\text{Pred}(x, y) = P(-1, y), \text{ with } x, y = 0..15 \tag{8-15}$$

8.5.2.3 Mode 2: DC prediction

$$\text{Pred}(x, y) = [\sum_{x'=0}^{15} P(x',-1) + \sum_{y'=0}^{15} P(-1, y') + 16] >> 5 \quad \text{with } x, y = 0..15 \tag{8-16}$$

If the neighbouring samples P(x, -1) are not available and the neighbouring samples P(-1, y) are available, the prediction for all luma samples in the macroblock is given by Equation 8-17.

$$\text{Pred}(x, y) = [\sum_{y'=0}^{15} P(-1, y') + 8] >> 4 \quad \text{with } x, y = 0..15, \tag{8-17}$$

If the neighbouring samples P(-1, y) are not available and the neighbouring samples P(x, -1) are available, the prediction for all luma samples in the macroblock is given by Equation 8-18.

$$\text{Pred}(x, y) = [\sum_{x'=0}^{15} P(x',-1) + 8] >> 4 \quad \text{with } x, y = 0..15, \tag{8-18}$$

If none of the neighbouring samples P(x,-1) and P(-1,y) are available, the prediction for all luma samples in the macroblock shall be 128.

8.5.2.4 Mode 3: plane prediction

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 available.

$$\text{Pred}(x,y) = \text{Clip1}((a + b\cdot(x-7) + c\cdot(y-7) +16) >> 5), \tag{8-19}$$

where:

$$a = 16\cdot(P(-1,15) + P(15,-1)) \tag{8-20}$$

$$b = (5*H+32)>>6 \tag{8-21}$$

$$c = (5*V+32)>>6 \tag{8-22}$$

and H and V are specified in Equations 8-23 and 8-24.

$$H = \sum_{x=1}^{8} x \cdot (P(7+x,-1) - P(7-x,-1)) \tag{8-23}$$

$$V = \sum_{y=1}^{8} y \cdot (P(-1,7+y) - P(-1,7-y)) \tag{8-24}$$

8.5.3 Prediction in intra coding of chroma blocks

The chroma in intra macroblocks is predicted in a manner very similar to the luma block in Intra_16x16 macroblock type (subclause 8.5.2), using one of four prediction modes. The same prediction mode is applied to both chroma blocks, but it is independent of the prediction mode used for the luma.

NOTE - If any portion of the luma macroblock is coded in intra mode, the entire chroma macroblock is coded intra.

Let P(-1,y), y=-1,0,...,7 and P(x,-1), with x=-1,0,…,7 denote the 17 neighbouring samples immediately above and/or to the left of the 8x8 chroma block. Let Pred(x,y) with x,y = 0..7 denote the prediction for the 8x8 chroma block samples. There are 4 different prediction modes as specified in subclauses 8.5.2.1 to 8.5.2.4. Only Mode 3 uses P(-1,-1).

Deleted: 71

Samples P(x,-1) or P(-1,y) shall be considered not available if they are outside the picture or outside the current slice, or if they are in a non-intra macroblock and constrained_intra_pred is 1. If none of the neighboring samples are available, Mode 2 (DC prediction) is the only possible 8x8 chroma prediction mode.

8.5.3.1 Mode 0: vertical prediction

This mode shall be used only if all neighbouring samples P(x, -1), with x = 0..7 are available.

$$\mathrm{Pred}(x, y) = P(x, -1), \text{ with } x, y = 0..7 \tag{8-25}$$

8.5.3.2 Mode 1: horizontal prediction

This mode shall be used only if all neighbouring samples P(-1, y), with y = 0..7 are available.

$$\mathrm{Pred}(x, y) = P(-1, y), \text{ with } x, y = 0..7 \tag{8-26}$$

8.5.3.3 Mode 2: DC prediction

If all samples P(-1,*n*) and P(*n*,-1) used in Equation 8-27 are available, the prediction is formed as

$$\mathrm{Pred}(x, y) = \left(\left(\sum_{n=0}^{7}\left(P(-1,n)+P(n,-1)\right)\right)+8\right) >> 4 \quad \text{with } x, y = 0..7, \tag{8-27}$$

If the 8 samples P(-1,*n*) are not available but the 8 samples P(*n*,-1) are available, the prediction is formed as

$$\mathrm{Pred}(x,y) = \left[\left(\sum_{n=0}^{7}P(n,-1)\right)+4\right] >> 3 \text{ with } x, y = 0..7, \tag{8-28}$$

If the 8 samples P(*n*,-1) are not available but the 8 samples P(-1, *n*) are available, the prediction is formed as

$$\mathrm{Pred}(x,y) = \left[\left(\sum_{n=0}^{7}P(n,-1)\right)+4\right] >> 3 \text{ with } x, y = 0..7, \tag{8-29}$$

If none of the neighbouring samples P(x,-1) and P(-1,y) are available, the prediction for all luma samples in the macroblock shall be 128.

8.5.3.4 Mode 3: plane prediction

For the plane mode, the prediction is formed as:

$$\mathrm{Pred}(x,y) = \mathrm{Clip1}(\ (a + b\cdot(x-3) + c\cdot(y-3) +16) >> 5\ ), \text{ with } x, y = 0,\ldots,7 \tag{8-30}$$

where:

$$a = 16\cdot(P(-1,7) + P(7,-1)) \tag{8-31}$$

$$b = (17*H+16)>>5 \tag{8-32}$$

$$c = (17*V+16)>>5 \tag{8-33}$$

and H and V are specified as:

$$H = \sum_{x=1}^{4} x\cdot\left[P(3+x,-1)-P(3-x,-1)\right] \tag{8-34}$$

$$V = \sum_{y=1}^{4} y\cdot\left[P(-1,3+y)-P(-1,3-y)\right] \tag{8-35}$$

Deleted: Let P(x,-1), x=0..7 be the neighbouring samples above the chroma macroblock and P(-1,y), y=0..7 be the neighbouring samples to the left of the chroma macroblock. Pred(x,y), x,y = 0..7 is the prediction for the whole chroma macroblock, and is computed as follows for the four prediction modes. Samples P(x,-1) or P(-1,y) shall be considered not available under the following circumstances:¶
– if they are outside the picture or outside the current slice,¶
– if they are in a non-intra macroblock and constrained_intra_pred is 1.¶
Whenever P(x,y) is not available, P(x,y) is inferred to have the value 128 except as specified in subclause 8.5.3.3.¶
For the horizontal and vertical prediction, P(x,y) is first filtered using a {1,2,1}/4 filter, with pixel replication at the edges.¶

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

8.6 Transform coefficient decoding and picture construction prior to deblocking

This subclause specifies aspects related to transform coefficient decoding and picture construction.

8.6.1 Zig-zag scan

The decoder maps the sequence of transform coefficient levels to the transform coefficient level positions. For this mapping, the scanning pattern is shown in Figure 8-12.

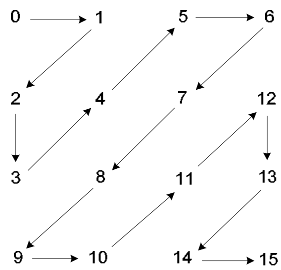


Figure 8-12 – Zig-zag scan

In the case of 16x16 intra macroblocks, the coefficients of the 4x4 luma DC transform are scanned in the same scan order as ordinary 4x4 coefficient blocks. Then for each 4x4 block of luma coefficients with AC coefficients to scan, the 15 remaining coefficients are scanned by starting the zig-zag scan at its second position.

The coefficients of the 2x2 chroma DC transform are scanned in raster order. Then for each 4x4 block of chroma coefficients with AC coefficients to scan, the 15 remaining coefficients are scanned by starting the zig-zag scan at its second position.

If adaptive_block_size_transform_flag = = 1, 4x4, 4x8, 8x4, and 8x8 luma coefficient blocks are scanned using zig-zag scans and field scans as specified in subclause 12.4.2.

8.6.2 Scaling and transformation

There are 52 different values of QP values that are used, ranging from 0 to 51, inclusive. The value of QP_C for chroma is determined from the current value of QP_Y. The scaling equations are specified such that the equivalent scaling parameter doubles for every increment of 6 in QP. Thus, there is an increase in scaling magnitude of approximately 12% from one QP to the next.

The value of QP_C shall be determined from the value of QP_Y as specified in Table 8-2:

Table 8-2 – Specification of QP_C as a function of QP_Y

QP_Y	<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	$=QP_Y$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

QP_Y shall be used as the QP to be applied for luma scaling and QP_C shall be used for chroma scaling.

The coefficients $R_{ij}^{(m)}$ specified in Equation 8-36 are used in Equations 8-39, 8-40, 8-41, 8-43 and 8-44.

$$R_{ij}^{(m)} = \begin{cases} V_{m0} & \text{for} \quad (i,j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ V_{m1} & \text{for} \quad (i,j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ V_{m2} & \text{otherwise}; \end{cases} \tag{8-36}$$

where the first and second subscripts of V are row and column indices, respectively, of the matrix specified as:

Deleted: <#>Mode 0: vertical prediction¶
$F(0,-1) = (P(0,-1) + P(1,-1)$ (8-25)¶
$F(x,-1) = (P(x-1,-1) + 2 \times P$, x=1,…,6 (8-26)¶
$F(7,-1) = (P(6,-1) + P(7,-1)$ (8-27)¶
Pred(x, y) = F(x, -1), x, y=0..7 (8-28)¶
<#>Mode 1: horizontal prediction¶
$F(-1,0) = (P(-1,0) + P(-1,1)$ (8-29)¶
$F(-1,y) = (P(-1,y-1) + 2 \times$, y=1,…,6 (8-30)¶
$F(-1,7) = (P(-1,6) + P(-1,7)$ (8-31)¶
Pred(x, y) = F(-1, y), x, y=0..7 (8-32)¶
<#>Mode 2: DC prediction¶
If all samples P(-1,n) and P(n,-1) used in Equation 8-33 are available, the prediction is formed as¶
Pred(x, y) = $\left(\left(\sum_{n=0}^{7}(P(-1,n)+P(n,-1))\right)+8\right) >$
x,y=0..7, (8-33)¶
If the 8 samples P(-1,n) are not available, the prediction is formed as¶
Pred(x,y) = $\left[\left(\sum_{n=0}^{7} P(n,-1)\right)+4\right] >>$
3 x,y=0..7, (8-33a)¶
If the 8 samples P(n,-1) are not available, the prediction is formed as¶
Pred(x,y) = $\left[\left(\sum_{n=0}^{7} P(n,-1)\right)+4\right] >>$
3 x,y=0..7, (8-33b)¶
If all 16 samples are not available, the prediction Pred(x,y) for all samples x,y=0..7is 128.¶
<#>Mode 3: plane prediction¶
For the plane mode, the prediction is formed as:¶
Pred(i,j) = Clip1((a + b·(i-3) + c·(j-3) +16) >> 5), i,j=0,…,7 (8-34)¶
where:¶
a = 16·(P(-1,7) + P(7,-1)) (8-35)¶
b = (17*H+16)>>5) (8-3 ... [9]

Formatted: Bullets and Numbering

Deleted: defines

Deleted: 40

Deleted: 43

Deleted: 44

Deleted: 46

Deleted: 47

Deleted: 48

Deleted: 40

$$V = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \qquad (8\text{-}37)$$

Deleted: 41

8.6.2.1 Luma DC coefficients for Intra_16x16 macroblock type

After decoding the coefficient levels for a 4x4 block of luma DC coefficients coded in 16x16 intra mode and assembling these into a 4x4 matrix C of elements c_{ij}, a transform process shall be applied in a manner mathematically equivalent to the following process. The process uses application of a transform before the scaling process.

The transform for the 4x4 luma DC coefficients in 16x16 intra macroblocks is specified by:

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \qquad (8\text{-}38)$$

Deleted: 42

A bitstream conforming to this Recommendation | International Standard shall not contain indicated 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 transform, scaling is performed according to the following:

a) If QP is greater than or equal to 12, then the scaled result shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] << (QP/6-2), \quad i, j = 0,\ldots,3. \qquad (8\text{-}39)$$

Deleted: 43

b) If QP is less than 12, then the scaled results shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)} + 2^{1-QP/6}] >> (2-QP/6), \quad i, j = 0,\ldots,3. \qquad (8\text{-}40)$$

Deleted: 44

A bitstream conforming to this Recommendation | International Standard shall not contain data that results in element of DC_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.6.2.2 Chroma DC coefficients

After decoding the coefficient levels for a 2x2 block of chroma DC coefficients and assembling these into a 2x2 matrix C of elements c_{ij}, the transform process is applied before the scaling process.

Definition of transform:

$$F = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \qquad (8\text{-}41)$$

Deleted: 45

A bitstream conforming to this Recommendation | International Standard shall not contain indicated 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 transform, scaling is performed according to the following.

a) If QP is greater than or equal to 6, then the scaling result shall be calculated as

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] << (QP/6-1), \quad i, j = 0,...,3. \qquad (8\text{-}42)$$

Deleted: 46

b) If QP is less than 6, then the scaling results shall be calculated by

$$DC_{ij} = [F_{ij} \cdot R_{00}^{(QP\%6)}] >> 1, \quad i, j = 0,...,3. \qquad (8\text{-}43)$$

Deleted: 47

A bitstream conforming to this Recommendation | International Standard shall not contain indicated data that results in any element of DC_{ij} that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

8.6.2.3 Residual 4x4 blocks

Scaling of 4x4 block coefficient levels c_{ij} other than those as specified in subclauses 8.6.2.1and 8.6.2.2shall be performed according to Equation 8-44

$$w_{ij} = [c_{ij} \cdot R_{ij}^{(QP\%6)}] << (QP/6), \quad i, j = 0,...,3 . \qquad (8\text{-}44)$$

Deleted: 48

Deleted: 48

A bitstream conforming to this Recommendation | International Standard shall not contain indicated 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 matrix W of elements w_{ij} illustrated as

$$W = \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \qquad (8\text{-}45)$$

Deleted: 49

in which the w_{00} element may be a result DC_{ij} from Equation 8-39, 8-40, 8-42, or 8-43; or may be from Equation 8-44, as appropriate, 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:

Deleted: 43

Deleted: 44

Deleted: 46

Deleted: 47

Deleted: 48

a) First, each row of reconstructed transform coefficients is transformed using a one-dimensional transform, and

b) Second, each column of the resulting matrix is transformed using the same one-dimensional transform.

The one-dimensional transform is specified as follows for four input samples w_0, w_1, w_2, w_3.

a) First, a set of intermediate values is computed:

$$z_0 = w_0 + w_2 \qquad (8\text{-}46)$$

$$z_1 = w_0 - w_2 \qquad (8\text{-}47)$$

$$z_2 = (w_1 >> 1) - w_3 \qquad (8\text{-}48)$$

$$z_3 = w_1 + (w_3 >> 1) \qquad (8\text{-}49)$$

Deleted: 50

Deleted: 51

Deleted: 52

Deleted: 53

b) Then the transformed result is computed from these intermediate values

$$x_0 = z_0 + z_3 \qquad (8\text{-}50)$$

$$x_1 = z_1 + z_2 \qquad (8\text{-}51)$$

$$x_2 = z_1 - z_2 \qquad (8\text{-}52)$$

$$x_3 = z_0 - z_3 \qquad (8\text{-}53)$$

Deleted: 54

Deleted: 55

Deleted: 56

Deleted: 57

A bitstream conforming to this Recommendation | International Standard shall not contain indicated 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. A bitstream conforming to this Recommendation | International Standard shall not contain indicated 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 transform in both the horizontal and vertical directions to produce a block of transformed samples,

$$X' = \begin{bmatrix} x'_{00} & x'_{01} & x'_{02} & x'_{03} \\ x'_{10} & x'_{11} & x'_{12} & x'_{13} \\ x'_{20} & x'_{21} & x'_{22} & x'_{23} \\ x'_{30} & x'_{31} & x'_{32} & x'_{33} \end{bmatrix}, \qquad (8\text{-}54)$$

the final reconstructed sample residual values shall be obtained as

$$X''_{ij} = [x'_{ij} + 2^5] >> 6 \qquad (8\text{-}55)$$

If adaptive_block_size_transform_flag = = 1, scaling and inverse transform for 4x8, 8x4, and 8x8 coefficient blocks is specified in subclause 12.4.3.

8.6.3 Adding decoded samples to prediction with clipping

Finally, the reconstructed sample residual values X'' from Equation 8-55 are added to the prediction values P_{ij} from motion-compensated prediction or spatial prediction and clipped to the range of 0 to 255 to form the final decoded sample result prior to application of the deblocking filter:

$$S'_{ij} = \text{Clip1}(\, P_{ij} + X''_{ij} \,) \qquad (8\text{-}56)$$

8.7 Deblocking Filter

A conditional filtering shall be applied to all macroblocks of a picture. This filtering is done on a macroblock basis, with macroblocks being processed in raster-scan order throughout the picture. For luma, as the first step, the 16 samples of the 4 vertical edges of the 4x4 raster shall be filtered beginning with the left edge, as shown on the left-hand side of Figure 8-13. Filtering of the 4 horizontal edges (vertical filtering) follows in the same manner, beginning with the top edge, as shown on the right-hand side of Figure 8-13. The same ordering applies for chroma filtering, with the exception that 2 edges of 8 samples each are filtered in each direction. [The chroma filtering description here needs further clarification.] This process also affects the boundaries of the already reconstructed macroblocks above and to the left of the current macroblock. Picture edges are not filtered.

When mb_adaptive_frame_field_flag = 1, a macroblock may be coded in frame or field decoding mode. For frame macroblock, deblocking is performed on the frame samples. In this case, if neighbouring macroblock pairs are field macroblocks, they shall be converted into frame macroblock pairs (Figure 8-3) before deblocking. For field macroblock, deblocking is performed on the field samples of the same field parity. In this case, if neighbouring macroblock pairs are frame macroblocks, they shall be converted into field macroblock pairs (Figure 8-3) before deblocking.

Deleted: 58

Deleted: 59

Deleted: 59

Deleted: motion

Deleted: 60

Deleted: in

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

Deleted: MB

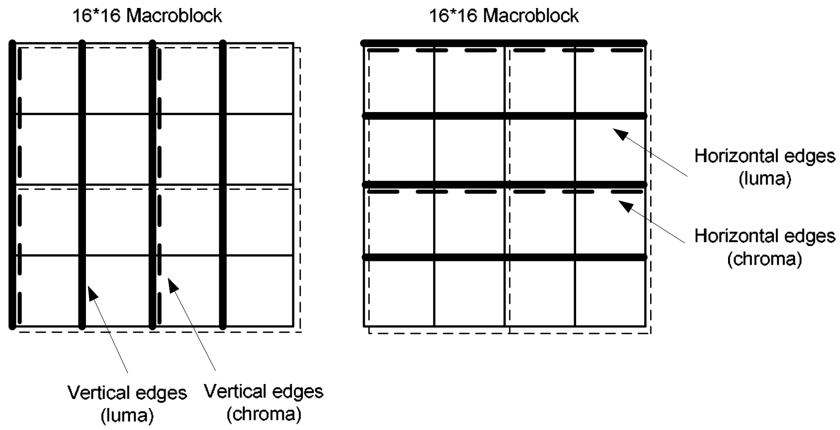


Figure 8-13 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines)

Intra prediction of a macroblock shall be done using the unfiltered content of the already decoded neighbouring macroblocks. Depending on the implementation, the values necessary for intra prediction may need to be stored before filtering in order to be used in the intra prediction of the macroblocks to the right and below the current macroblock.

When pic_structure indicates a field picture all decoding operations for the deblocking filter are based solely on samples within the current field. [Ed.Note: what about macroblock based AFF ?]

8.7.1 Content dependent boundary filtering strength

For each boundary between neighbouring 4x4 luma blocks, a “Boundary Strength” Bs is assigned as shown in Figure 8-14. If Bs=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 boundary segment.

For each edge, if one of the neighbouring blocks is intra-coded, a relatively strong filtering (Bs=3) is applied. A special procedure with even stronger filtering might be applied on intra-coded macroblock boundaries (Bs=4). If neither of the blocks are intra-coded and at least one of them contains non-zero coefficients, medium filtering strength (Bs=2) is used. If none of the previous conditions are satisfied, filtering takes place with Bs=1 if at least one of the following conditions is satisfied: (a) prediction of the two blocks is formed using different reference frames or a different number of reference frames. (b) a pair of motion vectors from the two blocks is referencing the same frame and either component of this pair has a difference of more than one sample. Otherwise filtering is skipped for that particular edge (Bs=0).

Figure 8-14 – Flow chart for determining the boundary strength (Bs), for the block boundary between two neighbouring blocks p and q, where V_1(p,x), V_1(p,y) and V_2(p, x), V_2(p, y) are the horizontal and vertical components of the motion vectors of block p for the first and second reference frames or fields.

8.7.2 Thresholds for each block boundary

p3	p2	p1	p0	q0	q1	q2	q3

Figure 8-15 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

In the following description, the set of eight samples across a 4x4 block horizontal or vertical boundary is denoted as shown in Figure 8-15 with the actual boundary lying between p_0 and q_0. Uppercase letters indicate filtered samples and lower case letters indicate unfiltered samples with regard to the current edge filtering operation. However, p_1 and p_2 may indicate samples that have been modified by the filtering of a previous block edge.

Sets of samples across this edge are only filtered if the condition

$$Bs \neq 0 \;\;\&\&\;\; |p_0 - q_0| < \alpha \;\;\&\&\;\; |p_1 - p_0| < \beta \;\;\&\&\;\; |q_1 - q_0| < \beta \qquad (8\text{-}57)$$

Deleted: 61

Formatted: Font: Not Italic

is true. The values of the thresholds α and β are dependent on the average value of QP for the two blocks as well as on a pair of index offsets “Filter_Offset_A” and “Filter_Offset_B” that may be transmitted in the slice header for the purpose of modifying the characteristics of the filter. The average QP value for the two blocks is computed as $QP_{av} = (QP_p+QP_q)>>1$. The index used to access the α-Table (Table 8-3), as well as the C0-Table (Table 8-4) that is used in the default filter mode, is computed as:

Deleted: QP_{av}

Deleted: =

$$\mathrm{Index}_A = \mathrm{Clip3}(0, 51, QP_{av} + \mathrm{Filter\_Offset\_A}) \qquad (8\text{-}58)$$

Deleted: 62

NOTE - In SP and SI slices, QP_{av} is calculated in the same way as in other slice types. QS_Y from Equation 7-8 is not used in the deblocking filter.

The index used to access the β-Table (Table 8-3) is computed as:

$$\mathrm{Index}_B = \mathrm{Clip3}(\,0,\,51,\,QP_{av} + \mathrm{Filter\_Offset\_B}\,) \qquad (8\text{-}59)$$

If adaptive_block_size_transform_flag = = 1, Index_A and Index_B are calculated as specified in subclause 12.4.4.

The relationships between the indices (Equations 8-58 and 8-59) and the thresholds (α and β) are shown in Table 8-3.

Table 8-3 – QP_{av} and offset dependent threshold parameters 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 8-3 (concluded)

	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 of edges with Bs < 4

Two types of filtering are specified. In the default case with 0 < B < 4, Equations 8-60, 8-61 and 8-62 are used to filter p_0 and q_0:

$$\Delta = \mathrm{Clip3}(\,-C,\,C,\,(((q_0 - p_0) << 2 + (p_1 - q_1) + 4) >> 3)\,) \qquad (8\text{-}60)$$

$$P_0 = \mathrm{Clip1}(\,p_0 + \Delta\,) \qquad (8\text{-}61)$$

$$Q_0 = \mathrm{Clip1}(q_0 - \Delta) \qquad (8\text{-}62)$$

where C is determined as specified below.

The two intermediate threshold variables

$$a_p = |p_2 - p_0| \qquad (8\text{-}63)$$

$$a_q = |q_2 - q_0| \qquad (8\text{-}64)$$

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 luma edge, a filtered sample P_1 shall be produced as specified by

$$P_1 = p_1 + \mathrm{Clip3}(\,-C0,\,C0,\,(p_2 + (\,p_0 + q_0\,)>>1\, - (p_1<<1))\;>>\;1) \qquad (8\text{-}65)$$

If $a_q < \beta$ for a luma edge, a filtered sample Q_1 shall be produced as specified by

$$Q_1 = q_1 + \mathrm{Clip3}(\,-C0,\,C0,\,(q_2 + (\,p_0 + q_0\,)>>1\, - (q_1<<1)\,)\;>>\;1) \qquad (8\text{-}66)$$

where C0 is specified in Table 8-4. Chroma samples p_1 and q_1 are never filtered.

C is determined by setting it equal to C0 and then incrementing it by one if $a_p<\beta$, and again by one if $a_q<\beta$.

Deleted: 63
Deleted: 62
Deleted: 63
Deleted: 64
Deleted: 65
Deleted: 66
Deleted: 64
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 65
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 66
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 67
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 68
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 69
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Formatted: Font: Not Italic
Deleted: 70
Formatted: Font: Not Italic
Formatted: Font: Not Italic

Table 8-4 – Value of filter clipping parameter 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 8-4 (concluded)

	$Index_A$																									
	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 of edges with Bs = 4

When Bs is equal to 4 and the following condition holds:

$$a_p < \beta \;\; \&\& \; |p_0 - q_0| < ((\alpha >> 2) + 2) \qquad (8\text{-}67)$$

filtering of the left/upper side of the block edge is specified by Equations 8-68 and 8-69.

$$P_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3 \qquad (8\text{-}68)$$

$$P_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \qquad (8\text{-}69)$$

In the case of luma filtering, the filter in Equation 8-70 is also applied.

$$P_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3 \qquad (8\text{-}70)$$

Otherwise, if the condition of 8-67 does not hold, the filter in Equation 8-71 is applied.

$$P_0 = (2*p_1 + p_0 + q_1 + 2) >> 2 \qquad (8\text{-}71)$$

Similarly, for filtering of the right/lower side of the edge, if the following condition holds:

$$a_q < \beta \;\; \&\& \; |p_0 - q_0| < ((\alpha >> 2) + 2) \qquad (8\text{-}72)$$

filtering is specified by Equations 8-73 and 8-74.

$$Q_0 = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3 \qquad (8\text{-}73)$$

$$Q_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \qquad (8\text{-}74)$$

In the case of luma filtering, the filter in Equation 8-75 is also applied.

$$Q_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3 \qquad (8\text{-}75)$$

Otherwise, if the condition of 8-72 does not hold, the filter in Equation 8-76 is applied:

$$Q_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \qquad (8\text{-}76)$$

Formatted ... [10]
Deleted: 71
Formatted ... [11]
Deleted: 72
Deleted: 72 …73 ... [12]
Formatted ... [13]
Deleted: 73
Formatted ... [14]
Deleted: 74
Formatted: Font: Not Italic
Deleted: 74
Formatted ... [15]
Deleted: 75
Formatted: Font: Not Italic
Deleted: 71 …75 ... [16]
Formatted ... [17]
Deleted: …76 ... [18]
Formatted ... [19]
Deleted: 77
Formatted: Font: Not Italic
Deleted: 77 …78 ... [20]
Formatted ... [21]
Deleted: 78
Formatted: Font: Not Italic
Formatted ... [22]
Deleted: 79
Formatted: Font: Not Italic
Deleted: 79
Formatted ... [23]
Deleted: 80
Deleted: 76 …80 ... [24]

9 Entropy coding

Deleted: ¶

Formatted: Bullets and Numbering

9.1 Variable length coding

9.1.1 Exp-Golomb entropy coding

The Table of Exp-Golomb codewords is written in the following compressed form.

```
            1
          0 1 x0
        0 0 1 x1 x0
      0 0 0 1 x2 x1 x0
    0 0 0 0 1 x3 x2 x1 x0
    …..............
```

where xn take values 0 or 1. A codeword can be referred by its length in bits (L = 2n-1) and INFO = xn,…x1, x0. Notice that the number of bits in INFO is n-1 bits. The codewords are numbered from 0 and upwards. The definition of the numbering is:

Code_num = $2^{L/2}$ + INFO -1 (L/2 denotes division with truncation and INFO = 0 when L = 1). The first 10 code numbers and codewords are specified explicitly in Table 9-1. As an example, for the code number 5, L = 5 and INFO = 10 (binary) = 2 (decimal).

Table 9-1 – Code number and Exp-Golomb codewords in explicit form and used as ue(v)

Code_num	Code word
0	1
1	0 1 0
2	0 1 1
3	0 0 1 0 0
4	0 0 1 0 1
5	0 0 1 1 0
6	0 0 1 1 1
7	0 0 0 1 0 0 0
8	0 0 0 1 0 0 1
9	0 0 0 1 0 1 0

When L (L = 2N-1) and INFO is known, the regular structure of the Table makes it possible to create a codeword using the structure of the table. A decoder shall decode a codeword by reading in N bit prefix followed by N-1 INFO. L and INFO are then available. For each parameter to be coded, there is a conversion rule from the parameter value to the code number (or L and INFO).

Deleted: n

9.1.2 Unsigned Exp-Golomb entropy coding

The value of syntax elements that are represented by unsigned Exp-Golomb entropy coding directly corresponds to the code_num value of Table 9-1. This type of entropy coding is indicated via ue(v).

9.1.3 Signed Exp-Golomb entropy coding

The syntax elements that are represented by signed Exp-Golomb entropy coding are assigned to the code_num by ordering using their absolute values in increasing order and representing the positive value with the lower code_num. Table 9-2provides the assignment rule.

Table 9-2 – Assignment of symbol values and code_nums for signed Exp-Golomb entropy coding se(v)

Code number	**Symbol value**

0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
7	4
8	-4
9	5
10	-5
k	$(-1)^{k+1}$ Ceil(k÷2)

This type of entropy coding is denoted as se(v).

9.1.4 Mapped Exp-Golomb entropy coding

Table 9-3specifies the assignment of all mapped Exp-Golomb-coded slice data symbols. This type of entropy coding is indicated via me(v). These symbols are decoded differently when entropy_coding_mode = = 1.

If adaptive_block_size_transform_flag = = 1, additional syntax elements are mapped to the Exp-Golomb code as specified in Table 12-5.

Table 9-3 – Assignment of codeword number and parameter values for mapped Exp-Golomb-coded symbols

Code number	coded_block_pattern assignment to macroblock prediction types		Tcoeff_chroma_DC1		Tcoeff_chroma_AC1 Tcoeff_luma1 Zig-zag scan	
	Intra, SIntra	Pred, SPred	Level	Run	Level	Run
0	47	0	EOB	-	EOB	-
1	31	16	1	0	1	0
2	15	1	-1	0	-1	0
3	0	2	2	0	1	1
4	23	4	-2	0	-1	1
5	27	8	1	1	1	2
6	29	32	-1	1	-1	2
7	30	3	3	0	2	0
8	7	5	-3	0	-2	0
9	11	10	2	1	1	3
10	13	12	-2	1	-1	3
11	14	15	1	2	1	4
12	39	47	-1	2	-1	4
13	43	7	1	3	1	5

14	45	11	-1	3	-1	5
15	46	13	4	0	3	0
16	16	14	-4	0	-3	0
17	3	6	3	1	2	1
18	5	9	-3	1	-2	1
19	10	31	2	2	2	2
20	12	35	-2	2	-2	2
21	19	37	2	3	1	6
22	21	42	-2	3	-1	6
23	26	44	5	0	1	7
24	28	33	-5	0	-1	7
25	35	34	4	1	1	8
26	37	36	-4	1	-1	8
27	42	40	3	2	1	9
28	44	39	-3	2	-1	9
29	1	43	3	3	4	0
30	2	45	-3	3	-4	0
31	4	46	6	0	5	0
32	8	17	-6	0	-5	0
33	17	18	5	1	3	1
34	18	20	-5	1	-3	1
35	20	24	4	2	3	2
36	24	19	-4	2	-3	2
37	6	21	4	3	2	3
38	9	26	-4	3	-2	3
39	22	28	7	0	2	4
40	25	23	-7	0	-2	4
41	32	27	6	1	2	5
42	33	29	-6	1	-2	5
43	34	30	5	2	2	6
44	36	22	-5	2	-2	6
45	40	25	5	3	2	7
46	38	38	-5	3	-2	7
47	41	41	8	0	2	8
K	-	-	see below	see below	see below	see below

For the entries above the horizontal line, the Table is needed for relation between code number and Level/Run/EOB. For the remaining Level/Run combination there is a simple rule. The Level/Run combinations are assigned a code number according to the following priority: 1) sign of Level (+ -) 2) Run (ascending) 3) absolute value of Level (ascending).

9.1.5 Entropy coding for Intra

In intra mode, prediction is always used for each sub block in a macroblock.

9.1.5.1 Coding of Intra 4x4 and SIntra 4x4 prediction modes

The chosen intra-prediction mode (intra_pred_mode) of a 4x4 block is highly correlated with the prediction modes of adjacent blocks. This is illustrated in Figure 9-1. When the prediction modes of A and B are known (including the case that A or B or both are outside the slice) the most probable mode (most_probable_mode) of C is given. If one of the blocks A or B is “outside” the most probable mode is equal to prediction mode 2. Otherwise it is equal to the minimum of modes used for blocks A and B. When an adjacent block is coded by 16x16 intra mode, prediction mode is “mode 2: DC prediction”; when it is coded a non-intra macroblock, prediction mode is “mode 2: DC prediction” in the usual case and “outside” in the case of constrained intra update.

To signal prediction mode number for a 4x4 block first parameter use_most_probable_mode is transmitted. This parameter is represented by 1 bit codeword and can take values 0 or 1. If use_most_probable_mode is equal to 1 the most probable mode is used. Otherwise an additional parameter remaining_mode_selector, which can take value from 0 to 7 is sent as 3 bit codeword. The codeword is a binary representation of remaining_mode_selector value. The prediction mode number is calculated as:

```
if (remaining_mode_selector < most_probable_mode)
        intra_pred_mode = remaining_mode_selector;
else
         intra_pred_mode = remaining_mode_selector+1;
```

The ordering of prediction modes assigned to blocks C is therefore the most probable mode followed by the remaining modes in the ascending order.

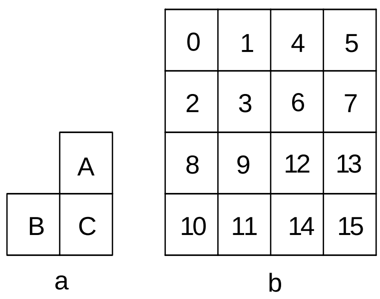


Figure 9-1 – a) Prediction mode of block C to be established, where A and B are adjacent blocks. b) order of intra prediction information in the bitstream

9.1.5.2 Coding of mode information for Intra-16x16 mode

Three numbers are specified at the end of the names of Intra-16x16 modes as specified in Table 7-10as a function of mb_type. The first of these numbers is termed *Imode* and ranges from 0 to 3, inclusive. The second is termed *nc* and contains the coded_block_pattern bits for chroma as specified in subclause 9.2.1.6. The third and final of these numbers is termed *ac_flag*. *ac_flag* equal to zero indicates that there are no AC coefficients in the 16x16 block. *ac_flag* equal to 1 indicates that at least one AC coefficient is present for the 16x16 block, requiring scanning of AC coefficient values for all 16 of the 4x4 blocks in the 16x16 block.

9.1.6 Context-based adaptive variable length coding (CAVLC) of transform coefficients

CAVLC (Context-based Adaptive VLC) is the method used for decoding of transform coefficients. The following coding elements are used:

1. If there are non-zero coefficients, it is typically observed that there is a string of coefficients at the highest frequencies that are ±1. The coding element coeff_token gives the total number of coefficients (from now referred to as TotalCoeffs) and also contains the number of "Trailing 1s" (from now referred to as T1s).

2. For T1s the sign is decoded from trailing_ones_sign and the level magnitude is 1**.**
3. For coefficients other than the T1s, level information is decoded from coeff_level.
4. The Run information is decoded. Since the number of coefficients is already known, this limits possible values for Run. Run is split into the Total number of zeros before all coefficients and Run before each non-zero coefficient, given by total_zeros and run_before.

Zig-zag scanning as described in subclause 9.4.1??? is used, but in the decoding of coefficient data, both levels and runs, the scanning is done in reverse order. Therefore the signs of T1s are decoded first (in reverse order), then the Level information of the last coefficient in the zig-zag scan order not included in the T1s, and so on. Run information is decoded similarly. First Total number of zeros in Runs is decoded, followed by Run before the last nonzero coefficient in the zig-zag scan order, and so on.

If adaptive_block_size_transform_flag = = 1, the VLC method for decoding 4x4, 4x8, 8x4, and 8x8 luma coefficient blocks is specified in subclause 12.5.1.

9.1.6.1 Entropy decoding of the number of coefficients and trailing ones: coeff_token

The syntax element coeff_token is decoded using the VLC specified in Table 9-4to Table 9-7.

Four VLC tables are used for combined decoding of number of coefficients and T1s, i.e. one codeword signals both parameters. VLCs are listed in the tables below. T1s is clipped to 3. Any remaining trailing 1s are decoded as normal levels. The variable TotalCoeff is the value returned by the function total_coeff( ); the variable T1s is the value returned by the function trailing_ones( ); and the variable NumCoeff is related to these quantities by TotalCoeff-T1s.

Table 9-4 – coeff_token: total_coeff( ) / trailing_ones( ): Num-VLC0

trailing_ones( )	0	1	2	3
total_coeff( )				
0	1	-	-	-
1	000101	01	-	-
2	00000111	000100	001	-
3	000000111	00000110	0000101	00011
4	0000000111	000000110	00000101	000011
5	00000000111	0000000110	000000101	0000100
6	0000000001111	00000000110	0000000101	00000100
7	0000000001011	0000000001110	00000000101	000000100
8	0000000001000	0000000001010	0000000001101	0000000100
9	00000000001111	00000000001110	0000000001001	00000000100
10	00000000001011	00000000001010	00000000001101	0000000001100
11	000000000001111	000000000001110	00000000001001	00000000001100
12	000000000001011	000000000001010	000000000001101	00000000001000
13	0000000000001111	000000000000001	000000000001001	000000000001100
14	0000000000001011	0000000000001110	0000000000001101	000000000001000
15	0000000000000111	0000000000001010	0000000000001001	0000000000001100
16	0000000000000100	0000000000000110	0000000000000101	0000000000001000

Table 9-5 – coeff_token: total_coeff( ) / trailing_ones( ): Num-VLC1

trailing_ones( )	0	1	2	3

total_coeff( )				
0	11	-	-	-
1	001011	10	-	-
2	000111	00111	011	-
3	0000111	001010	001001	0101
4	00000111	000110	000101	0100
5	00000100	0000110	0000101	00110
6	000000111	00000110	00000101	001000
7	00000001111	000000110	000000101	000100
8	00000001011	00000001110	00000001101	0000100
9	000000001111	00000001010	00000001001	000000100
10	000000001011	000000001110	000000001101	00000001100
11	000000001000	000000001010	000000001001	00000001000
12	0000000001111	0000000001110	0000000001101	000000001100
13	0000000001011	0000000001010	0000000001001	0000000001100
14	0000000000111	00000000001011	0000000000110	0000000001000
15	00000000001001	00000000001000	00000000001010	0000000000001
16	00000000000111	00000000000110	00000000000101	00000000000100

Table 9-6 – coeff_token: total_coeff( ) / trailing_ones( ): Num-VLC2

trailing_ones( )	0	1	2	3
total_coeff( )				
0	1111	-	-	-
1	001111	1110	-	-
2	001011	01111	1101	-
3	001000	01100	01110	1100
4	0001111	01010	01011	1011
5	0001011	01000	01001	1010
6	0001001	001110	001101	1001
7	0001000	001010	001001	1000
8	00001111	0001110	0001101	01101
9	00001011	00001110	0001010	001100
10	000001111	00001010	00001101	0001100
11	000001011	000001110	00001001	00001100
12	000001000	000001010	000001101	00001000
13	0000001101	000000111	000001001	000001100

14	0000001001	0000001100	0000001011	0000001010
15	0000000101	0000001000	0000000111	0000000110
16	0000000001	0000000100	0000000011	0000000010

Table 9-7 – coeff_token: total_coeff( ) / trailing_ones( ): Num-VLC_Chroma_DC

trailing_ones( )	0	1	2	3
total_coeff( )				
0	01	-	-	-
1	000111	1	-	-
2	000100	000110	001	-
3	000011	0000011	0000010	000101
4	000010	00000011	00000010	0000000

9.1.6.1.1 Table selection

For all elements, except chroma DC, a choice between three tables and one FLC is made. N is a value used for Table selection. Selection is done as follows: N is calculated using total_coeff() of the block above and to the left of the current block: N_U and N_L. In the Table below, X means that the block is available in the same slice. The block's coding mode is not taken into account when determining availability. When finding the block above and to the left for a block of Intra16x16 DC coefficients, the location of the block is assumed to be (0,0), i.e. the upper left corner of the macroblock.

Table 9-8 – Calculation of N for Num-VLCN

Upper block (N_U)	**Left block (N_L)**	**N**
X	X	$(N_L+N_U)/2$
X		N_U
	X	N_L
		0

0 <= N < 2 : Num-VLC0

2 <= N < 4 : Num-VLC1

4 <= N < 8 : Num-VLC2

N >= 8 : 6 bit FLC xxxxyy, as follows:

As a part of the coeff_token, TotalCoeff–1 is transmitted in the first 4 bits (xxxx) and T1s is transmitted as the last 2 bits (yy). There is one exception: the codeword 000011 represents TotalCoeff=0.

For chroma DC, Num-VLC_Chroma_DC is used.

9.1.6.2 Decoding of level information: coeff_level

First, the sign of T1s are decoded from 1 bit each of trailing_ones_sign. A value of zero indicates a positive level and a value of 1 indicates a negative level. A maximum of 3 bits are read.

For the remaining level information, seven structured VLCs are used to decode levels. The structured level tables are explained in Table 9-9 to Table 9-15. Lev-VLC0 has its own structure while the other tables, **Lev-VLC*N*, *N* = 1 to 6**, share a common structure.

Deleted: 9-15

Table 9-9 – Level tables

Lev-VLC0

Code no	Code	LevelCode (±1, ±2..)
0	1	1
1	01	-1
2	001	2
3	0001	-2
..	..	..
13	00000000000001	-7
14-29	000000000000001xxxx	±8 to ±15
30->	0000000000000001xxxxxxxxxxxx	±16 ->

For **Lev-VLC*N*, *N* = 1 to 6**, the following structure is used:

let level_code be the level information to be decoded from the VLC tables,

if (|level_code|-1) < (15<<(*N*-1)),

Code: 0….01x..xs,

where number of 0's = (|level_code|-1) >> (*N* -1),

number of x's = *N*-1,

value of x's = (|level_code|-1) % $2^{(N-1)}$,

s = sign bit (0 – positive, 1 – negative)

else,

28-bit escape code: 0000 0000 0000 0001 xxxx xxxx xxxs,

where value of x's = (|level_code|-1) - (15<<(*N* -1)),

s = sign bit (0 – positive, 1 – negative)

Table 9-10 – Level VLC1

Lev-VLC1		
Code no	Code	LevelCode (±1, ±2..)
0-1	1s	±1
2-3	01s	±2
..	..	..
28-29	000000000000001s	±15
30 ->	0000000000000001xxxxxxxxxxxs	±16 ->

Table 9-11 – Level VLC2

Lev-VLC2		
Code no	Code	LevelCode (±1, ±2..)
0-3	1xs	±1 to ±2
4-7	01xs	±3 to ±4
..	..s	..
56-59	0000000000000001xs	±29 to ±30
60 ->	00000000000000001xxxxxxxxxxxxs	±31 ->

Table 9-12 – Level VLC3

Lev-VLC3		
Code no	Code	LevelCode (±1, ±2..)
0-7	1xxs	±1 to ±4
8-16	01xxs	±5 to ±8
..	..	..
112-119	0000000000000001xxs	±57 to ±60
120 ->	00000000000000001xxxxxxxxxxxxs	±61 ->

Table 9-13 – Level VLC4

Lev-VLC4		
Code no	Code	LevelCode (±1, ±2..)
0-15	1xxxs	±1 to ±8
16-31	01xxxs	±9 to ±16
..	..	..
224-239	0000000000000001xxxs	±113 to ±120
240 ->	00000000000000001xxxxxxxxxxxxs	±121 ->

Table 9-14 – Level VLC5

Lev-VLC5		
Code no	Code	LevelCode (±1, ±2..)
0-31	1xxxxs	±1 to ±16
32-63	01xxxxs	±17 to ±32
..	..	..
448-479	000000000000001xxxxs	±225 to ±240
480 ->	0000000000000001xxxxxxxxxxxs	±241 ->

Deleted: xx

Table 9-15 – Level VLC6

Lev-VLC6		
Code no	Code	LevelCode (±1, ±2..)
0-63	1xxxxxs	±1 to ±32
64-127	01xxxxxs	±33 to ±64
..	..	..
896-959	000000000000001xxxxxs	±449 to ±480
960 ->	0000000000000001xxxxxxxxxxxs	±481 ->

Normally all coefficient levels (coeff_level) are equal to the decoded LevelCode value given in Table 9-9 to Table 9-15. However, when T1s is less than 3, the level of the first coefficient (after T1s) is equal to the decoded LevelCode plus 1:

if(first_coefficient && trailing_ones( ) < 3)
    coeff_level = (|level_code| + 1) * sign(level_code)

else
    coeff_level = level_code

The last two entries in Lev-VLC0 Table are escape codes. The first escape code with 19 bits, four "x"'s, is used to decode the 8 levels above the last regularly coded level. The next escape code with 28-bits, 12 "x"'s, is used to decode all remaining levels. For Lev-VLC1 to Lev-VLC6 tables, only the 28-bit escape code is used.

9.1.6.3 Table selection

Selections of the tables are changed during the decoding process based on number of coefficients, number of trailing ones, and the previously decoded level value (coeff_level).

Deleted: size of the

Let VLC denote the **Lev-VLC*N* (N=0-6) to be used.** After each level is decoded, the VLCN is updated according to the following method, where Level is the absolute value of the previously decoded level (coeff_level).

Deleted:

```
// VLC initialization for decoding first level
if (total_coeff(coeff_token) > 10 && trailing_ones(coeff_token)< 3)
   VLC = 1       // use Lev-VLC1 for first level
else
   VLC = 0       // use Lev-VLC0 for first level
// Assign FirstCoeff
// Decode level_code here and assign coeff_level and Level
// VLC update after decoding each level

vlc_inc table[6] = {0, 3, 6, 12, 24, 48}
```

```
if ((Level > vlc_inc[VLC]) && (VLC < 6))
   VLC ++
if (FirstCoeff_and Level > 3)
   VLC = 2
```

The first coefficient is always decoded with Lev-VLC0 or Lev-VLC1 while the rest of the coefficients are always decoded with Lev-VLC1 to Lev-VLC6.

The same procedure is used for chroma AC and DC coefficient levels.

9.1.6.4 Decoding of run information

Run decoding is separated in total number of Zeros (i.e. the number of zeros located before the last non-zero coefficient in the zig-zag scan) and Run (of zeros) before each coefficient.

9.1.6.4.1 Entropy Decoding of the total number of zeros: total_zeros

The variable TotalZeros as given by total_zeros**,** is the sum of all zeros located before the last non-zero coefficient in a zig-zag scan. For example, given the string of coefficients 0 0 3 0 0 4 0 0 0 0 2 0 1 0 0 0, TotalZeros will be 2+2+4+1=9. Since TotalCoeff is already known, it determines the maximum possible value of TotalZeros. One out of 15 VLC tables is chosen based on TotalCoeff.

If TotalCoeff indicates that all coefficients are non-zero, TotalZeros is not decoded since it is known to be zero

Table 9-16 – total_zeros tables for all 4x4 blocks

TotalCoeff ***TotalZeros***	**1**	**2**	**3**	**4**	**5**	**6**	**7**	
0	1	111	0101	00011	0101	000001	000001	
1	011	110	111	111	0100	00001	00001	
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	00011	0101	0011	101	101	100	11	
6	00010	0100	100	100	100	011	010	
7	000011	0011	011	0011	011	010	0001	
8	000010	0010	0010	011	0010	0001	001	
9	0000011	00011	00011	0010	00001	001	000000	
10	0000010	00010	00010	00010	0001	000000	-	
11	00000011	000011	000001	00001	00000	-	-	
12	00000010	000010	00001	00000	-	-	-	
13	000000011	000001	000000	-	-	-	-	
14	000000010	000000	-	-	-	-	-	
15	000000001	-	-	-	-	-	-	
TotalCoeff ***TotalZeros***	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	000001	000001	00001	0000	0000	000	00	0
1	0001	000000	00000	0001	0001	001	01	1

2	00001	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	00001	-	-	-	-	-	-
8	000000	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-

Table 9-17 – TotalZeros Table for chroma DC 2x2 blocks

NumCoeff TotalZeros	**1**	**2**	**3**
0	1	1	1
1	01	01	0
2	001	00	-
3	000	-	-

9.1.6.4.2 Run before each coefficient

At this stage it is known how many zeros are left to distribute (call this ZerosLeft). When decoding a run_before for the first time, ZerosLeft begins at TotalZeros, and decreases as more run_before elements are decoded.

For example, if there is only 1 zero left, the run before the next coefficient must be either of length 0 or 1, and only one bit is needed.

The number of preceding zeros before each non-zero coefficient (called RunBefore) needs to be decoded to properly locate that coefficient. Before decoding the next RunBefore, ZerosLeft is updated and used to select one out of 7 tables. RunBefore does not need to be decoded in the following two situations:

- If the total number of zeros has been reached (ZerosLeft = 0)
- For the last coefficient in the reverse zig-zag scan. Then the value is known to be ZerosLeft. This also means that the maximum value to be decoded is 14.

Table 9-18 – Tables for run_before

ZerosLeft Run Before	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 Context-based adaptive binary arithmetic coding (CABAC)

All syntax elements that are described as ae(v) in the syntax tables in subclauses 7.3.4 and 7.3.5 are decoded as specified in this subclause using the CABAC method if entropy_coding_mode = = 1 is indicated. Each syntax element is first mapped into a binary representation (binarization) as described in subclause 9.2.1. Each bin of the binary representation is decoded as specified subclauses 9.2.2, 9.2.3, and 9.2.4.

9.2.1 Decoding flow and binarization

A binarization scheme provides a mapping of a non-binary valued alphabet of symbols onto a set of sequences of binary decisions, so-called bins. In subclauses 9.2.1.1- 9.2.1.4the elementary types of binarization schemes for CABAC are specified.

A specification of the decoding flow and the assignment of binarization schemes for all syntax elements is given in subclauses 9.2.1.5 - 9.2.1.8.

9.2.1.1 Unary binarization

Table 9-19 shows the first five codewords of the unary code used for binarization of code symbols. For a code symbol *C* it consists of |*C*| ‘1’ bits followed by the last bit with value ‘0’. The first bin number corresponds to the first bit of the unary codeword with increasing bin numbers towards the last bit, as shown in Table 9-19.

Table 9-19 – Binarization by means of the unary code tree

Code symbol	Binarization						
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	
bin_num	1	2	3	4	5	6	

Formatted: Keep lines together

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

Formatted: Keep with next

9.2.1.2 Truncated unary (TU) binarization

The truncated unary (TU) binarization is specified for a finite alphabet $\{0,\ldots,C_{max}\}$ by applying the unary binarization of subclause 9.2.1.1to all code symbols *C* with $C < C_{max}$; the binarization of the symbol $C = C_{max}$ is specified by a code word consisting of C_{max} ‘1’s (without the last bit of value‘0’). Numbering of the bins is the same as for unary binarization.

9.2.1.3 Concatenated unary/ kth-order Exp-Golomb (UEGk) binarization

Concatenated unary/kth-order Exp-Golomb (UEGk) binarization consists of a concatenation of a prefix code word and a suffix code word. The prefix is formed by using a truncated unary binarization with $C_{max} = UCoff$, where *UCoff* denotes the cut-off parameter. For all code symbols *C* with $C < UCoff$, the suffix code word is void; for code symbols *C* with $C \geq UCoff$, the suffix consists of an Exp-Golomb code of order *k* for the symbol $C - UCoff$. For a given symbol *S* the Exp-Golomb code of order *k* is constructed as follows:

```
while( 1 ) {
    //first unary part of EGk
    if( symbol  >=  (unsigned int)( 1 << k ) )     {
       put( ‘1’ );
       S = S - ( 1<<k );
       k++;
    } else {
       put( ‘0’ );   //now terminating zero of unary part of EGk
       while( k-- )  //finally binary part of EGk
          put( ( S>>k )  &  0x01 );
       break;
    }
}
```

The numbering of bins is such that bin number bin_num = 1 relates to the first binary decision in the unary prefix code with increasing numbers towards the least significant bits of the binary part of the Exp-Golomb suffix.

9.2.1.4 Fixed-length (FL) binarization

Fixed-length (FL) binarization is applied to a finite alphabet $\{0,\ldots, C_{max}\}$ of code symbols. It is constructed by using a *L*-bit binary representation of a code symbol, where $L = \log_2\lfloor C_{max} \rfloor + 1$. The numbering of bins for the fixed-length binarization is such that the bin_num = 1 relates to the least significant bit with increasing bin numbers towards the most significant bit.

9.2.1.5 Binarization schemes for macroblock type and sub macroblock type

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-20. If adaptive_block_size_transform_flag = = 1, the binarization for decoding of macroblock type in I slices is specified in Table 12-10.

For macroblock types in SI slices the binarization consists of a prefix and a suffix part. The prefix consists of a single bit b_1 = ((mb_type = = SIntra_4x4) ? 0 : 1). The suffix part for mb_type SIntra_4x4 is void, while the suffix parts for all other macroblock types are given by the corresponding binarization pattern specified in Table 9-20.

The binarization schemes for P, SP, and B slices are specified in Table 9-21. The binarization for intra macroblock types in P and SP slices corresponding to mb_type values 7 to 30 consists of a prefix as specified in Table 9-21 and a suffix as specified in Table 9-20for the corresponding intra mb_type. For intra macroblock types in B slices (mb_type values 23 to 47) the binarization consists of a prefix as specified in Table 9-21 and a suffix as specified in Table 9-20 for the corresponding intra mb_type. If adaptive_block_size_transform_flag = = 1, the same prefix is used as specified in Table 9-21 for intra macroblock types of the corresponding slice type. However, the corresponding suffix parts are specified in Table 12-10.

For P, SP, and B slices the specification of the binarization for sub_mb_type is given in Table 9-22.

Table 9-20 – Binarization for macroblock types for I slices

Value (name) of mb_type	Binarization					
0 (Intra_4x4)	0					
1 (Intra_16x16_0_0_0)	1	0	0	0	0	
2 (Intra_16x16_1_0_0)	1	0	0	0	1	
3 (Intra_16x16_2_0_0)	1	0	0	1	0	
4 (Intra_16x16_3_0_0)	1	0	0	1	1	
5 (Intra_16x16_0_1_0)	1	0	1	0	0	0

6 (Intra_16x16_1_1_0)	1	0	1	0	0	1
7 (Intra_16x16_2_1_0)	1	0	1	0	1	0
8 (Intra_16x16_3_1_0)	1	0	1	0	1	1
9 (Intra_16x16_0_2_0)	1	0	1	1	0	0
10 (Intra_16x16_1_2_0)	1	0	1	1	0	1
11 (Intra_16x16_2_2_0)	1	0	1	1	1	0
12 (Intra_16x16_3_2_0)	1	0	1	1	1	1
13 (Intra_16x16_0_0_1)	1	1	0	0	0	
14 (Intra_16x16_1_0_1)	1	1	0	0	1	
15 (Intra_16x16_2_0_1)	1	1	0	1	0	
16 (Intra_16x16_3_0_1)	1	1	0	1	1	
17 (Intra_16x16_0_1_1)	1	1	1	0	0	0
18 (Intra_16x16_1_1_1)	1	1	1	0	0	1
19 (Intra_16x16_2_1_1)	1	1	1	0	1	0
20 (Intra_16x16_3_1_1)	1	1	1	0	1	1
21 (Intra_16x16_0_2_1)	1	1	1	1	0	0
22 (Intra_16x16_1_2_1)	1	1	1	1	0	1
23 (Intra_16x16_2_2_1)	1	1	1	1	1	0
24 (Intra_16x16_3_2_1)	1	1	1	1	1	1
bin_num	1	2	3	4	5	6

Table 9-21 – Binarization for macroblock types for P, SP, and B slices

Slice type	Value (name) of mb_type	Binarization						
P, SP slices	0 (Pred_L0_16x16)	0	0	0				
	1 (Pred_L0_L0_16x8)	0	1	1				
	2 (Pred_L0_L0_8x16)	0	1	0				
	4 (Pred_8x8)	0	0	1				
	6 (Pred_8x8ref0)	na						
	7 to 30 (Intra, prefix only)	1						
B slices	0 (Direct_16x16)	0						
	1 (Pred_L0_16x16)	1	0	0				
	2 (BiPred_L1_16x16)	1	0	1				
	3 (BiPred_Bi_16x16)	1	1	0	0	0	0	
	4 (Pred_L0_L0_16x8)	1	1	0	0	0	1	
	5 (Pred_L0_L0_8x16)	1	1	0	0	1	0	
	6 (BiPred_L1_L1_16x8)	1	1	0	0	1	1	
	7 (BiPred_L1_L1_8x16)	1	1	0	1	0	0	

	8 (BiPred_L0_L1_16x8)	1	1	0	1	0	1	
	9 (BiPred_L0_L1_8x16)	1	1	0	1	1	0	
	10 (BiPred_L1_L0_16x8)	1	1	0	1	1	1	
	11 (BiPred_L1_L0_8x16)	1	1	1	1	1	0	
	12 (BiPred_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (BiPred_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (BiPred_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (BiPred_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (BiPred_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (BiPred_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (BiPred_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (BiPred_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (BiPred_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (BiPred_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (BiPred_8x8)	1	1	1	1	1	1	
	23 to 47 (Intra, prefix only)	1	1	1	1	0	1	
bin_num		1	2	3	4	5	6	7

Table 9-22 – Binarization for sub macroblock types in P and B slices

Slice type	Value (name) of sub_mb_type	Binarization					
P slices	0 (Pred_L0_8x8)	1					
	1 (Pred_L0_8x4)	0	0	0			
	2 (Pred_L0_4x8)	0	0	1	1		
	3 (Pred_L0_4x4)	0	0	1	0		
	4 (Intra_8x8)	0	1				
B slices	0 (Direct_8x8)	0					
	1 (Pred_L0_8x8)	1	0	0			
	2 (BiPred_L1_8x8)	1	0	1			
	3 (BiPred_Bi_8x8)	1	1	0	0	0	
	4 (Pred_L0_8x4)	1	1	0	0	1	
	5 (Pred_L0_4x8)	1	1	0	1	0	
	6 (BiPred_L1_8x4)	1	1	0	1	1	
	7 (BiPred_L1_4x8)	1	1	1	0	0	0
	8 (BiPred_Bi_8x4)	1	1	1	0	0	1
	9 (BiPred_Bi_4x8)	1	1	1	0	1	0
	10 (Pred_L0_4x4)	1	1	1	0	1	1

	11 (BiPred_L1_4x4)	1	1	1	1	0	0
	12 (BiPred_Bi_4x4)	1	1	1	1	0	1
	13 (Intra_8x8)	1	1	1	1	1	
bin_num		1	2	3	4	5	6

9.2.1.6 Decoding flow and assignment of binarization schemes

In this subclause, the binarization schemes used for decoding of coded_block_pattern, delta_qp, the syntax elements of reference picture index, motion vector data, Intra_4x4 prediction modes are specified.

The coded block pattern information is decoded using the relationship as given in subclause 7.4.6???: coded_block_pattern = coded_block_patternY + 16∗nc. In a first step, the luma part coded_block_patternY of coded_block_pattern is decoded using the fixed-length (FL) binarization with $C_{max} = 15$ and L = 4. Then, the chroma part nc is decoded using TU binarization with $C_{max} = 2$.

For decoding of the delta_qp parameter the unsigned valued code number obtained by using the unary binarization is mapped to the signed value of the delta_qp parameter according to the relationship given in Table 9-2.

The decoding process for spatial intra prediction modes associated with luma of macroblock type Intra_4x4 and Sintra_4x4 is as follows. First, a parameter *intra_pred_indicator* is decoded using the truncated unary (TU) binarization with $C_{max} = 8$. If intra_pred_indicator = 0, use_most_probable_mode is set to 1. If intra_pred_indicator ≥ 1, remaining_mode_selector = intra_pred_indicator – 1. Given the parameters most_probable_mode and remaining_mode_selector, the intra prediction mode intra_pred_mode is obtained in the same way as specified in subclause 9.1.5.1. The decoding order of the prediction modes is the same as shown in Figure 9-1 b). For decoding of the spatial intra prediction mode for chroma intra_chroma_pred_mode, the truncated unary (TU) binarization with $C_{max} = 3$ is used.

The reference picture index parameter is decoded using the unary binarization as given in subclause 9.2.1.1.

Each component of the motion vector data is decoded separately starting with the horizontal (h) component. First the absolute value *abs_mvd_comp* and then the sign *sign_mvd_comp* of each component (*comp=h,v*) shall be decoded. The binarization scheme applied to *abs_mvd_comp* is given by the concatenated unary/3rd-order Exp-Golomb (UEG3) binarization with cut-off parameter Ucoff = 9. Decoding of the sign information *sign_mvd_comp* of the motion vector components is performed as follows. In a first step, a binary indicator *sign_ind* is decoded. Then the sign information sign_info is recovered by *sign_info = ((sign_ind = = 0) ? 1 : -1).*Note that for decoding of the sign information as well as for decoding of the Exp-Golomb suffix the decoder bypass Decode_eq_prob as specified in subclause 9.2.4.3.5??? is used.

9.2.1.7 Decoding flow and binarization of transform coefficients

Decoding of transform coefficients is a three-step process. First, the one-bit coded_block_flag is decoded for each block of transform coefficients unless the coded_block_pattern symbol on macroblock level indicates that the regarded block has no non-zero coefficients. If the coded_block_flag symbol is zero, no further information has to be decoded for the block. Otherwise, it is indicated that there are significant coefficients inside the block. The latter case implies that, in a second decoding step, for each scanning position i except the last position in a block the binary-valued significant_coeff_flag[i] has to be decoded. If significant_coeff_flag[i] has the value of one, the corresponding position i in the block contains a significant coefficient and a further binary-valued last_significant_coeff_flag[i] is decoded. If last_significant_coeff_flag[i] is zero, there is at least one further significant coefficient to be decoded; otherwise, the last significant coefficient along the scanning path is reached. If this is the case, the absolute value minus 1 coeff_absolute_value_minus_1 and then the sign of the coefficient coeff_sign is decoded for each significant transform coefficient by traversing the block in reverse scanning order. coeff_absolute_value_minus_1 is decoded using the concatenated unary/zero-order Exp-Golomb (UEG0) binarization with UCoff=14. Similar to the decoding of absolute values of the motion vector components, the Exp-Golomb suffix is decoded by using the decoder bypass Decode_eq_prob. . For decoding of coeff_sign, the decoder bypass shall be used as well.

9.2.1.8 Decoding of sign information related to motion vector data and transform coefficients

Decoding of the sign information *sign_mvd_comp* of the motion vector components and *coeff_sign* of the levels corresponding to significant transform coefficients is performed as follows. Using the decoder bypass Decode_eq_prob as specified in subclause 9.2.4.3.5??? first a binary indicator *sign_ind* is decoded. Then the sign information sign_info is recovered by *sign_info = ((sign_ind = = 0) ? 1 : -1).*

9.2.1.9 Decoding of macroblock skip flag and end-of-slice flag

Decoding of the mb_skip_flag is as follows: First, a binary-valued *mb_skip_flag_decoded* is decoded using the context model as specified in subclause 9.2.2.2. In a second step, the actual value of mb_skip_flag is obtained by inverting mb_skip_flag_decoded, *i.e.*, mb_skip_flag = mb_skip_flag_decoded ^ 0x01.

The end_of_slice_flag is decoded using a fixed, non-adaptive model by chosing State = 63 and MPS = 0. The following mechanism guarantees a fixed model, although the coding engine uses a probability estimator after each decoding step as further specified in subclause 9.2.4.2. By observing a sequence of end_of_slice values ‘0’ meaning that the end of a slice has not been reached, the initial chosen state will not be altered, since for the observation of a MPS symbol the state variable State = 63 will be mapped onto itself in the probability estimation. However, as soon as a LPS value of ‘1’ is decoded for end_of_slice_flag, the probability estimation for the LPS will not affect the subsequent decoding process, because the end of a slice is reached, and all context models are refreshed using the initial states.

9.2.2 Context definition and assignment

For each bin number, a *context variable* is specified by a conditioning term containing prior decoded symbols or parts thereof. The possible numerical values of a context variable specify the different *context models* associated with a specific bin number. Typically, there are several possible values or *context labels* for each bin number bin_num. However, in some cases the context variable may simply be a constant label, in which case there is only one fixed context model.

This subclause specifies first a variety of generic types of context variables, so-called *context templates*, for conditional coding of syntax elements. Then, for each bin number of a syntax element, the specification of the corresponding context variable is given. For the different bin numbers associated with the binarization of a given syntax element or parts thereof, a unique *context identifier* context_id is chosen such that the context variable associated to bin number k is given by context_id[k]. Since there is always a maximum bin number N with a context variable context_id[N] that is different from the corresponding context variable context_id[N−1] for the preceding bin number N−1, it is sufficient to specify a context identifier for each index k with 1≤ k≤ N, where max_idx_ctx_id = N is called the maximum index of the context identifier context_id.

Deleted: defines

Table 9-23provides an overview of the context identifiers associated to each category of syntax elements. A detailed description of the corresponding context variables is given in the subsequent subclauses. Note that each context identifier corresponds to a unique range of context labels, which, in the case of macroblock type, may overlap for the different slice types I, SI, P, SP, and B.

If adaptive_block_size_transform_flag = = 1, the context identifiers related to decoding of transform coefficients utilize an additional set of ranges as further specified in Table 12-12.

Table 9-23 – Syntax elements and associated context identifiers

Syntax element	Context identifier	Type of Binarization	max_idx_ctx_id	Range of context label
mb_skip_flag	ctx_mb_skip	-/-	1	0 – 2
mb_type	ctx_mb_type_I	Table 9-20	6	0 – 7
	ctx_mb_type_SI_pref	-/-	1	0 – 2
	ctx_mb_type_SI_suf	Table 9-20	6	3 – 10
	ctx_mb_type_P	Table 9-21	3	3 – 6
	ctx_mb_type_B		4	3 – 8
	ctx_mb_type_P_suf	Table 9-20	5	6 – 9
	ctx_mb_type_B_suf		5	8 – 11
	ctx_b8_mode_P	Table 9-22	4	12 – 15
	ctx_b8_mode_B		4	12 – 15
mvd_l0 and mvd_l1	ctx_abs_mvd_h	UEG3, UCoff=9	5	16 – 22

	ctx_abs_mvd_v	UEG3, UCoff=9	5	23 – 29
ref_idx_l0 and ref_idx_l1	ctx_ref_idx	Unary	3	30 – 35
delta_qp	ctx_delta_qp	Unary	3	36 – 39
[Ed. Note: The Table corruption was here in prior versions – missing end of cell markers]				
chroma_pred_mode	ctx_ipred_chroma	TU, C_{max}=3	3	40 – 44
intra_pred_mode	ctx_ipred_luma	TU, C_{max}=8	2	45 – 62
coded_block_pattern	ctx_cbp_luma	FL, C_{max}=15	4	63 – 66
	ctx_cbp_chroma	TU, C_{max}=2	2	67 – 74
coded_block_flag	ctx_cbp4	-/-	-/-	75 – 94
significant_coeff	ctx_sig	-/-	-/-	95 – 155
last_coeff	ctx_last	-/-	-/-	156 – 216
coeff_absolute_value_minus_1	ctx_abs_level	UEG0, UCoff=14	2	217 – 266
end_of_slice_flag	ctx_eos	Fixed, non-adaptive model with State(ctx_eos) = 63, MPS(ctx_eos) = 0		

9.2.2.1 Overview of assignment of context labels

Table 9-24and Table 9-25contain all context identifiers along with their corresponding range of context labels. The association of context labels (modulo some offset) and bin numbers shows which context variable uses a fixed model and which one implies a choice of different models. The latter are characterized by those entries where a set of different context labels are given for a specific bin number bin_num (Table 9-24) or block type dependent context_category (Table 9-25). These context variables are specified in the following subclauses.

Table 9-24 – Overview of context identifiers and associated context labels

Context identifier	Range of context label	Offset for context label	max_idx_ctx_id	bin_num					
				1	2	3	4	5	≥ 6
ctx_mb_skip	0 – 2	0	1	0,1,2	-/-	-/-	-/-	-/-	-/-
ctx_mb_type_I	0 – 7	0	6	0,1,2	3	4	5,6	6,7	7
ctx_mb_type_SI_pref	0 – 2	0	1	0,1,2	-/-	-/-	-/-	-/-	-/-
ctx_mb_type_SI_suf	3 – 10	3	6	0,1,2	3	4	5,6	6,7	7
ctx_mb_type_P	3 – 6	3	3	0	1	2,3	-/-	-/-	-/-
ctx_mb_type_P_suf	6 – 9	6	5	0	1	2	2,3	3	3
ctx_mb_type_B	3 – 8	3	4	0,1,2	3	4,5	5	5	5
ctx_mb_type_B_suf	8 – 11	8	5	0	1	2	2,3	3	3
ctx_b8_mode_P	12 – 15	12	4	0	1	2	3	-/-	-/-
ctx_b8_mode_B	12 – 15	12	4	0	1	2,3	3	3	3
ctx_abs_mvd_h	16 – 22	16	5	0,1,2	3	4	5	6	6
ctx_abs_mvd_v	23 – 29	23	5	0,1,2	3	4	5	6	6
ctx_ref_idx	30 – 35	30	3	0,1,2,3	4	5	5	5	5
ctx_delta_qp	36 – 39	36	3	0,1	2	3	3	3	3

ctx_ipred_chroma	40 – 44	36	3	0,1,2	3	4	-/-	-/-	-/-
ctx_ipred_luma	45 – 62	42	2	0,...,8	9,...,17	9,...,17	9,...,17	9,...,17	9,...,17
ctx_cbp_luma	63 – 66	60	4	0,1,2,3	0,1,2,3	0,1,2,3	0,1,2,3	-/-	-/-
ctx_cbp_chroma	67 – 74	64	2	0,1,2,3	4,5,6,7	-/-	-/-	-/-	-/-
ctx_abs_level	217 – 266	217+ 10*context_category	2	0,...,4	5,...,9	5,...,9	5,...,9	5,...,9	5,...,9

Table 9-25 – Overview of context identifiers and associated context labels (continued)

Context identifier	**Offset (range) of context label**	**context_category (as specified in Table 9-28)**				
		0	**1**	**2**	**3**	**4**
ctx_coded_block	75 (75 – 94)	0 – 3	4 – 7	8 – 11	12 – 15	16 – 19
ctx_sig_coeff	95 (95 – 155)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60
ctx_last_coeff	156 (156 – 216)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60

Formatted: Font: Bold

Deleted: Table 9-28

Formatted: Font: Bold, Check spelling and grammar

Formatted: Font: Bold, Check spelling and grammar

Formatted: Font: Bold

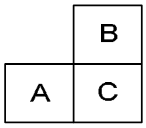


Figure 9-2 – Illustration of the generic context template using two neighbouring symbols A and B for conditional coding of a current symbol C

9.2.2.2 Context templates using two neighbouring symbols

The generic design of this type of context variable is shown in Figure 9-2. It involves two previously decoded symbols or bins of the same syntax element that correspond to the spatially neighbouring blocks to the left (A) and on the top (B) of the regarded block C. The generic form of the equation defining this type of context is given by

$$ctx\_var\_spat = cond\_term(A, B), \qquad (9\text{-}1)$$

where the conditioning term *cond_term(A, B)* describes the functional relationship between the spatially neighbouring symbols *A* and *B*, on the one hand, and the values of the context variable, on the other hand. Three special cases of this template are specified as follows:

$$ctx\_var\_spat1 = cond\_term(A) + cond\_term(B), \qquad (9\text{-}2)$$

$$ctx\_var\_spat2 = cond\_term(A) + 2{*}cond\_term(B), \qquad (9\text{-}3)$$

$$ctx\_var\_spat3 = cond\_term(A). \qquad (9\text{-}4)$$

Table 9-26 – Specification of context variables using context templates according to Equations (9-2) – (9-4)

Context variable	**Context template**	**cond_term(X), semantics of X**	**cond_term(X), if X not available**
ctx_mb_skip	*ctx_var_spat1*	(mb_skip_flag(X) = = 0) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_I[1]	*ctx_var_spat1*	(mb_type(X) ! = Intra_4x4) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_SI_pref[1]	*ctx_var_spat1*	(mb_type(X) ! = Sintra_4x4) ? 1: 0 X: neighbouring macroblock	0

ctx_mb_type_SI_suf[1]	*ctx_var_spat1*	(mb_type(X) != Intra_4x4) ? 1: 0 X: neighbouring macroblock	0
ctx_mb_type_B[1]	*ctx_var_spat1*	((mb_type(X) != Direct) ? 1: 0) X: neighbouring macroblock	0
ctx_ipred_chroma[1]	*ctx_var_spat1*	((intra_chroma_pred_mode(X) != 0) ? 1: 0) X: neighbouring macroblock	0
ctx_ref_idx[1]	*ctx_var_spat2*	(ref_idx_l0/ref_idx_l1(X) ! = 0) ? 1: 0 X: neighbouring block	0
ctx_ipred_luma[i], i=1,2	*ctx_var_spat3*	9*(i–1) + intra_pred_mode(X) X: neighbouring block	0
ctx_cbp_luma[i], i=1,…,4	*ctx_var_spat2*	i-th bit of coded_block_patternY(X) X: neighbouring 8x8 block of i-th block	0
ctx_cbp_chroma[1]	*ctx_var_spat2*	(nc(X) != 0) ? 1: 0 X: neighbouring macroblock	0
ctx_cbp_chroma[2]	*ctx_var_spat2*	4 + (nc(X) == 2) ? 1: 0 X: neighbouring macroblock	0
ctx_coded_block	*ctx_var_spat2*	coded_block_flag(X) X: neighbouring block of same block type	1, if X is INTRA; 0, otherwise
ctx_delta_qp	*ctx_var_spat3*	(delta_qp(X) != 0) ? 1: 0 X: neighbouring macroblock	0

Table 9-26contains the specification of context variables relying on templates using two neighboring symbols. Note that the conditioning term of the context variable *ctx_coded_block* depends on six block types as given in Table 9-28(Luma-DC, Luma-AC, Chroma-U-DC, Chroma-V-DC, Chroma-U-AC, Chroma-V-AC).

The specification of the context variables ctx_abs_mvd_h[1] and ctx_abs_mvd_v[1] is given as follows.

$$ctx\_abs\_mvd\_comp[1] = \begin{cases} 0, (|mvd\_comp(A)| + |mvd\_comp(B)|) < 3, \\ 2, (|mvd\_comp(A)| + |mvd\_comp(B)|) > 32, \\ 1, otherwise, \end{cases} \tag{9-5}$$

where *comp* denotes the component h (horizontal) or v (vertical) and *A*, *B* denote the neighbouring blocks of the block to decode as shown in Figure 9-2. Since the neighbouring blocks *A* and *B* may belong to different macroblock partitions, the following principle for identifying the proper neighbouring blocks that are used in Equation (9-5) is established. First, the motion vector data is assumed to be given in oversampled form such that each 4x4 block has its own *motion vector (MV)*. That means, on the one hand, that in case of a neighbouring block having a coarser partition, the related 4x4 sub-blocks are assumed to inherit the *MV* from the corresponding parent block(s) in the quadtree partition. On the other hand, if the current block *C* represents a larger block than a 4x4 block, it is assumed to be represented by the corresponding leftmost 4x4 sub-block in the top row of 4x4 sub-blocks. Then, given a block *C*, the neighbouring 4x4 sub-blocks *B* on top and *A* to the left of the representing 4x4 sub-block of *C* are chosen for evaluation of Equation (9-5).

9.2.2.3 Context templates using preceding bin values

Let (b_1, …, b_N) denote the binarization of a given symbol C. Then, a generic type of context variable associated with the k-th bin of C is specified as

$$ctx\_var\_bin[k] = cond\_term(b_1,\ldots,b_{k-1}), \tag{9-6}$$

where 1<k≤N. In Table 9-27, the specification of context variables using this type of context template is given.

Table 9-27 – Definition of context variables using the context template according to Equation (9-6)

Context variable	cond_term($b_1,\ldots,b_{k-1}$)
ctx_mb_type_I[4]	(b_3 != 0) ? 5: 6
ctx_mb_type_I[5]	(b_3 != 0) ? 6: 7

ctx_mb_type_SI_suf[4]	(b_3 ! = 0) ? 5: 6
ctx_mb_type_SI_suf[5]	(b_3 ! = 0) ? 6: 7
ctx_mb_type_P[3]	(b_2 ! = 0) ? 2: 3
ctx_mb_type_P_suf[4]	(b_3 ! = 0) ? 2: 3
ctx_mb_type_B[3]	(b_2 ! = 1) ? 4: 5
ctx_mb_type_B_suf[4]	(b_3 ! = 0) ? 2: 3
ctx_b8_mode_B[3]	(b_2 = = 1) ? 2: 3

9.2.2.4 Additional context definitions for information related to transform coefficients

Three different additional context identifiers are used for conditioning of information related to transform coefficients. All these three types depend on context categories of different block types denoted by the variable *context_category*. The definition of these context categories is given in Table 9-28.

Table 9-28 – Context categories for the different block types

block_type	MaxNumCoeff	context_category
Luma DC block for INTRA16x16 mode	16	0:Luma-Intra16-DC
Luma AC block for INTRA16x16 mode	15	1:Luma-Intra16-AC
Luma block for INTRA 4x4 mode	16	2:Luma-4x4
Luma block for INTER 4x4 mode	16	
U-Chroma DC block for INTRA mode	4	3:Chroma-DC
V-Chroma DC block for INTRA mode	4	
U-Chroma DC block for INTER mode	4	
V-Chroma DC block for INTER mode	4	
U-Chroma AC block for INTRA mode	15	4:Chroma-AC
V-Chroma AC block for INTRA mode	15	
U-Chroma AC block for INTER mode	15	
V-Chroma AC block for INTER mode	15	

Additional context categories are used in the case of adaptive_block_size_transform_flag = = 1 as specified in subclause 12.5.2. The context identifiers ctx_sig_coeff and ctx_last_coeff are related to significant_coeff_flag and last_significant_coeff_flag, respectively; the related context variables include an additional dependency on the scanning position *scanning_pos* within the regarded block:

$$ctx\_sig\_coeff[scanning\_pos] = Map\_sig(\ scanning\_pos), \tag{9-7}$$

$$ctx\_last\_coeff[scanning\_pos] = Map\_last(scanning\_pos). \tag{9-8}$$

The specifiaction of Map_sig and Map_last in Equations (9-7) and (9-8) depends on the block type. For context_category 0 – 4 the corresponding maps are given by the identity, i.e.,

$$Map\_sig(scanning\_pos) = Map\_last(scanning\_pos) = scanning\_pos,\ if\ context\_category = 0,\ldots,4,$$

where scanning_pos denotes the position related to the zig-zag scan. For the additional context categories 5 – 7, which are only in use if adaptive_block_size_transform_flag = = 1, the specification of Map_sig and Map_last is given in subclause 12.5.2.

For decoding of coeff_absolute_value_minus_1, the context identifier ctx_abs_level consisting of the two context variables ctx_abs_level[1] and ctx_abs_level[2] is used, which are specified as follows:

$$ctx\_abs\_lev[1] = ((num\_decod\_abs\_lev\_gt1 \,!= 0)\ ?\ 4 : min(3, num\_decod\_abs\_lev\_eq1)), \tag{9-9}$$

$$ctx\_abs\_lev[2] = min(4,\ num\_decod\_abs\_lev\_gt1), \tag{9-10}$$

where *num_decod_abs_lev_eq1* denotes the accumulated number of decoded coefficients with absolute value equal to 1, and *num_decod_abs_lev_gt1* 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 takes place, which means that no additional information outside the regarded transform coefficient block is used for the context variables ctx_abs_level[k], k=1,2.

9.2.3 Initialisation of context models

9.2.3.1 Initialisation procedure

At the beginning of each slice, each context model is initialised with an initial state, which consists of a state number and the meaning of the most probable symbol (MPS) as further described in subclause 9.2.4.2. The actual initial state of a context model depends linearly on the (initial) quantization parameter QP of the given slice, such that for each context model a pair of Table entries {m, n} is given, from which the initial state is computed in the following way

1. Compute pre_state = ((m∗(QP−12))>>4) + n;
2. Limit pre_state to the range of [0,101] for P- and B-slices and to the range of [27,74] for I-slices, i.e., pre_state = min (101, max(0,pre_state)) for P- and B-slices and pre_state = min (74, max(27,pre_state)) for I-slices;
3. Map pre_state to {state, MPS} pair according to the following rule: if (pre_state <= 50) then {state = 50-pre_state, MPS = 0} else {state = pre_state-51, MPS = 1}

9.2.3.2 Initialisation procedure

Table 9-29 – Table 9-34contain the initialisation parameters related to the context variables of all syntax elements, from which the initial states can be obtained by using the rules given in subclause 9.2.3.1.

Table 9-29 – Initialisation parameters for context identifiers *ctx_mb_type_I, ctx_mb_type_SI_pref, , ctx_mb_type_SI_suf,ctx_mb_skip, ctx_mb_type_P, ctx_mb_type_B*

Context label	ctx_mb_type_I		ctx_mb_type_SI_pref		ctx_mb_skip					
	m	n	m	n	m	n	m	n	m	n
0	7	25	7	25	-23	66				
1	8	35	8	35	-14	77				
2	-2	63	-2	63	-9	88				
			ctx_mb_type_SI_suf				ctx_mb_type_P		ctx_mb_type_B	
3	-9	68	7	25			2	13	9	49
4	-15	74	8	35			14	24	3	65
5	-3	36	-2	63			-21	69	0	78
6	-1	51	-9	68			-1	52	-13	81
7	0	50	-15	74					-14	73
8			-3	36					-8	64
9			-1	51						
10			0	50						

Table 9-30 – Initialisation parameters for context identifiers *ctx_b8_mode_P, ctx_b8_mode_B, ctx_mb_type_P_suf, ctx_mb_type_B_suf*

Context label	ctx_mb_type_P_suf ctx_mb_type_B_suf		Context label	ctx_b8_mode_P		ctx_b8_mode_B	
	m	n		m	n	m	n
9	-9	55	12	8	46	-9	62
10	-7	50	13	12	11	-12	66
11	2	47	14	-4	62	-9	56
			15	18	48	3	47

Table 9-31 – Initialisation parameters for context identifiers *ctx_abs_mvd_h, ctx_abs_mvd_v, ctx_ref_idx*

Context label	ctx_abs_mvd_h		Context label	ctx_abs_mvd_v		Context label	ctx_ref_idx	
	m	n		m	n		m	n
16	1	48	23	-1	45	30	3	27
17	-5	60	24	-5	59	31	-1	47
18	-8	70	25	-9	71	32	0	45
19	2	52	26	0	50	33	-2	60
20	2	62	27	3	61	34	-1	57
21	-3	64	28	-3	63	35	0	48
22	-2	80	29	1	80			

Table 9-32 – Initialisation parameters for context identifiers *ctx_delta_qp, ctx_ipred_chroma, ctx_ipred_luma*

Context label	ctx_delta_qp		Context label	ctx_ipred_luma		Context label	ctx_ipred_luma	
	m	n		m	n		m	n
36	0	28	45	-5	49	54	-4	61
37	0	50	46	-3	58	55	-6	64
38	0	50	47	-5	58	56	-6	63
39	0	50	48	-4	58	57	-3	75
	ctx_ipred_chroma		49	-5	59	58	-4	63
40	-5	50	50	-4	60	59	-7	65
41	0	50	51	-6	61	60	-1	63
42	0	50	52	-5	62	61	-18	66
43	0	50	53	-4	60	62	-9	67
44	0	50						

Table 9-33 – Initialisation parameters for context identifiers *ctx_cbp_luma, ctx_cbp_chroma*

Context label	ctx_cbp_luma		Context label	ctx_cbp_chroma		Context label	ctx_cbp_chroma	
	I slices	P, B slices		I slices	P, B slices		I slices	P, B slices

	m	n	m	n		m	n	m	n		m	n	m	n
63	-3	75	-21	81	67	-7	65	-23	58	71	-18	66	-11	46
64	-4	63	-15	60	68	-1	63	-18	64	72	-9	67	-6	56
65	-4	70	-14	61	69	-9	77	-16	63	73	-13	70	-8	59
66	-5	56	-15	47	70	-4	76	-18	73	74	-7	74	-18	74

Table 9-34 – Initialisation parameters for context identifiers *ctx_coded_block*, *ctx_sig_coeff*, *ctx_last_coeff*, *ctx_abs_level* for context category 0 – 4

Context label	Context category 0		Context label	Context category 1		Context label	Context category 2				Context label	Context category 3		Context label	Context category 4	
							I slices		P, B slices							
	m	n		m	n		m	n	m	n		m	n		m	n
ctx_coded_block																
75	-4	72	79	-4	37	83	-2	52	-4	57	87	0	55	91	-3	41
76	-4	68	80	-3	50	84	-7	68	-9	66	88	-3	70	92	-4	62
77	-6	75	81	-6	49	85	-5	61	-7	64	89	-4	68	93	-6	58
78	-6	75	82	-3	61	86	-8	77	-17	80	90	-4	75	94	-8	73
ctx_sig_coeff																
95	-6	68				124	-7	67	4	46	139	-3	71			
96	-11	66	110	0	44	125	-11	64	1	43	140	-9	69	142	-6	60
97	-5	63	111	2	53	126	-8	66	2	45	141	0	70	143	0	63
98	-5	56	112	0	49	127	-11	63	-4	46				144	-3	54
99	2	43	113	1	43	128	-9	63	-1	45				145	-4	54
100	1	47	114	4	45	129	-8	60	3	43				146	4	52
101	-8	58	115	-2	40	130	-11	55	-6	44				147	-5	44
102	-3	46	116	-1	45	131	-10	61	1	45				148	-1	48
103	4	38	117	0	50	132	-7	63	2	46				149	-7	57
104	0	58	118	2	55	133	-7	60	0	46				150	11	51
105	1	51	119	-7	52	134	-16	61	-12	49				151	-13	51
106	-7	57	120	-2	57	135	-2	62	3	50				152	7	55
107	-1	53	121	7	50	136	-1	58	11	46				153	5	57
108	2	47	122	2	52	137	-8	61	4	50				154	2	51
109	-1	59	123	4	66	138	-3	68	9	64				155	4	68
ctx_last_coeff																
156	0	5				185	11	25	16	27	200	12	27			
157	1	1	171	4	42	186	9	24	21	19	201	12	28	203	10	28
158	2	2	172	5	46	187	12	24	20	23	202	16	38	204	14	30

159	3	6	173	9	40	188	14	23	21	22				205	17	30
160	4	3	174	7	41	189	13	23	21	23				206	20	30
161	5	4	175	6	46	190	16	22	24	23				207	15	37
162	6	4	176	10	40	191	19	20	25	24				208	21	39
163	7	3	177	14	33	192	18	21	23	27				209	22	33
164	8	4	178	10	43	193	21	21	25	29				210	21	39
165	9	5	179	12	48	194	23	25	23	35				211	15	52
166	10	9	180	13	39	195	20	23	19	36				212	8	49
167	11	2	181	13	41	196	24	25	21	40				213	13	52
168	12	3	182	16	43	197	25	29	23	45				214	8	60
169	13	1	183	21	35	198	24	33	15	53				215	15	56
170	14	6	184	11	55	199	14	53	8	70				216	3	71
ctx_abs_level																
217	-5	55	227	-10	57	237	-10	63	-6	51	247	-9	70	257	-7	58
218	-3	36	228	-1	30	238	-5	37	-5	24	248	-14	55	258	0	33
219	-1	35	229	0	32	239	-7	43	-7	32	249	-10	57	259	-1	40
220	-2	40	230	-1	35	240	-6	46	-4	34	250	-5	56	260	-2	45
221	-6	50	231	0	40	241	-5	49	-5	39	251	-4	57	261	-3	49
222	-3	44	232	-5	39	242	-8	50	-12	43	252	-14	63	262	-7	48
223	-4	51	233	-4	47	243	-7	56	-7	50	253	-11	67	263	-9	58
224	-3	53	234	-9	55	244	-9	62	0	48	254	-5	68	264	-16	66
225	-4	55	235	-6	58	245	-9	64	-3	53	255	-9	71	265	-12	65
226	-11	63	236	-4	56	246	-11	70	-8	60	256	0	50	266	-12	68

9.2.4 Table-based arithmetic coding

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 interval with range R will be subdivided into two sub-intervals having range p(‘0’)×R and R–p(‘0’)×R, 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 model *CTX* is specified by the probability p_{LPS} of the LPS and the value of MPS, 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 $\{P_k \mid 0 \le k < 64\}$ for the LPS probability p_{LPS}.
- The range R representing the state of the coding engine is quantized to a small set $\{Q_1,\ldots,Q_4\}$ of pre-set quantization values prior to the calculation of the new interval range. Storing a table containing all 64×4 pre-computed product values of $Q_i \times P_k$ allows a multiplication-free approximation of the product $R \times P_k$.
- For syntax elements or parts thereof with an approximately uniform probability distribution a separate simplified encoding and decoding path is used.

9.2.4.1 Probability estimation

The probability estimator is realized by a set of representative probabilities $\{ P_k \mid 0 \le k < 64 \}$ for the LPS together with some appropriately specified state transition rules.Table 9-35shows the transition rules for adapting to a given MPS or LPS decision. For transition from one state to another each state is only addressed by its index *State*, which will be appropriately changed to a new index *Next_State_MPS(State)* or *Next_State_LPS(State)* after the decoding of a *MPS* or *LPS* symbol, respectively.

The numbering of the states is arranged in such a way that the state with index *State*=0 corresponds to a LPS probability value of 0.5, with decreasing LPS probability towards higher states. However, for I-slices it is of advantage to restrict the number of states to the first 24 state indices. Therefore, Table 9-35contains a separate column containing the transition rule *Next_State_MPS_INTRA* that is used for decoding the syntax elements of an I-slice only. Note, that *Next_State_MPS_INTRA* differs from *Next_State_MPS* only by one entry.

NOTE - To prevent the probability estimator from switching to states higher than *State*=23, we set *Next_State_MPS(23)*= 23 for I-slice decoding. For the clarity of presentation, a separate table entry for I-slice decoding is shown in Table 9-35.

After encoding or decoding a decision, an update of the probability estimate is obtained by switching the state index *State* to a new index, such that for I-slice coding

```
if(decision = = MPS)
        State ← Next_State_MPS_INTRA(State)
else
        State ← Next_State_LPS(State)
```

and all other slice types

```
if(decision = = MPS)
        State ← Next_State_MPS(State)
else
        State ← Next_State_LPS(State).
```

In the case, where the current state corresponds to a probability value of 0.5, which corresponds to the *State* index of 0, and a LPS symbol is observed, the sense of MPS and LPS has to be interchanged.

Table 9-35 – Probability transition

State	Next_State_MPS_INTRA	Next_State_MPS	Next_State_LPS	State	Next_State_MPS	Next_State_LPS
0	1	1	0	32	33	22
1	2	2	0	33	34	22
2	3	3	1	34	35	23
3	4	4	2	35	36	23
4	5	5	2	36	37	24
5	6	6	3	37	38	24
6	7	7	4	38	39	25
7	8	8	5	39	40	25
8	9	9	6	40	41	26
9	10	10	7	41	42	26
10	11	11	8	42	43	27
11	12	12	8	43	44	27
12	13	13	10	44	45	28
13	14	14	10	45	46	28
14	15	15	10	46	47	29
15	16	16	11	47	48	29
16	17	17	12	48	49	30
17	18	18	13	49	50	30
18	19	19	14	50	51	30
19	20	20	14	51	52	31
20	21	21	14	52	53	32

21	22	22	14	53	54	33
22	23	23	15	54	55	33
23	23	24	16	55	56	34
24	-/-	25	17	56	57	34
25	-/-	26	18	57	58	35
26	-/-	27	19	58	59	35
27	-/-	28	19	59	60	36
28	-/-	29	20	60	61	37
29	-/-	30	20	61	62	37
30	-/-	31	21	62	63	38
31	-/-	32	21	63	63	38

Table 9-36 – RTAB[State][Q] Table for interval subdivision

State	0	1	2	3	State	0	1	2	3
0	9216	11264	13312	15360	32	896	1152	1344	1536
1	8832	10816	12800	14720	33	896	1088	1280	1472
2	8512	10368	12288	14144	34	832	1024	1216	1408
3	8128	9920	11712	13504	35	832	960	1152	1344
4	7680	9344	11072	12736	36	768	960	1088	1280
5	7168	8768	10368	11968	37	768	896	1088	1216
6	6912	8448	9984	11520	38	704	896	1024	1152
7	6336	7808	9216	10624	39	704	832	960	1152
8	5888	7232	8512	9856	40	640	832	960	1088
9	5440	6656	7872	9088	41	640	768	896	1088
10	5120	6208	7360	8512	42	640	768	896	1024
11	4608	5632	6656	7680	43	576	704	832	960
12	4224	5184	6144	7104	44	576	704	832	960
13	3968	4800	5696	6592	45	576	704	832	960
14	3712	4480	5312	6144	46	512	640	768	896
15	3456	4224	4992	5760	47	512	640	768	896
16	3072	3776	4416	5120	48	512	640	768	832
17	2816	3456	4096	4736	49	512	640	704	832
18	2624	3200	3776	4416	50	512	576	704	832
19	2432	3008	3520	4096	51	448	576	704	768
20	2304	2816	3328	3840	52	448	576	640	768
21	2048	2496	2944	3392	53	448	512	640	704

22	1856	2240	2688	3072	54	448	512	640	704
23	1664	2048	2432	2816	55	448	512	576	704
24	1536	1856	2240	2560	56	384	512	576	704
25	1408	1728	2048	2368	57	384	512	576	640
26	1344	1600	1920	2176	58	384	448	576	640
27	1216	1472	1792	2048	59	384	448	576	640
28	1152	1408	1664	1920	60	384	448	512	640
29	1088	1344	1536	1792	61	384	448	512	640
30	1024	1280	1472	1728	62	384	448	512	576
31	960	1216	1408	1600	63	384	448	512	576

9.2.4.2 Description of the arithmetic decoding engine

The status of the arithmetic decoding engine is represented by a value *V* pointing into the code sub-interval and the corresponding range *R* of that sub-interval. Figure 9-3 gives an illustration of the whole decoding process. Performing the InitDecoder procedure, which is further specified in subclause 9.2.4.3.1???, *V* and *R* will be appropriately initialised. For decoding of each single decision *S*, the following two-step operation is employed: First, the related context model *CTX* is determined according to the rules specified in subclauses 9.2.2. Given the context model *CTX*, the decoding operation Decode(*CTX*) then delivers the decoded symbol *S* as is described in detail in subclause 9.2.4.3.2???.

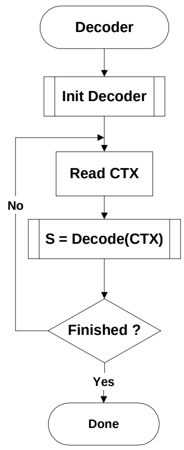


Figure 9-3 – Overview of the Decoding Process

9.2.4.2.1 Initialisation of the decoding engine

In the initialisation procedure of the decoder, as illustrated in Figure 9-4, *V* is first filled with two bytes of the compressed data using the GetByte routine as specified in subclause 9.2.4.3.4???, and then the range *R* is set to 0x8000.

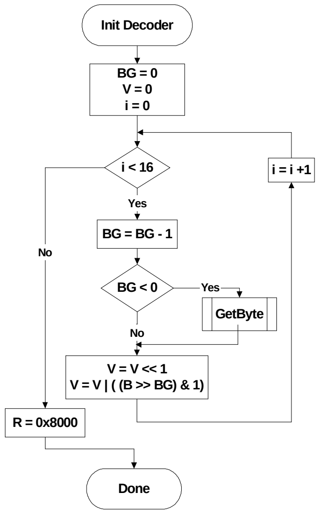


Figure 9-4 – Flowchart of initialisation of the decoding engine

[Ed. Note: this Figure is modified to reflect a bug-fix. In the previous versions of the draft the consumption of bits of each byte of compressed data was described in such a way that the least significant bit of the current byte B to be processed were moved first into the V register. This however was not conforming to the spec of SODB, where the leftmost bit (MSB) is considered to be the first bit and the rightmost (LSB) to be the last in terms of the order of consumption. Hence, the description is changed such that data bits are read in MSB to LSB order.]

9.2.4.2.2 Decoding a decision

Figure 9-5 shows the flowchart for decoding a single decision. In a first step, the estimation of the sub-interval ranges R_{LPS} and $R - R_{LPS}$ corresponding to the LPS and the MPS decision is performed as follows.

Given the interval range *R*, we first map *R* to a quantized value *Q* using

$$Q=(R\text{-}0x4001)>>12, \tag{9-11}$$

such that the state index *State* and *Q* are used as an entry in the look-up table RTAB to determine R_{LPS}:

$$R_{LPS}=RTAB[State][Q]. \tag{9-12}$$

Table 9-36specifies the corresponding values of RTAB in 16-bit representation. The tabulated values are actually given in 8-bit accuracy; the maximum value of RTAB corresponds to 14 bits and all values have been left-shifted by 6 bits for a better access in a 16-bit architecture.

In a second step, the current value of *V* is compared to the size of the MPS sub-interval $R \leftarrow R - R_{LPS}$. If *V* is greater than or equal to *R* a LPS is decoded, *V* is decremented by *R* and the new range *R* is set to R_{LPS}; otherwise a MPS is decoded. Given the decoded decision, the probability update is performed accordingly as specified in subclause 9.2.4.2. Depending on the current value of the new range *R*, renormalization will be performed as described in more detail in subclause 9.2.4.3.3???.

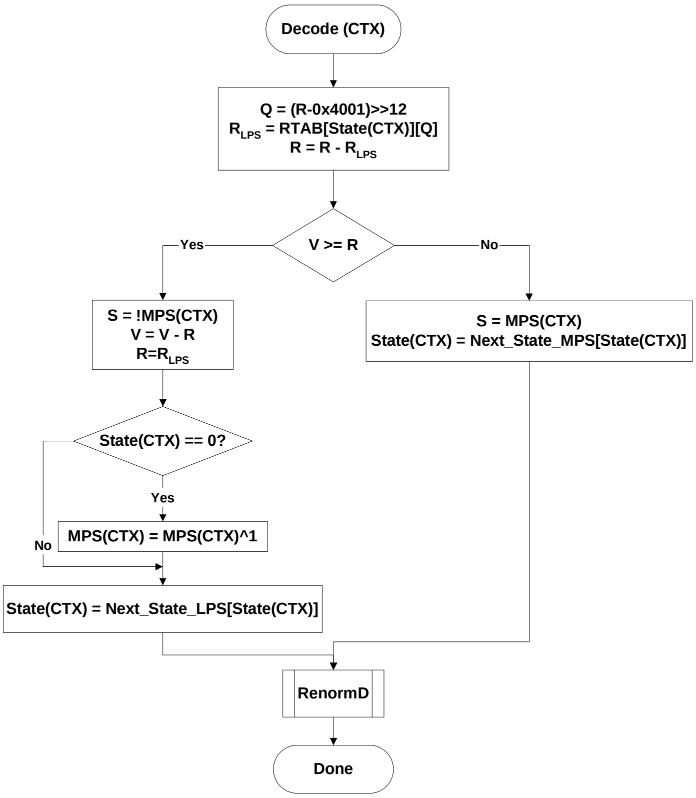


Figure 9-5 – Flowchart for decoding a decision

9.2.4.2.3 Renormalization in the decoding engine (RenormD)

Renormalization is illustrated in Figure 9-6. The current range *R* is first logically compared to 0x4000: If it is greater than that value, no renormalization is needed and RenormD is finished; otherwise the renormalization loop is entered. Within this loop, the range *R* is doubled, i.e. left-shifted by 1 and the bit-counter *BG* is decremented by 1. In the case, that the condition *BG*<0 holds, a new byte of compressed is inserted into *B* by calling the GetByte routine. Finally, the next bit of *B* is shifted into *V*.

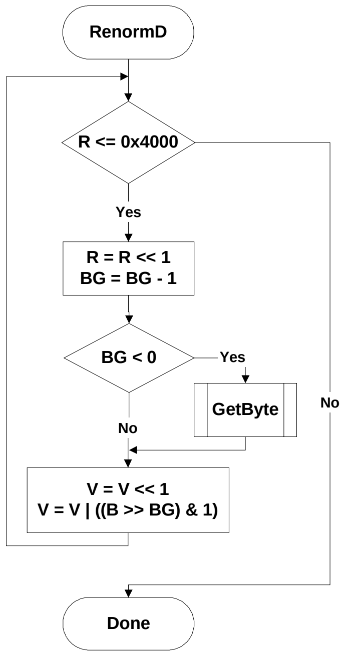


Figure 9-6 – Flowchart of renormalization

[Ed. Note: this Figure is modified to reflect a bug-fix. In the previous versions of the draft the consumption of bits of each byte of compressed data was described in such a way that the least significant bit of the current byte B to be processed were moved first into the V register. This however was not conforming to the spec of SODB, where the leftmost bit (MSB) is considered to be the first bit and the rightmost (LSB) to be the last in terms of the order of consumption. Hence, the description is changed such that data bits are read in MSB to LSB order.]

9.2.4.2.4 Input of compressed bytes (GetByte)

Figure 9-7 shows how the input of compressed data is performed. At the initialsation stage of the whole decoding process or in case a renormalization occurs and the bit-counter *BG* has a negative value, this procedure will be invoked. First, a new byte of compressed data is read out of the bitstream *C*; then the index *CL* pointing to the current position of the bitstream *C* is incremented by 1 and the bit-counter is set to 7.

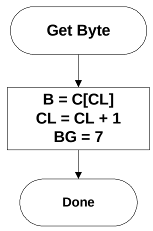


Figure 9-7 – Flowchart for Input of Compressed Bytes

9.2.4.2.5 Decoder bypass for decisions with uniform pdf (Decode_eq_prob)

This special decoding routine applies to decoding of the sign information of motion vector data and the sign of the levels of significant transform coefficients, which are assumed to have a uniform probability distribution. Consequently omitting the probability estimation in this special case reduces the decoding process to a single comparison ($V>=R_{half}$?)

in order to determine the right subinterval and its corresponding decoded symbol value *S*. The subsequent renormalization process is similar to that performed in the renormalization procedure RenormD, as depicted in Figure 9-6 with two modifications. Firstly, the rescaling operation *R←(R<<1)* is unnecessary and secondly, the initial comparison (*R<=0x4000?)* can be omitted.

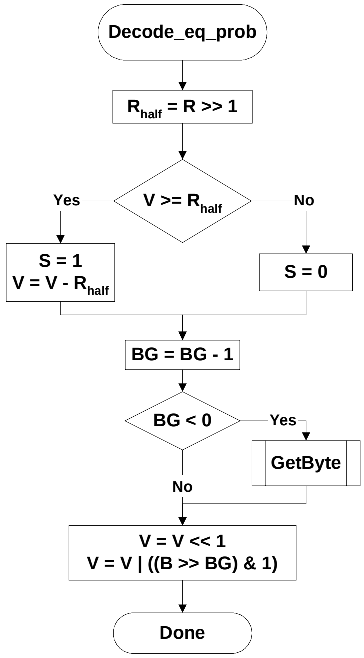


Figure 9-8 – Flowchart of decoding bypass

[Ed. Note: this Figure is modified to reflect a bug-fix. In the previous versions of the draft the consumption of bits of each byte of compressed data was described in such a way that the least significant bit of the current byte B to be processed were moved first into the V register. This however was not conforming to the spec of SODB, where the leftmost bit (MSB) is considered to be the first bit and the rightmost (LSB) to be the last in terms of the order of consumption. Hence, the description is changed such that data bits are read in MSB to LSB order.]

10 Decoding process for B slices

10.1 Introduction

The use of B (bi-predictive) slices is indicated in the nal_unit_type. A B slice is an inter predicted slice. The major difference between a B slice and P slice is that B slices are coded in a manner in which some macroblocks or blocks may use a weighted average of two distinct inter prediction values for building the prediction signal. Generally, B slices utilize two distinct reference index lists. Each of these index lists refer to pictures in the reference picture buffer.

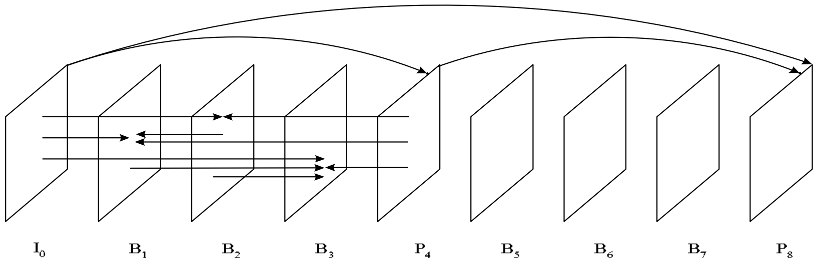


Figure 10-1 – Illustration of B picture concept

NOTE - The location of pictures in the bitstream is in a decoding order. Pictures that are dependent on other pictures shall occur in the bitstream after the pictures on which they depend. Figure 10-1 shows one hypothetical example, where three B pictures are inserted in output order between an I and a P picture and the subscripts indicate the output order. The P picture P_4 only depends on the first Intra picture I_0. The B picture B_2, which is temporally located between I_0 and P_4, depends on both of these pictures. The B picture B_1 depends on I_0, P_4, and B_2; the B picture B_3 additionally depends on B_1. While the output order for this example is given by I_0, B_1, B_2, B_3, P_4, the decoding order is I_0, P_4, B_2, B_1, B_3.

10.2 Decoding process for macroblock types and sub macroblock types

There are five different prediction modes supported by B pictures. They are the list 0, list 1, bi-predictive, direct, and intra prediction modes. In bi-predictive mode, the prediction signal is formed by a weighted average of list 0 and list 1 prediction values. The direct mode can result in prediction modes list 0, list 1, or bi-predictive. Prediction using the direct mode is derived from a combination of the motion vectors and macroblock type used in the co-located macroblock of the first picture (the picture at index 0) in list 1.

Coded macroblocks in B pictures utilize a similar tree-structured macroblock partitioning to P pictures. Depending on the number of elements in the two reference picture lists, up to two reference picture indices are coded for each bi-predicted region. Additionally, for each luma block of 16x16, 16x8, 8x16 samples and the associated chroma blocks, and each sub macroblock, the prediction mode (list 0, list 1, bi-predictive, direct, intra) can be chosen separately. In order to avoid a separate codeword to specify the prediction mode, the indication of the prediction direction is incorporated into the codewords for macroblock type and sub macroblock type, respectively, as shown in the Table 7-12and Table 7-17. A sub macroblock of a B picture macroblock can also be coded in direct mode.

When mb_adaptive_frame_field_flag = = 1, the current direct-mode macroblock shall follow the same frame/field coding mode as its co-located macroblock.

10.3 Decoding process for motion vectors

10.3.1 Differential motion vectors

In the following, the terms *temporal ordering* and *temporal distance* refer to ordering and distance according to the picture order counter described in subclause 8.3.2. This is also the decoder output order and the intended display order.

Motion vectors for list 0, list 1, or bi-predictive blocks are differentially encoded. A prediction has to be added to the motion vector differences in order to reconstruct motion vectors for the current macroblock.

As a special case of bi-predictive blocks, if the two reference pictures used for the bi-prediction both occur temporally earlier or both occur temporally later than the current picture being decoded, then the motion vector decoding is performed as described in subclause 10.3.2.

Otherwise, the predictions for bi-predictive blocks are formed from the motion vectors of spatially neighbouring blocks in a way similar to that described in subclause 8.4.1, but with a few important distinctions.

First, a list i, where i = 0 or 1, motion vector MV_i from the current block is predicted only from neighbouring blocks that contain motion vectors with the same temporal direction (earlier or later in time) as MV_i. If a neighbouring block does not have a motion vector with the same temporal direction, the predictor for that block is set to zero and the neighbouring block shall be considered as belonging to a *different reference picture* for purposes of computing the median prediction of subclause 8.4.1.1.

Second, if a neighbouring bi-predicted block has both motion vectors pointing in the same temporal direction as MV_i, and both motion vectors point to the same reference picture, then the list 0 motion vector from that block is used as a prediction. Otherwise, if a neighbouring bi-predicted block has both motion vectors pointing in the same temporal direction as MV_i but they point to different reference pictures, then the motion vector that points to the temporally closest reference picture is used.

Third, reconstructed motion vectors in direct mode neighbouring blocks shall be used as predictions for the current block motion vectors.

If a direct mode neighbouring block has two motion vectors, then this block is treated as if it were a bi-predictive neighbouring block.

If a direct mode neighbouring block has only one motion vector, then this block is considered as a list 0 or list 1 block.

In the case that the co-located block in such a direct mode block is intra coded and the direct_spatial_mv_pred_flag is 0, the direct mode block is treated as belonging to a *different reference picture* for purposes of computing the median prediction of subclause 8.4.1.1.

10.3.2 Motion vector decoding with scaled MV

If both reference pictures (ref_idx_l0 & ref_idx_l1) occur temporally earlier or both occur temporally later than the current picture being decoded, then the following process is followed for decoding the motion vectors MV_1 and MV_2 for bi-predictive modes. The motion vector decoding process is illustrated in Figure 10-2. The motion vector MV_1 for the first reference picture (ref_idx_l0) is differentially decoded using motion vector prediction as described in 10.3.1. However, the method used for decoding the motion vector MV_2 for the second reference picture (ref_idx_l1) is as follows:

The scaled motion vector (*smv*) is first calculated from the motion vector MV_1 as:

$$Z = (TD_2 \times 256) / TD_1 \qquad smv = (Z \times MV_1 + 128) >> 8$$

where TD_1 is the temporal distance between the current picture and the reference picture indicated by ref_idx_l0, and TD_2 is the temporal distance between the current picture and the reference picture indicated by ref_idx_l1.

Then, MV_2 is differentially coded with respect to *smv*.

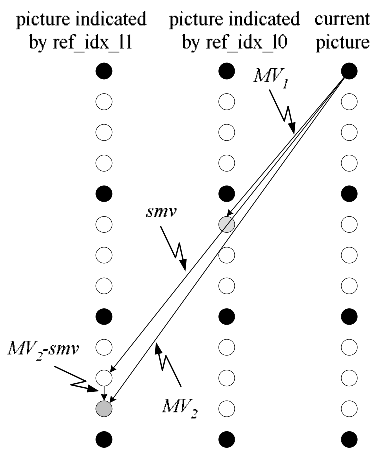


Figure 10-2 – Differential motion vector decoding with scaled motion vector

10.3.3 Motion vectors in direct mode

The direct_spatial_mv_pred_flag identifies for the current slice whether the direct mode motion vectors are calculated using a spatial or temporal technique. If this indicator is set to 1 then the spatial technique is used. Otherwise, if this indicator is set to 0, then direct mode motion vectors are calculated using the temporal technique.

10.3.3.1 Spatial technique of obtaining the direct mode motion parameters

The first step in the spatial technique for direct mode prediction is the determination of a candidate reference picture index for each list (list 0 and list 1).

The reference picture indices for the neighbouring blocks A, B, C, and D within the current slice for the 16x16 current luma block E, as described in subclause 8.4.1.1and shown in Figure 8-4, shall be used to determine a preliminary candidate reference picture index for each list. The preliminary candidate reference picture index for each list (list 0 and list 1) shall be the minimum reference picture index among the reference picture indices used from the same list for the prediction or bi-prediction of the neighbouring blocks. If no neighbouring blocks are present within the current slice that use prediction from the same list, either for prediction or bi-prediction, the preliminary candidate reference index for that list shall be interpreted as not existing.

If a preliminary candidate reference index exists for either list, decision of the final candidate reference indices and associated motion vector values for that list for each 4x4 block of the current macroblock depends on the coded parameters of the co-located 4x4 blocks in the first picture in list 1. If the first picture in list 1 is a short-term picture and if all lines of the co-located 4x4 block were predicted using list 0 prediction with reference picture index 0 and motion vector components in the range of -1 to 1 inclusive, then the final candidate reference picture index for the list of the current 4x4 block shall be 0 and shall be associated with a candidate motion vector value of (0, 0). Otherwise, the final candidate reference picture index for the list shall be the preliminary candidate reference picture index for the list and the associated candidate motion vector shall be obtained, using the 16x16 block motion vector prediction, as described in subclause 8.4.1by using the final reference picture index.

If both candidate reference picture indices exist, then the block is predicted as a bi-prediction block using the final candidate reference picture index and motion vector for each list. Otherwise, if a candidate reference picture index exists for only one of the two lists, the block shall be predicted by single-list prediction using the final candidate reference picture index and associated motion vector for the existing candidate. Finally, if neither candidate reference picture index exists, bi-prediction shall be used with reference picture index zero and associated motion vector (0, 0) for both lists.

10.3.3.2 Temporal technique of obtaining the direct mode motion parameters

In the temporal technique direct mode, the same block structure as for the co-located macroblock of the first picture (the picture at index 0) in list 1 is used. For each block of the current macroblock, the list 0 and list 1 motion vectors are computed as scaled versions of the list 0 motion vector of the co-located block in the list 1 reference picture as described below.

The list 0 reference picture for the direct mode current block is the same as the list 0 reference picture used for the co-located block in the list 1 reference picture. The list 0 and list 1 motion vectors for direct mode macroblocks are calculated differently depending on whether picture_struct and the reference indicate fields or frames. Also if the list 1 co-located macroblock is an intra-coded block, the motion vectors are set to zero, and the list 0 reference picture for the direct mode is the most recent temporally preceding stored picture.

With possible adaptive switching of frame/field coding at picture level, a B frame or its list 1 reference frame can be coded in either frame structure or field structure. Hence, there are four possible combinations of frame or field coding for a pair of a macroblock in the current B picture and its co-located macroblock in the list 1 reference picture. Calculations of the two motion vectors in direct mode are slightly different for the four cases.

Case 1: Both the current macroblock and its co-located in the list 1 reference picture are in frame mode, as shown in Figure 10-3. The list 0 reference for each block within the current macroblock is the same as the list 0 reference of the co-located block in the list 1 reference picture. Two motion vectors (MV_F, MV_B) are calculated by

$$Z = (TD_B \times 256) / TD_D \qquad MV_F = (Z \times MV + 128) >> 8$$
$$W = Z - 256 \qquad MV_B = (W \times MV + 128) >> 8$$

where TD_B is the temporal distance between the current B frame and the list 0 reference frame, and TD_D is the temporal distance between the list 1 reference frame and the list 0 reference frame (see Figure 10-3).

In the case that the co-located block in the list 1 reference frame has a list 0 motion vector pointing to a long-term frame, the list 0 and list 1 motion vectors for the direct mode current block are calculated by

$$MV_F = MV$$
$$MV_B = 0$$

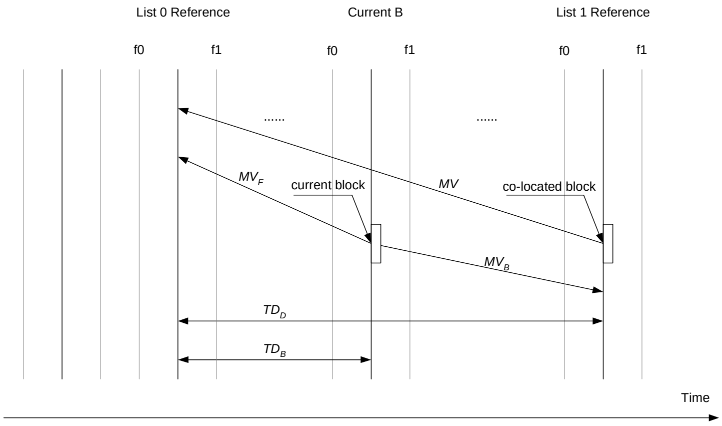


Figure 10-3 – Both the current block and its co-located block in the list 1 reference picture are in frame mode (f0 and f1 indicate the corresponding fields)

Case 2: Both the current macroblock and its co-located macroblock in the list 1 reference picture are in field mode. Two motion vectors for each block within the current macroblock are calculated from the list 0 motion vector of the co-located block in the list 1 reference field of the same parity.

For field 0, the list 0 motion vector of the co-located block will always point to one of the previously coded list 0 fields, as shown in Figure 10-4. The list 0 reference field for the direct mode block will be the same as the list 0 field of the co-located list 1 block, and the list 1 reference field will be field 0 of the list 1 reference frame. The list 0 and list 1 motion vectors ($MV_{F,i}$, $MV_{B,i}$) for the direct mode block are calculated as follows:

$$Z = TD_{B,i} \times 256/\ TD_{D,i} \qquad MV_{F,i} = (Z \times MV_i + 128) >> 8$$

$$W = \mathrm{Z} - 256 \qquad MV_{B,i} = (W \times MV_i + 128) >> 8$$

where the subscript i is the field index (0 for the 1st field and 1 for the 2nd field), and MV_i is the list 0 motion vector of the co-located block in field i of the list 1 reference frame. $TD_{B,i}$ is the temporal distance between the current B field and the list 0 reference field. $TD_{D,i}$ is the temporal distance between the list 1 reference field and the list 0 reference field.

For field 1, the list 0 motion vector of the co-located list 1 block may point to one of the temporally previous coded fields, in which case calculation of the list 0 and list 1 motion vectors follows the same equations as above.

However, it is also possible that the list 0 motion vector of the co-located block in field 1 of the list 1 reference frame points to field 0 of the same frame, as shown in Figure 10-5. In this case, the two motion vectors for field 1 of the direct mode current block are calculated as follows:

$$Z = -TD_{B,1} \times 256/\ TD_{D,1} \qquad MV_{F,1} = (\mathrm{Z} \times MV_1 + 128) >> 8$$

$$W = Z - 256 \qquad MV_{B,1} = (W \times MV_1 + 128) >> 8$$

Note that both motion vectors are now pointing to field 0 and field 1 of the list 1 reference frame respectively.

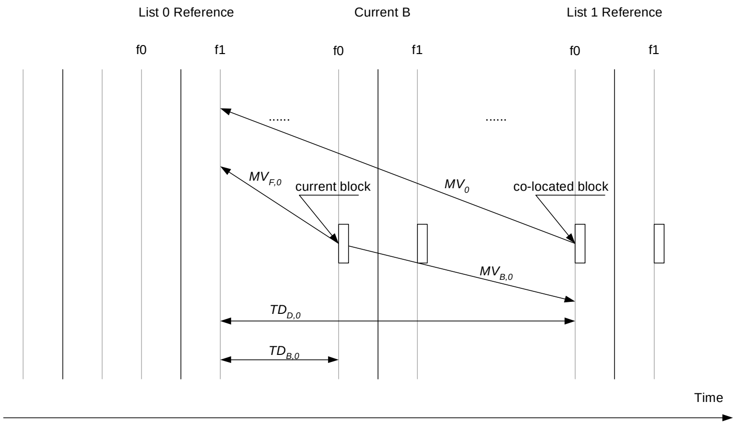


Figure 10-4 – Both the current macroblock and its co-located macroblock in the temporally subsequent picture are in field mode.

In the case that the co-located block in the list 1 reference field has a list 0 motion vector pointing to a long-term field, the list 0 and list 1 motion vectors for the direct mode are calculated by

$$MV_{F,i} = MV_i$$

$$MV_{B,i} = 0$$

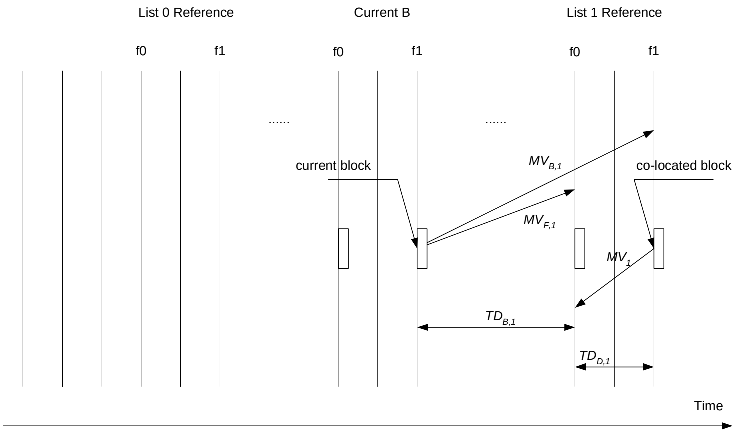


Figure 10-5 – The list 0 motion vector of the co-located block in field 1 of the list 1 reference frame may point to field 0 of the same frame.

Case 3: The current macroblock is in field mode and its co-located macroblock in the list 1 reference picture is in frame mode, as shown in Figure 10-6. Let $y_{current}$ and $y_{co\text{-}located}$ be the vertical indices of the blocks in the current macroblock and in its co-located macroblock respectively. Then, $y_{co\text{-}located} = 2 \times y_{current}$. The blocks in the current macroblock and its co-located macroblock have the same horizontal indices. The list 0 reference field is the same-parity field of the list 0 frame, and the list 1 reference field will be the same-parity field of the list 1 reference frame as shown in Figure 10-6.

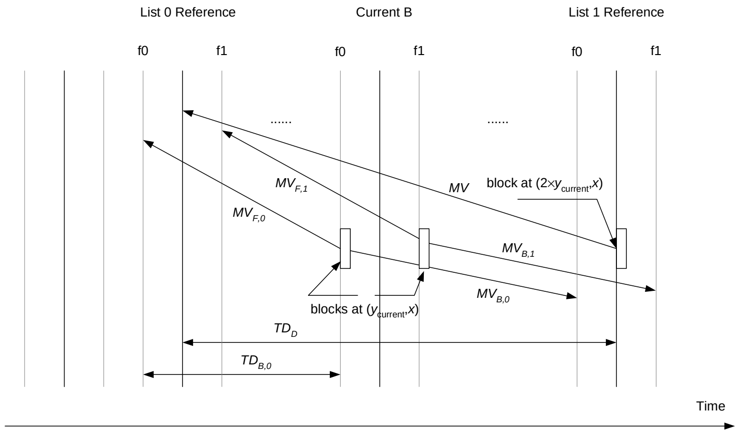


Figure 10-6 – The current macroblock is in field mode and its co-located macroblock in the list 1 reference picture is in frame mode

The motion vectors of a direct mode block are calculated from the list 0 motion vector of the co-located block in the list 1 reference frame as follows:

$Z = TD_{B,i} \times 256/\ TD_D \qquad MV_{F,i} = (Z \times MV + 128) >> 8$

$W = Z - 256 \qquad MV_{B,i} = (W \times MV + 128) >> 8$

In the case that the block in the list 1 reference frame used for the direct mode motion vector calculation has a list 0 motion vector pointing to a long-term picture, the list 0 and list 1 motion vectors for the direct mode are calculated by

$MV_{F,i} = MV$

$MV_{B,i} = 0$

Case 4: The current macroblock is in frame mode while its co-located macroblock in the list 1 reference picture is in field mode, as shown in Figure 10-7. Let $y_{current}$ and $y_{co\text{-}located}$ be the vertical indices of the blocks in the current macroblock and in its co-located macroblock respectively. Then, $y_{co\text{-}located} = y_{current} / 2$. The blocks in the current macroblock and in its co-located macroblock have the same horizontal indices. The two fields of the co-located block in the list 1 reference may be coded in different modes. Since field 0 of the list 1 reference is temporally close to the current B picture, it is used in calculating the motion vectors and determining the references for direct mode current blocks as shown in Figure 10-7.

Two frame-based motion vectors of direct mode block are calculated as follows:

$Z = TD_B \times 256/\ TD_{D,0} \qquad MV_F = (Z \times MV_0 + 128) >> 8$

$W = Z - 256 \qquad MV_B = (W \times MV_0 + 128) >> 8$

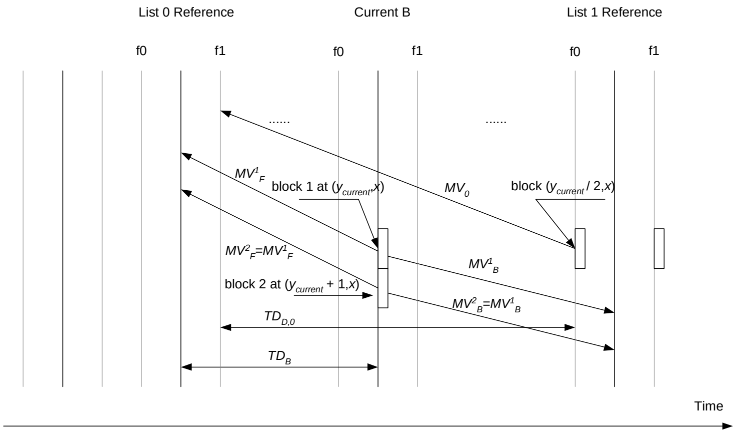


Figure 10-7 – The current macroblock is in frame mode while its co-located macroblock in the list 1 reference picture is in field mode.

In the case that the block in the list 1 reference field used for the direct mode motion vector calculation has a list 0 motion vector pointing to a long-term picture, the list 0 and list 1 motion vectors for the direct mode are calculated by

$MV_F = MV_0$

$MV_B = 0$

Ed.Note: I don't understand the following sentence. More explanation maybe needed here. This appears to indicate that if the co-located macroblock is in field, then also field mode should be used for the current macroblock/block. Considering though that AFF is done in pairs of macroblocks, is somehow the whole group forced to be in the same mode as its co-located? This is what this seems to indicate.] When mb_frame_field_adaptive_flag = = 1, the current direct-mode macroblock is in the same frame/field coding mode as its co-located macroblock.

10.4 Weighted prediction signal generation procedure

If weighted_pred_ flag is equal to one, explicit weighted prediction is applied to P and SP slices. If weighted_pred_explicit_flag is equal to one, explicit weighted bi-prediction is applied to B slices. If weighted_bipred_implicit_flag is equal to one, implicit weighted bi-prediction is applied to B slices.

10.4.1 Weighted prediction in P and SP slices

In P and SP slices, when weighted_pred_flag is equal to one, weighted prediction is applied for predicted macroblocks. When 0 ≤ mb_type ≤ 4, the luma prediction is generated as

$$P = \text{clip1}((P_0 \times W_0 + 2^{LWD-1}) >> LWD + O_0)$$

and the chroma prediction block is generated as

$$P = \text{clip1}(128 + ((CP_0 - 128) \times CW_0 + 2^{CWD-1}) >> CWD + CO_0)$$

where

P_0 = reference prediction block

LWD = luma_log_weight_denom

W_0 = luma_weight_l0[ref_idx_l0]

O_0 = luma_offset_l0[ref_idx_l0]

j = 0 for Cb and 1 for Cr

CP_0 = chroma reference prediction block

CWD = chroma_log_weight_denom

CW_0 = chroma_weight_l0[ref_idx_l0][j]

CO_0 = chroma_offset_l0[ref_idx_l0][j]

To limit the calculation to 16-bit precision, the following conditions shall be met:

$$-128 \le W_0 \le 127$$

$$-128 \le CW_0 \le 127$$

10.4.2 Explicit weighted bi-prediction in B slices

In B slices, when weighted_bipred_explicit_flag is equal to one, weighted prediction is applied for predicted macroblocks. For reference list 0 prediction, the luma prediction is generated as

$$P = \mathrm{clip1}((P_0 \times W_0 + 2^{LWD-1}) >> LWD + O_0)$$

For reference list 1 prediction, the luma prediction block is generated as:

$$P = \mathrm{clip1}((P_1 \times W_1 + 2^{LWD-1}) >> LWD + O_1)$$

where

P_0 = reference prediction block from list 0

P_1 = reference prediction block from list 1

LWD = luma_log_weight_denom

W_0 = luma_weight_l0[ref_idx_l0]

W_1 = luma_weight_l1[ref_idx_l1]

O_0 = luma_offset_l0[ref_idx_l0]

O_1 = luma_offset_l1[ref_idx_l1]

When bi-prediction is used, the luma prediction block is generated as:

$$P = \mathrm{clip1}((P_0 \times BDW_0 + P_1 \times BDW_1 + 2^{LWD}) >> (LWD + 1) + BDO)$$

where

BDW_0 = luma_weight_bipred_l0[ref_idx_l0][ref_idx_l1]

BDW_1 = luma_weight_bipred_l1[ref_idx_l0][ref_idx_l1]

BDO = luma_offset_bipred[ref_idx_l0][ref_idx_l1]

For reference list 0 prediction, the chroma prediction block is generated as

$$P = \mathrm{clip1}(128 + ((CP_0 - 128) \times CW_0 + 2^{CWD-1}) >> CWD + CO_0) .$$

For reference list 1 prediction, the chroma prediction block is generated as:

$$P = \mathrm{clip1}(128 + ((CP_1 - 128) \times CW_1 + 2^{CWD-1}) >> CWD + CO_1) .$$

When bi-prediction is used, the chroma prediction block is generated as

$$P = \mathrm{clip1}(128 + ((CP_0 - 128) \times CBDW_0 + (CP_1 - 128) \times CBDW_1 + 2^{CWD}) >> (CWD + 1) + CBDO)$$

where

j = 0 for Cb and 1 for Cr

CP_0 = chroma reference prediction block from list 0

CP_1 = chroma reference prediction block from list 1

CWD = chroma_log_weight_denom

CW_0 = chroma_weight_l0[ref_idx_l0][j]

CW_1 = chroma_weight_l1[ref_idx_l1][j]

CO_0 = chroma_offset_l0[ref_idx_l0][j]

CO_1 = chroma_offset_l1[ref_idx_l1][j]

$CBDW_0$ = chroma_weight_bipred_l0[ref_idx_l0][ref_idx_l1][j]

$CBDW_1$ = chroma_weight_bipred_l1[ref_idx_l0][ref_idx_l1][j]

$CBDO$ = chroma_offset_bipred[ref_idx_l0][ref_idx_l1][j]

To limit the calculation to 16-bit precision, the following conditions shall be met:

$$-128 \leq W_0 \leq 127$$

$$-128 \leq W_1 \leq 127$$

$$-128 \leq W_0 + W_1 \leq 127$$

$$-128 \leq CW_0 \leq 127$$

$$-128 \leq CW_1 \leq 127$$

$$-128 \leq CW_0 + CW_1 \leq 127$$

$$-128 \leq BDW_0 + BDW_1 \leq 127$$

$$-128 \leq CBDW_0 + CBDW_1 \leq 127$$

10.4.3 Implicit bi-predictive weighting

When weighted_bipred_implicit_flag is equal to 1, the prediction weighting factors are not sent explicitly and the luma and chroma predictions are generated as follows.

If the decoding order of the reference picture indicated by ref_idx_l0 is previous to or the same as that indicated by ref_idx_l1, or for skipped macroblocks or direct mode, the prediction signals are generated as follows:

$$P = \text{clip1}(P_0 \times 2 - P_1)$$

where

P_0 = reference prediction block from list 0

P_1 = reference prediction block from list 1;

otherwise, the prediction signals are generated as follows

$$P = (P_0 + P_1 + 1) >> 1$$

where

P_0 = reference prediction block from list 0

P_1 = reference prediction block from list 1.

11 Decoding process for SP and SI slices

11.1 General

SP slices make use of motion-compensated predictive coding to exploit temporal redundancy in the sequence, in a similar manner to P slices. SI slices make use of spatial prediction, in a similar manner to I slices. Unlike P slices, however, SP slice coding allows identical reconstruction of a slice even when different reference pictures are being used. SI slice coding allows identical reconstruction to a corresponding SP slice.

NOTE - Above mentioned 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 Intra type (Intra_4x4 or Intra_16x16) or Pred type (MbSkip, Pred_L0_16x16, Pred_L0_L0_16x8, Pred_L0_L0_8x16, Pred_8x8 or Pred_8x8ref0). An SI slice consists of macroblocks coded either as Intra type or SIntra4x4 type.

Intra macroblocks in SP slices shall be decoded as described in subclause 8.3. All other macroblocks, including MbSkip, are decoded as described below.

The Intra_8x8 sub-partition mode shall not be present in SP slices.

Intra macroblocks in SI slices are decoded as described in subclause 8.3, with the addition that the prediction mode of a neighbouring SIntra_4x4 block is considered to be “mode 2: DC prediction”. SIntra_4x4 macroblocks are decoded as described below.

11.2 SP decoding process for non-switching pictures

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag = = 0, except those with slice_type() equal to Intra_4x4 or Intra_16x16. It does not apply to SI slices.

Figure 11-1 depicts generic decoding for non-intra coded macroblocks in SP slices. The prediction P(x,y) for the current macroblock of the slice being decoded is formed by the motion-compensated prediction block using the same process as is used in P slice decoding. After forming the predicted block P(x,y), decoding is performed as follows.

Deleted: compensated

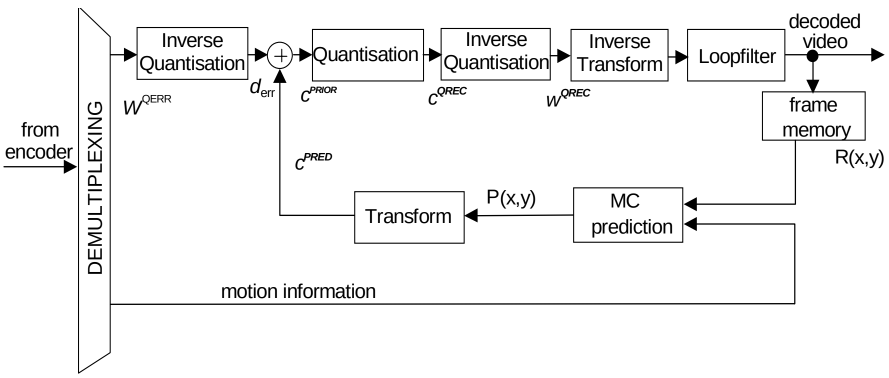


Figure 11-1 – A block diagram of a conceptual decoder for non-intra coded macroblocks in SP slices in which sp_for_switch_flag = = 0.

11.2.1 Luma transform coefficient decoding

The predicted block, P(x,y), where P(x,y) ={p_{00} … p_{33}}, is transformed according to Equation 11-1 to produce transform coefficients c^{PRED}.

$$c^{PRED} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \quad (11\text{-}1)$$

The received prediction residual coefficients, w^{QERR}, are scaled using quantisation parameter QP, and added to the transform coefficients of the prediction block, as in Equation 11-2.

$$c_{ij}^{PRIOR} = c_{ij}^{PRED} + (((w_{ij}^{QERR} * R_{ij}^{(QP\%6)} * A_{ij}) << (QP/6)) >> 6) \quad i, j = 0,\ldots,3 \quad (11\text{-}2)$$

where R is specified in Equation 8-40, and where A is specified as:

$A_{ij}^{(m)}$= 16 for (i,j) = {(0,0),(0,2),(2,0),(2,2)},

$A_{ij}^{(m)}$ = 25 for (i,j) = {(1,1),(1,3),(3,1),(3,3)},

$A_{ij}^{(m)}$ = 20 otherwise;.

For luma, QP = QP_Y, as specified in Equation 7-7 and 7-9.

The coefficients $Q_{ij}^{(m)}$, used in the formulas below, are specified as:

$Q_{ij}^{(m)}$= $M_{m,0}$ for (i,j) = {(0,0),(0,2),(2,0),(2,2)},

$Q_{ij}^{(m)}$ = $M_{m,1}$ for (i,j) = {(1,1),(1,3),(3,1),(3,3)},

$Q_{ij}^{(m)}$ = $M_{m,2}$ otherwise;

where the first and second subscripts of *M* are row and column indices, respectively, of the matrix specified as:

$$M = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix}$$

The resulting sum, c^{PRIOR}, is quantised with a quantisation parameter *QS*, as in Equation 11-3.

For luma, QS = QS_Y, which is specified in Equation 7-8.

$$c_{ij}^{QREC} = \{Sign(c_{ij}^{PRIOR}) * [Abs(c_{ij}^{PRIOR}) * Q_{ij}^{(QS\%6)} + (1<<(15+QS/6))]\}>>(16+QS/6) \quad i, j = 0,\ldots,3 \quad (11\text{-}3)$$

These quantised levels, c^{QREC}, are scaled as in Equation 11-4.

$$w_{ij}^{QREC} = (c_{ij}^{QREC} * R_{ij}^{(QS\%6)}) << (QS/6) \quad i, j = 0,\ldots, 3 \quad (11\text{-}4)$$

where R is specified in Equation 8-40.

The transform and reconstruction processes are performed for these scaled levels, as specified in Equations 8-48 through 8-59. Finally, after the reconstruction of a macroblock, filtering takes place as described in subclause 8.7.

11.2.2 Chroma transform coefficient decoding

The decoding of chroma components for non-intra coded macroblocks in SP slices is similar to the decoding of luma components.

The predicted block, P(x,y), where P(x,y) ={p_{00} … p_{33}}, is transformed according to Equation 11-1 to produce transform coefficients c^{PRED}. An additional 2x2 transform is applied to the DC coefficients of these blocks. The 2 dimensional 2x2 transform procedure is specified in Equation 11-5:

$$c^{PRED} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} DC_{00} & DC_{01} \\ DC_{10} & DC_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (11\text{-}5)$$

The received DC prediction residual coefficients, w^{QERR}, are scaled using quantisation parameter QP, and added to the DC transform coefficients of the prediction block, as in Equation 11-6.

$$c_{ij}^{PRIOR} = c_{ij}^{PRED} + (((w_{ij}^{QERR} * R_{ij}^{(QP\%6)} * A_{ij}) << (QP/6)) >>5) \quad i, j = 0,\ldots, 3 \qquad (11\text{-}6)$$

The resulting sum, c^{PRIOR}, is quantised with a quantisation parameter *QS*, as in Equation 11-7.

$$c_{ij}^{QREC} = \{Sign(c_{ij}^{PRIOR})*[Abs(c_{ij}^{PRIOR})*Q_{ij}^{(QS\%6)} + (1<<(16+QS/6))]\}>>(17+QS/6)$$
$$i, j = 0, \ldots, 3 \qquad (11\text{-}7)$$

These quantised levels, c_{ij}^{QREC}, are scaled as in Equation 11-4.

AC coefficients are decoded in an identical process to that used for luma.

The value of *QP* to be used for chroma data, denoted QP_C, is obtained from QP_Y using the relationship specified in Table 9-1. Similarly, the value of *QS* to be used for chroma data, denoted QS_C, is obtained from QS_Y using the relationship specified in Table 9-1.

11.3 SP and SI slice decoding process for switching pictures

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag = = 1, except those with slice_type() equal to Intra_4x4 or Intra_16x16; and to all macroblocks in SI slices, except those with slice_type() equal to Intra_4x4 or Intra_16x16.

Figure 11-2 depicts generic decoding for macroblocks in SI and SP slices that are not Intra coded. In SP slices, the prediction P(x,y) for the current coded macroblock of the slice being decoded is formed by the inter prediction block using the same process as is used in P slice decoding. In SI slices, the prediction P(x,y) is formed by intra prediction using the same process as is used in I slice decoding. After forming the predicted block P(x,y), the decoding of SI and non-intra coded macroblocks in SP slices follows the same steps.

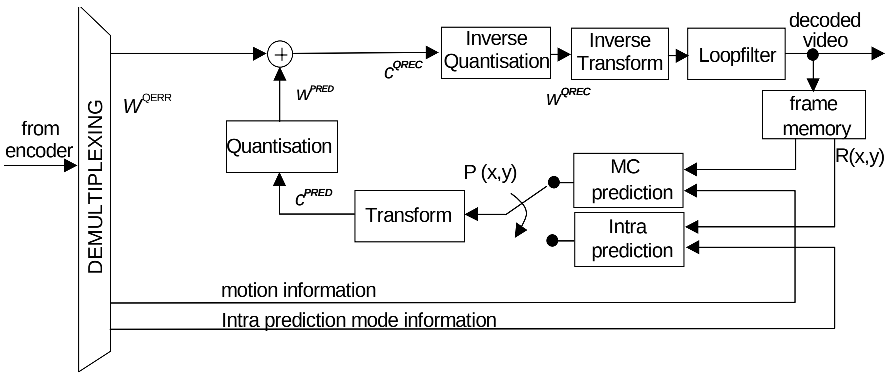


Figure 11-2 – A block diagram of a conceptual decoder for non-intra macroblocks in SI slices; and for non-intra coded macroblocks in SP slices in which sp_for_switch_flag = = 1.

When decoding SIntra_4x4 macroblocks, the intra prediction modes of the neighbouring SIntra_4x4 and Intra_4x4 blocks are taken into account as described in subclause 8.5.

When decoding Intra_4x4 macroblocks, the intra prediction modes of the neighbouring Intra_4x4 blocks are taken into account as described in subclause 8.5, but the prediction mode of the neighbouring SIntra_4x4 blocks are considered to be "mode 2: DC prediction".

11.3.1 Luma transform coefficient decoding

The predicted block, P(x,y), where P(x,y) ={p_{00} … p_{33}}, is transformed according to Equation 11-1 to produce transform coefficients c^{PRED}. These transform coefficients are then quantised with a quantisation parameter *QS*, as in Equation 11-8.

$$w_{ij}^{PRED} = \{Sign(c_{ij}^{PRED})*[Abs(c_{ij}^{PRED})*Q_{ij}^{(QS\%6)} + (1<<(15+QS/6))]\}>>(16+QS/6) \quad i, j = 0,\ldots, 3 \qquad (11\text{-}8)$$

Note: Equation 11-8 is the same as Equation 11-3 except for the change of name of input and output variables.

For luma, QS = QS_Y, which is specified in Equation 7-8.

The received prediction residual coefficients, w^{QERR}, are added to these quantised transform coefficients of the prediction block, as in Equation 11-9.

$$c_{ij}^{QREC} = w_{ij}^{PRED} + w_{ij}^{QERR} \qquad i, j = 0,\ldots, 3 \qquad (11\text{-}9)$$

These quantised levels, c^{QREC}, are decoded as in subclause 11.1.1???

11.3.1.1 Chroma transform coefficient decoding

The decoding of chroma components for SP and SI non-intra macroblocks is similar to the decoding of luma components.

The predicted block, P(x,y), where P(x,y) ={p_{00} … p_{33}}, is transformed according to Equation 11-1 to produce transform coefficients c^{PRED}. An additional 2x2 transform is applied to the DC coefficients of these blocks as in Equation 11-5:

The DC transform coefficients are then quantised with a quantisation parameter *QS*, as in Equation 11-10.

$$w_{ij}^{PRED} = \{Sign(c_{ij}^{PRED})*[Abs(c_{ij}^{PRED})*Q_{ij}^{(QS\%6)} + (1<<(16+QS/6))]\}>>(17+QS/6) \quad i, j = 0,\ldots, 3 \qquad (11\text{-}10)$$

NOTE - Equation 11-10 is the same as Equation 11-7 except for the change of name of input and output variables.

The received prediction residual DC coefficients, w^{QERR}, are added to these quantised DC transform coefficients of the prediction block, as in Equation 11-9.

AC coefficients are decoded in an identical process to that used for luma.

The value of *QP* to be used for chroma data, denoted QP_C, is obtained from QP_Y using the relationship specified in Table 9-1. Similarly, the value of *QS* to be used for chroma data, denoted QS_C, is obtained from QS_Y using the relationship specified in Table 9-1.

12 Adaptive block size transforms

12.1 Introduction

In this clause, the modifications to the syntax and semantics in clause 7 and the changes to the decoding process in clause 8 and to entropy coding in clause 9 for adaptive block size transforms are described.

If adaptive_block_size_transform_flag = = 1, additional transforms of size 4x8, 8x4, and 8x8 are specified for the luma residual. The chroma residual decoding process remains unchanged. Adaptive block size transforms are used for all macroblocks with QP_Y >= 12. In inter predicted macroblocks, the transform block size is indicated by the block size used for inter prediction. For intra macroblocks, the block size used for intra prediction is connected to the block size of the transformation. For intra macroblocks in inter slices, the block size is indicated by the syntax element intra_block_typeABT. For intra slices, the intra block size is indicated by the macroblock block mode.

Figure 12-1 shows the order of the assignments of syntax elements for luma resulting from coding a macroblock to sub-blocks of the macroblock if the ABT feature is used. The assignment of blocks and coded_block_patternY is specified in Figure 12-1. An 8x8 block may contain 1, 2, or 4 transform blocks. An indication that an 8x8 block contains coefficients means that the 8x8 transform blocks or one or more of the 2, or 4 transform blocks within the 8x8 block contains coefficients. The chroma 4x4 residual blocks are ordered after the luma blocks as indicated in Figure 6-6.

Deleted: s

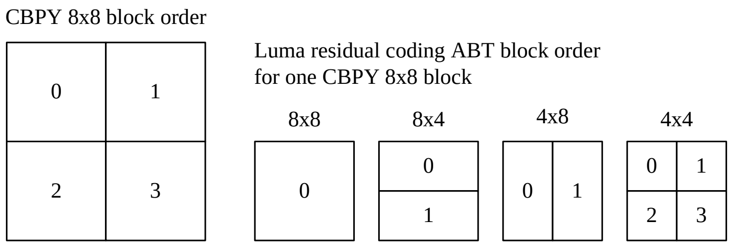


Figure 12-1 – Ordering of blocks for CBPY and luma residual coding of ABT blocks

12.2 ABT Syntax

12.2.1 Macroblock layer syntax

[Ed.Note: changed according to macroblock_layer in FCD-34wcm]

macroblock_layer_abt( ) {	**Category**	**Descriptor**
mb_type	4	ue(v) | ae(v)
if(num_mb_partition[mb_type] = = 4)		
sub_mb_pred_abt(mb_type)	4	
else		
mb_pred_abt(mb_type)	4	
SendResidual = 0		
if(mb_partition_pred_mode(, 1) = = Intra && mb_type ! = Intra_4x4) { /* Intra_16x16_X_Y_Z mb_type */		
coded_block_pattern	4	me(v) | ae(v)
if(coded_block_pattern > 0)		
SendResidual = 1		
} else {		
coded_block_pattern	4	me(v) | ae(v)
if(coded_block_pattern > 0)		
SendResidual = 1		
}		
if(SendResidual) {		
if(!mb_frame_field_adaptive_flag | | (mb_frame_field_adaptive_flag && (pic_structure = = 0 | | pic_structure = = 3 | | pic_structure = = 4) && first_non_skip_mb_in_pair( ))		
delta_qp	4	se(v) | ae(v)
residual_abt( )	5 | 6	
}		
}		

12.2.1.1 Macroblock prediction syntax

[Ed.Note: changed according to mb_pred in FCD-34wcm]

mb_pred_abt(mb_type) {	**Category**	**Descriptor**
if(mb_partition_pred_mode(, 1) = = Intra) {		

if(mb_type = = Intra_4x4
mb_type = = ABTIntra_4x4 ||
mb_type = = ABTIntra_4x8 ||
mb_type = = ABTIntra_8x4 ||
mb_type = = ABTIntra_8x8) {		

if(MbABTFlag && (slice_type( ) = = Pred || slice_type( ) = = BiPred)		
intra_block_typeABT	4	me(v) | ae(v)
for(i = 0; i < num_mb_intra_partition( ); i++) /* for each luma block */		
intra_pred_mode	4	ce(v) | ae(v)
}		
intra_chroma_pred_mode	4	ue(v) | ae(v)
} else if(mb_type != Direct_16x16) {		
for(i = 0; i < num_mb_partition(mb_type, i); i++)		
if(num_ref_idx_l0_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) != Pred_L1)		
ref_idx_l0	4	ue(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type, i); i++) {		
if(num_ref_idx_l1_active_minus1 > 0 && mb_partition_pred_mode(mb_type, i) != Pred_L0)		
ref_idx_l1	4	ue(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode (mb_type, i) != Pred_L1)		
for(j = 0; j < 2; j++)		
mvd_l0[i **][** j **]**	4	se(v) | ae(v)
for(i = 0; i < num_mb_partition(mb_type); i++) {		
if(mb_partition_pred_mode(mb_type, i) != Pred_L0)		
for(j = 0; j < 2; j++)		
mvd_l1[i **][** j **]**	4	se(v) | ae(v)
}		
}		

12.2.1.2 Sub macroblock prediction syntax

[Ed.Note: changed according to sub_ mb_pred in FCD-34wcm]

sub_mb_pred_abt(mb_type) {	Category	Descriptor
for(i = 0; i < 4; i++) /* for each sub macroblock */		
sub_mb_type[i **]**	4	ue(v) | ae(v)
IntraChromaPredModeFlag = 0		
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] = = Intra_8x8) {		
if(MbABTFlag) {		
intra_block_typeABT	4	me(v) | ae(v)
for(j = 0; j < num_sub_mb_intra_partition( ); j++) {		
intra_pred_mode	4	ce(v) | ae(v)
IntraChromaPredModeFlag = 1		
}		
}		
if(IntraChromaPredModeFlag)		
intra_chroma_pred_mode	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l0_active_minus1 > 0 && mb_type ! = Pred_8x8ref0 && sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L1)		
ref_idx_l0	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(num_ref_idx_l1_active_minus1 > 0 && (sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L0)		
ref_idx_l1	4	ue(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L1)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l0[i **][** j **][** k **]**	4	se(v) | ae(v)
for(i = 0; i < 4; i++) /* for each sub macroblock */		
if(sub_mb_type[i] ! = Intra_8x8 && sub_mb_type[i] ! = Direct_8x8 && sub_mb_pred_mode(sub_mb_type[i]) ! = Pred_L0)		
for(j = 0; j < num_sub_mb_partition(sub_mb_type[i]); j++)		
for(k = 0; k < 2; k++)		
mvd_l1[i **][** j **][** k **]**	4	se(v) | ae(v)
}		

12.2.1.3 Residual data syntax

residual_abt(mb_type) {	**Category**	**Descriptor**
if(entropy_coding_mode = = 1) {		
residual_4x4block = residual_4x4block_cabac( ) /* function pointer */	5 | 6	
residual_subblock = residual_subblock_cabac( ) /* function pointer */	5 | 6	
} else {		
residual_4x4block = residual_4x4block_cavlc( ) /* function pointer */	5 | 6	
residual_subblock = residual_subblock_cavlc( ) /* function pointer */	5 | 6	
}		
if(mb_type = = Intra_16x16)		
residual_4x4block(intra16x16DC, 16) [Ed.Note: define intra16x16DC]	5	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
for(i4x4 = 0; i4x4 < num_sub_blocks( ); i4x4++) /* each sub-block of block */		
if(coded_block_pattern & (1 << i8x8))		
if(mb_type = = Intra_16x16)		
residual_4x4block(intra16x16AC, 16)	5	
else		
if(MbABTFlag && (slice_type() = = Intra || mb_type = = Intra_4x4)		
residual_subblock(IntraABT, sub_block_type)	5 | 6	
else		
residual_subblock(InterABT, sub_block_type)	5 | 6	
if(coded_block_pattern & 0x30) /* chroma DC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
residual_4x4block(chromaDC, 4)) [Ed.Note: define chromaDC]	5 | 6	
if(coded_block_pattern & 0x20) /* chroma AC residual coded */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
For(i4x4 = 0; i4x4 < 4; i4x4++)		
residual_4x4block(chromaAC, 16)	5 | 6	
}		

12.2.1.3.1 Residual sub block CAVLC syntax

residual_subblock_cavlc(block_coding_type, sub_block_type) {		
if(! MbABTFlag)		
residual_4x4block_cavlc(luma, 16)	5 | 6	
else {		
if(block_coding_type = = IntraABT) {		
num_coeff_abt	5 | 6	ce(v)
for (i=0; i<num_coeff_abt; i++) {		
code_number	5 | 6	ce(v)
if(code_number ! = escape) {		
level[i] = code_number2level_intra(code_number, sub_block_type)		
run[i] = code_number2run_intra(code_number, sub_block_type)		
} else {		
code_number	5 | 6	ce(v)
level[i] = escape_level[code_number]		
code_number	5 | 6	ce(v)
run[i] = escape_run[code_number]		
}		
}		
} else {		
for(i=0; i<max_numcoeffABT(sub_block_type); i++) {		
code_number	5 | 6	ce(v)
if(code_number = = 0)		
break;		
if(code_number ! = escape) {		
run[i] = code_number2run_inter(code_number, sub_block_type)		
level[i] = code_number2level_inter(code_number, sub_block_type)		
} else {		
code_number	5 | 6	ce(v)
level[i] = escape_level[code_number]		
code_number	5 | 6	ce(v)
run[i] = escape_run[code_number]		
}		
}		
}		
}		
}		

12.2.1.3.2 Residual sub block CABAC syntax

residual_subblock_cabac(block_coding_type, sub_block_type) {	**Category**	**Descriptor**
if(! MbABTFlag)		
residual_4x4block_cabac(luma, 16)	5 | 6	
else {		
if(sub_block_type ! = 8x8)		
cbp4	5 | 6	ae(v)
if(cbp4 || sub_block_type = = 8x8) {		
max_numcoeff = max_num_coeff_abt(sub_block_type)		
significant_coeff[max_numcoeff - 1] = 1 [Ed.Note from 4x4]		
for(i = 0; i < max_numcoeff; i++) {		
significant_coeff[i **]**	5 | 6	ae(v)
if(significant_coeff[i] && i < max_numcoeff-1) {		
last_significant_coeff[i **]**	5 | 6	ae(v)
if(last_significant_coeff[i])		
max_numcoeff = i + 1		
}		
}		
for(i = max_numcoeff-1; i >= 0; i--)		
if(significant_coeff[i]) {		
coeff_absolute_value_minus1[i **]**	5 | 6	ae(v)
coeff_sign[i **]**	5 | 6	ae(v)
}		
}		
}		
}		

12.3 ABT Semantics

12.3.1 Macroblock layer semantics

MbABTFlag indicates the usage of ABT for the macroblock. If adaptive_block_transform_flag is equal 1 and QP_Y >= '12' MbABTFlag = 1 else MbABTFlag = 0.

The meaning of mb_type 'Intra_4x4' in Inter slices is modified for ABT. If MbABTFlag is equal 1, this macroblock type indicates application of ABT Intra prediction and transformation. In Inter slices, the block size used for prediction and transform is indicated by the syntax element intra_block_typeABT. For I slices, the prediction and transform block size is derived from the macroblock mode. The modified macroblock types for I slices are specified in Table 12-1below.

Table 12-1 – Modified macroblock types for I slices

Value of mb_type	Name of mb_type in I slices if MbABTFlag = = 1	mb_partition_pred_ mode(, 1)
0	ABTIntra_4x4	Intra
1	ABTIntra_4x8	Intra
2	ABTIntra_8x4	Intra
3	ABTIntra_8x8	Intra

12.3.1.1 Macroblock prediction semantics

num_mb_intra_partition( ) depends on mb_type in I slices and on intra_block_typeABT in P slices. The assignment of num_mb_intra_partition( ) is specified in Table 12-2.

intra_block_typeABT indicates the blocksize used for ABT intra decoding in Inter slices. Blocks of size 4x4, 4x8, 8x4 and 8x8 samples may be used for ABT intra prediction. Table 12-2provides num_mb_intra_partition and num_sub_mb_intra_partition specifying the number of intra_pred_mode syntax elements to be decoded.

Table 12-2 – ABT intra partitions

mb_type	intra_block_typeABT	num_mb_intra_partition( )	num_sub_mb_intra_partition( )
ABTIntra_4x4	4x4	16	4
ABTIntra_4x8	4x8	8	2
ABTIntra_8x4	8x4	8	2
ABTIntra_8x8	8x8	4	1

12.3.1.2 Sub macroblock prediction semantics

If MbABTFlag is equal 1, the number of decoded intra prediction modes is indicated by num_sub_mb_intra_partition( ) as specified in Table 12-2.

12.3.1.3 Residual data semantics

num_sub_blocks( ) indicates the number of partitions in a luma 8x8 block. If MbABTFlag is equal 0 num_sub_blocks is 4. If MbABTFlag equals 1, num_sub_blocks may be 1, 2, or 4 depending on sub_block_type as specified in Table 12-3.

Table 12-3 – ABT Intra Block Types

sub_block_type	num_sub_blocks( )	max_num_coeff_abt( )
4x4	4	16
4x8	2	32
8x4	2	32
8x8	1	64

sub_block_type indicates the block type used for decoding of an 8x8 block. The decoded blocks may be of size 4x4, 4x8, 8x4, or 8x8 samples. For intra slices with MbABTFlag = = 1, sub_block_type for macroblock type ABTIntra_NxM is NxM. For intra macroblocks and intra sub macroblocks in inter slices sub_block_type is equal to intra_block_typeABT. For inter macroblocks with macroblock type Pred_x_NxM or Pred_x_y_NxM or BiPred_x_NxM or BiPred_x_y_NxM, x, y = L0, L1, Bi, sub_block_type is N'xM'. N' and M' are derived from N and M by clipping:

$$N' = \mathrm{Clip3}(\ 4, 8, N\ ) \qquad (12\text{-}1)$$

$$M' = \mathrm{Clip3}(\ 4, 8, M\ ) \qquad (12\text{-}2)$$

For sub macroblocks with sub macroblock type Pred_x_NxM or Pred_x_y_NxM or BiPred_x_NxM or BiPred_x_y_NxM, x, y = L0, L1, Bi, sub_block_type is NxM.

For macroblocks and sub macroblocks that are predicted in direct mode, sub_block_type is derived from the co-located macroblock or sub macroblock. If MbABTFlag is equal 1 for the co-located macroblock, sub_block_type for the current macroblock is derived from the co-located macroblock type or sub macroblock type as specified above. If MbABTFlag is equal 0 for the co-located macroblock and the type of the co-located macroblock is not Intra_16x16, sub_block_type is equal 4x4. If MbABTFlag is equal 0 for the co-located macroblock and the type of the co-located macroblock is Intra_16x16, sub_block_type is equal 8x8. If the co-located macroblock type is MbSkip, sub_block_type is equal 8x8.

block_coding_type indicates if an Inter or an Intra block is decoded. Inter blocks are indicated by block_coding_type = 'InterABT', intra blocks are indicated by block_coding_type = 'IntraABT'.

12.3.1.3.1 Residual sub block CAVLC semantics

num_coeff_abt indicates the number of coefficients to be decoded. num_coeff_abt is bound by max_num_coeff_abt( ) specified in Table 12-3.

code_number indicates the number of the decoded codeword. The code structure of the codewords depending on the syntax element to be decoded is specified in subclause 12.4.1.

code_number2run_intra( ) retrieves run from Table 12-9for Intra blocks dependent on QP_Y.

code_number2run_inter( ) retrieves run from Table 12-9for Inter blocks.

code_number2level_intra( ) retrieves level from Table 12-9for Intra blocks dependent on QP_Y.

code_number2level_inter( ) retrieves level from Table 12-9for Inter blocks.

escape indicates separate( ) decoding of level and run. The value of escape is specified in subclause 12.5.1.2.2

escape_level( ) retrieve level symbol after escape as specified in Table 12-8.

escape_run( ) retrieve run symbol after escape as specified in Table 12-6.

12.3.1.3.2 Residual sub block CABAC semantics

The syntax elements of residual_subblock_cabac( ) are specified in subclause 7.4.5.3.2.???

12.4 ABT decoding process

12.4.1 Intra Prediction for 4x8, 8x4, and 8x8 luma blocks

[Ed.Note: currently rounding for these prediction modes is slightly different from 4x4 prediction modes. harmonization at the next meeting and merging 4x4-8x8 into one subclause].

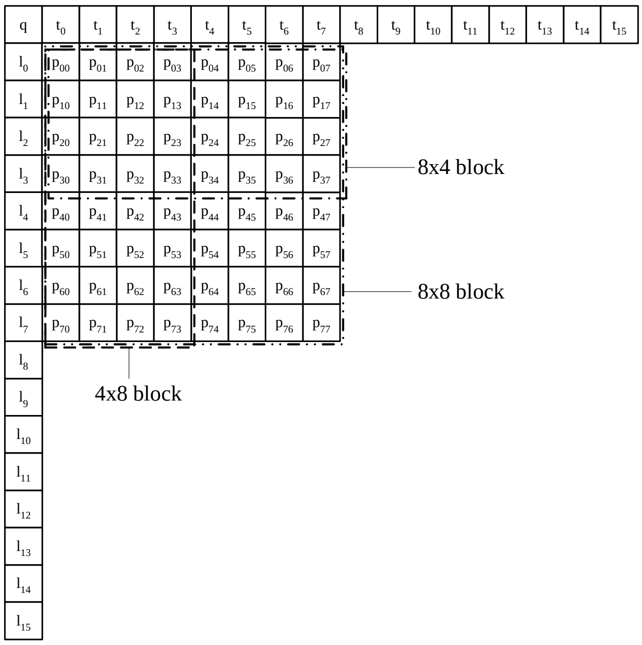


Figure 12-2 – Identification of samples used for ABT intra spatial prediction for 4x8, 8x4, and 8x8 luma blocks

Figure 12-2 illustrates the intra prediction for 4x8, 8x4, and 8x8 blocks that may be used in addition to the 4x4 intra prediction specified in subclause 8.5. The samples p_{mn}, with m=0 to M-1 and n=0 to N-1, M,N = {4,8}, are predicted using samples t_k, k=0 to (N+M-1), q, and l_k, k=0 to (N+M-1), from neighbouring blocks.

Samples t_k, k=0 to (N+M-1) or samples l_k, k=0 to (N+M-1) shall be considered not available under the following circumstances:

- if they are outside the picture or outside the current slice,
- if they belong to a macroblock that is subsequent to the current macroblock in raster scan order,
- if they are sent later than the current block in the order shown in Figure 12-1, or
- if they are in a non-intra macroblock and constrained_intra_pred is 1.

When samples t_k, k=N to (N+M-1) are not available the sample value of t_{N-1} is substituted for the samples t_k, k=N to (N+M-1). When samples l_k, k=M to (N+M-1) are not available the sample value of l_{M-1} is substituted for the samples l_k, k=N to (N+M-1).

12.4.1.1 Mode 0: vertical prediction

t_k, k = 0 to (N-1) shall be available. Prediction sample q is denoted as t_{-1}. If q is not in the slice, t_0 substitutes q.

block size		samples	predicted by
4x8	k = 0 to 7, i = 0 to 3		
8x4	k = 0 to 3, i = 0 to 7	p_{ki}	$(t_{i-1} + t_i<<1 + t_{i+1} + 2) >> 2$
8x8	k = 0 to 7, i = 0 to 7		

12.4.1.2 Mode 1: horizontal prediction

l_k, k = 0 to (M-1) shall be available. Prediction sample q is denoted as l_{-1}. If q is not in the slice, l_0 substitutes q.

block size		samples	predicted by
4x8	k = 0 to 7, i = 0 to 3		
8x4	k = 0 to 3, i = 0 to 7	p_{ki}	$(l_{i-1} + l_i<<1 + l_i+1 + 2) >> 2$
8x8	k = 0 to 7, i = 0 to 7		

12.4.1.3 Mode 2: DC prediction

For DC prediction, all samples p_{ki} are predicted by the same value p.

a) If l_k, k = 0 to (M-1) are available and t_k, k = 0 to (N-1) are available

block size	p
4x8	$((t_0 + t_1 + t_2 + t_3 + 2) >> 2 + (l_0 + l_1 + l_2 + l_3 + l_4 + l_5 + l_6 + l_7 + 4) >> 3) >> 1$
8x4	$((t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + 4) >> 3 + (l_0 + l_1 + l_2 + l_3 + 2) >> 2) >> 1$
8x8	$(t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + l_0 + l_1 + l_2 + l_3 + l_4 + l_5 + l_6 + l_7 + 8) >> 4$

b) If l_k, k = 0 to (M-1) are available and t_k, k = 0 to (N-1) are not available

block size	p
4x8	$(l_0 + l_1 + l_2 + l_3 + l_4 + l_5 + l_6 + l_7 + 4) >> 3$
8x4	$(l_0 + l_1 + l_2 + l_3 + 2) >> 2$
8x8	$(l_0 + l_1 + l_2 + l_3 + l_4 + l_5 + l_6 + l_7 + 4) >> 3$

c) If l_k, k = 0 to (M-1) are not available and t_k, k = 0 to (N-1) are available

block size	p
4x8	$(t_0 + t_1 + t_2 + t_3 + 2) >> 2$
8x4	$(t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + 4) >> 3$

8x8	$(t_0 + t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + 4) >> 3$

d) If l_k, k = 0 to (M-1) are not available and t_k, k = 0 to (N-1) are not available

block size	p
4x8	128
8x4	128
8x8	128

12.4.1.4 Mode 3: diagonal down/left prediction

t_k, k = 0 to (N-1) shall be available, and l_k, k = 0 to (M-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
p_{00}	p_{00}	p_{00}	$((l_2 + l_1<<1 + l_0 + 2) >> 2 + (t_0 + t_1<<1 + t_2 + 2) >> 2) >> 1$
p_{01}, p_{10}	p_{01}, p_{10}	p_{01}, p_{10}	$((l_3 + l_2<<1 + l_1 + 2) >> 2 + (t_1 + t_2<<1 + t_3 + 2) >> 2) >> 1$
p_{02}, p_{11}, p_{20}	p_{02}, p_{11}, p_{20}	p_{02}, p_{11}, p_{20}	$((l_4 + l_3<<1 + l_2 + 2) >> 2 + (t_2 + t_3<<1 + t_4 + 2) >> 2) >> 1$
p_{03}, p_{12}, p_{21}, p_{30}	p_{03}, p_{12}, p_{21}, p_{30}	p_{03}, p_{12}, p_{21}, p_{30}	$((l_5 + l_4<<1 + l_3 + 2) >> 2 + (t_3 + t_4<<1 + t_5 + 2) >> 2) >> 1$
p_{13}, p_{22}, p_{31}, p_{40}	p_{04}, p_{13}, p_{22}, p_{31}	p_{04}, p_{13}, p_{22}, p_{31}, p_{40}	$((l_6 + l_5<<1 + l_4 + 2) >> 2 + (t_4 + t_5<<1 + t_6 + 2) >> 2) >> 1$
p_{23}, p_{32}, p_{41}, p_{50}	p_{05}, p_{14}, p_{23}, p_{32}	p_{05}, p_{14}, p_{23}, p_{32}, p_{41}, p_{50}	$((l_7 + l_6<<1 + l_5 + 2) >> 2 + (t_5 + t_6<<1 + t_7 + 2) >> 2) >> 1$
p_{33}, p_{42}, p_{51}, p_{60}	p_{06}, p_{15}, p_{24}, p_{33}	p_{06}, p_{15}, p_{24}, p_{33}, p_{42}, p_{51}, p_{60}	$((l_8 + l_7<<1 + l_6 + 2) >> 2 + (t_6 + t_7<<1 + t_8 + 2) >> 2) >> 1$
p_{43}, p_{52}, p_{61}, p_{70}	p_{07}, p_{16}, p_{25}, p_{34}	p_{07}, p_{16}, p_{25}, p_{34}, p_{43}, p_{52}, p_{61}, p_{70}	$((l_9 + l_8<<1 + l_7 + 2) >> 2 + (t_7 + t_8<<1 + t_9 + 2) >> 2) >> 1$
p_{53}, p_{62}, p_{71}	p_{17}, p_{26}, p_{35}	p_{17}, p_{26}, p_{35}, , p_{44}, p_{53}, p_{62}, p_{71}	$((l_{10} + l_9<<1 + l_8 + 2) >> 2 + (t_8 + t_9<<1 + t_{10} + 2) >> 2) >> 1$
p_{63}, p_{72}	p_{27}, p_{36}	p_{27}, p_{36}, p_{45}, p_{54}, p_{63}, p_{72}	$((l_{11} + l_{10}<<1 + l_9 + 2) >> 2 + (t_9 + t_{10}<<1 + t_{11} + 2) >> 2) >> 1$
p_{73}	p_{37}	p_{37}, p_{46}, p_{55}, p_{64}, p_{73}	$((l_{12} + l_{11}<<1 + l_{10} + 2) >> 2 + (t_{10} + t_{11}<<1 + t_{12} + 2) >> 2) >> 1$
-	-	p_{47}, p_{56}, p_{65}, p_{74}	$((l_{13} + l_{12}<<1 + l_{11} + 2) >> 2 + (t_{11} + t_{12}<<1 + t_{13} + 2) >> 2) >> 1$
-	-	p_{57}, p_{66}, p_{75}	$((l_{14} + l_{13}<<1 + l_{12} + 2) >> 2 + (t_{12} + t_{13}<<1 + t_{14} + 2) >> 2) >> 1$
-	-	p_{67}, p_{76}	$((l_{15} + l_{14}<<1 + l_{13} + 2) >> 2 + (t_{13} + t_{14}<<1 + t_{15} + 2) >> 2) >> 1$
-	-	p_{77}	$((l_{15} + l_{15}<<1 + l_{14} + 2) >> 2 + (t_{14} + t_{15}<<1 + t_{15} + 2) >> 2) >> 1$

12.4.1.5 Mode 4: diagonal down/right prediction

t_k, k = 0 to (N-1) shall be available, and q, and l_k, k = 0 to (M-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
-	p_{07}	p_{07}	$(t_5 + t_6<<1 + t_7 + 2) >> 2$
-	p_{06}, p_{17}	p_{06}, p_{17}	$(t_4 + t_5<<1 + t_6 + 2) >> 2$
-	p_{05}, p_{16}, p_{27}	p_{05}, p_{16}, p_{27}	$(t_3 + t_4<<1 + t_5 + 2) >> 2$
-	p_{04}, p_{15}, p_{26}, p_{37}	p_{04}, p_{15}, p_{26}, p_{37}	$(t_2 + t_3<<1 + t_4 + 2) >> 2$
p_{03}	p_{03}, p_{14}, p_{25}, p_{36}	p_{03}, p_{14}, p_{25}, p_{36}, p_{47}	$(t_1 + t_2<<1 + t_3 + 2) >> 2$
p_{02}, p_{13}	p_{02}, p_{13}, p_{24}, p_{35}	p_{02}, p_{13}, p_{24}, p_{35}, p_{46}, p_{57}	$(t_0 + t_1<<1 + t_2 + 2) >> 2$
p_{01}, p_{12}, p_{23}	p_{01}, p_{12}, p_{23}, p_{34}	p_{01}, p_{12}, p_{23}, p_{34}, , p_{45}, p_{56}, p_{67}	$(q + t_0<<1 + t_1 + 2) >> 2$
p_{00}, p_{11}, p_{22}, p_{33}	p_{00}, p_{11}, p_{22}, p_{33}	p_{00}, p_{11}, p_{22}, p_{33}, p_{44}, p_{55}, p_{66}, p_{77}	$(l_0 + q<<1 + t_0 + 2) >> 2$

p_{10}, p_{21}, p_{32}, p_{43}	p_{10}, p_{21}, p_{32}	p_{10}, p_{21}, p_{32}, , p_{43}, p_{54}, p_{65}, p_{76}	$(l_1 + l_0<<1 + q + 2) >> 2$
p_{20}, p_{31}, p_{42}, p_{53}	p_{20}, p_{31}	p_{20}, p_{31}, p_{42}, p_{53}, p_{64}, p_{75}	$(l_2 + l_1<<1 + l_0 + 2) >> 2$
p_{30}, p_{41}, p_{52}, p_{63}	p_{30}	p_{30}, p_{41}, p_{52}, p_{63}, p_{74}	$(l_3 + l_2<<1 + l_1 + 2) >> 2$
p_{40}, p_{51}, p_{62}, p_{73}	-	p_{40}, p_{51}, p_{62}, p_{73}	$(l_4 + l_3<<1 + l_2 + 2) >> 2$
p_{50}, p_{61}, p_{72}	-	p_{50}, p_{61}, p_{72}	$(l_5 + l_4<<1 + l_3 + 2) >> 2$
p_{60}, p_{71}	-	p_{60}, p_{71}	$(l_6 + l_5<<1 + l_4 + 2) >> 2$
p_{70}	-	p_{70}	$(l_7 + l_6<<1 + l_5 + 2) >> 2$

12.4.1.6 Mode 5: vertical-left prediction

t_k, k = 0 to (N-1) shall be available, and q, and l_k, k = 0 to (M-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
-	p_{07}	p_{07}	$((t_6 + t_7<<1 + t_8 + 2) >> 2 + (t_7 + t_8<<1 + t_9 + 2) >> 2) >> 1$
-	p_{17}	p_{17}	$(t_6 + t_7<<1 + t_8 + 2) >> 2$
-	p_{06}, p_{27}	p_{06}, p_{27}	$((t_5 + t_6<<1 + t_7 + 2) >> 2 + (t_6 + t_7<<1 + t_8 + 2) >> 2) >> 1$
-	p_{16}, p_{37}	p_{16}, p_{37}	$(t_5 + t_6<<1 + t_7 + 2) >> 2$
-	p_{05}, p_{26}	p_{05}, p_{26}, p_{47}	$((t_4 + t_5<<1 + t_6 + 2) >> 2 + (t_5 + t_6<<1 + t_7 + 2) >> 2) >> 1$
-	p_{15}, p_{36}	p_{15}, p_{36}, p_{57}	$(t_4 + t_5<<1 + t_6 + 2) >> 2$
-	p_{04}, p_{25}	p_{04}, p_{25}, p_{46}, p_{67}	$((t_3 + t_4<<1 + t_5 + 2) >> 2 + (t_4 + t_5<<1 + t_6 + 2) >> 2) >> 1$
-	p_{14}, p_{35}	p_{14}, p_{35}, p_{56}, p_{77}	$(t_3 + t_4<<1 + t_5 + 2) >> 2$
p_{03}	p_{03}, p_{24}	p_{03}, p_{24}, p_{45}, p_{66}	$((t_2 + t_3<<1 + t_4 + 2) >> 2 + (t_3 + t_4<<1 + t_5 + 2) >> 2) >> 1$
p_{13}	p_{13}, p_{34}	p_{13}, p_{34}, p_{55}, p_{76}	$(t_2 + t_3<<1 + t_4 + 2) >> 2$
p_{02}, p_{23}	p_{02}, p_{23}	p_{02}, p_{23}, p_{44}, p_{65}	$((t_1 + t_2<<1 + t_3 + 2) >> 2 + (t_2 + t_3<<1 + t_4 + 2) >> 2) >> 1$
p_{12}, p_{33}	p_{12}, p_{33}	p_{12}, p_{33}, p_{54}, p_{75}	$(t_1 + t_2<<1 + t_3 + 2) >> 2$
p_{01}, p_{22}, p_{43}	p_{01}, p_{22}	p_{01}, p_{22}, p_{43}, p_{64}	$((t_0 + t_1<<1 + t_2 + 2) >> 2 + (t_1 + t_2<<1 + t_3 + 2) >> 2) >> 1$
p_{11}, p_{32}, p_{53}	p_{11}, p_{32}	p_{11}, p_{32}, p_{53}, p_{74}	$(t_0 + t_1<<1 + t_2 + 2) >> 2$
p_{00}, p_{21}, p_{42}, p_{63}	p_{00}, p_{21}	p_{00}, p_{21}, p_{42}, p_{63}	$((l_0 + q<<1 + t_0 + 2) >> 2 + (t_0 + t_1<<1 + t_2 + 2) >> 2) >> 1$
p_{10}, p_{31}, p_{52}, p_{73}	p_{10}, p_{31}	p_{10}, p_{31}, p_{52}, p_{73}	$(l_0 + q<<1 + t_0 + 2) >> 2$
p_{20}, p_{41}, p_{62}	p_{20}	p_{20}, p_{41}, p_{62}	$(l_2 + l_1<<1 + l_0 + 2) >> 2$
p_{30}, p_{51}, p_{72}	p_{30}	p_{30}, p_{51}, p_{72}	$(l_3 + l_2<<1 + l_1 + 2) >> 2$
p_{40}, p_{61}	-	p_{40}, p_{61}	$(l_4 + l_3<<1 + l_2 + 2) >> 2$
p_{50}, p_{71}	-	p_{50}, p_{71}	$(l_5 + l_4<<1 + l_3 + 2) >> 2$
p_{60}	-	p_{60}	$(l_6 + l_5<<1 + l_4 + 2) >> 2$
p_{70}	-	p_{70}	$(l_7 + l_6<<1 + l_5 + 2) >> 2$

12.4.1.7 Mode 6: horizontal-down prediction

t_k, k = 0 to (N-1) shall be available, and q, and l_k, k = 0 to (M-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
-	p_{07}	p_{07}	$(t_5 + t_6<<1 + t_7 + 2) >> 2$

-	p_{06}	p_{06}	$(t_4 + t_5 \ll 1 + t_6 + 2) \gg 2$
-	p_{05}, p_{17}	p_{05}, p_{17}	$(t_3 + t_4 \ll 1 + t_5 + 2) \gg 2$
-	p_{04}, p_{16}	p_{04}, p_{16}	$(t_2 + t_3 \ll 1 + t_4 + 2) \gg 2$
p_{03}	p_{03}, p_{15}, p_{27}	p_{03}, p_{15}, p_{27}	$(t_1 + t_2 \ll 1 + t_3 + 2) \gg 2$
p_{02}	p_{02}, p_{14}, p_{26}	p_{02}, p_{14}, p_{26}	$(t_0 + t_1 \ll 1 + t_2 + 2) \gg 2$
p_{01}, p_{13}	p_{01}, p_{13}, p_{25}, p_{37}	p_{01}, p_{13}, p_{25}, p_{37}	$(q + t_0 \ll 1 + t_1 + 2) \gg 2$
p_{00}, p_{12}	p_{00}, p_{12}, p_{24}, p_{36}	p_{00}, p_{12}, p_{24}, p_{36}	$((l_0 + q \ll 1 + t_0 + 2) \gg 2 + (q + l_0 \ll 1 + l_1 + 2) \gg 2) \gg 1$
p_{10}, p_{22}	p_{10}, p_{22}, p_{34}	p_{10}, p_{22}, p_{34}, p_{46}	$((q + l_0 \ll 1 + l_1 + 2) \gg 2 + (l_0 + l_1 \ll 1 + l_2 + 2) \gg 2) \gg 1$
p_{11}, p_{23}	p_{11}, p_{23}, p_{35}	p_{11}, p_{23}, p_{35}, p_{47}	$(q + l_0 \ll 1 + l_1 + 2) \gg 2$
p_{20}, p_{32}	p_{20}, p_{32}	p_{20}, p_{32}, p_{44}, p_{56}	$((l_0 + l_1 \ll 1 + l_2 + 2) \gg 2 + (l_1 + l_2 \ll 1 + l_3 + 2) \gg 2) \gg 1$
p_{21}, p_{33}	p_{21}, p_{33}	p_{21}, p_{33}, p_{45}, p_{57}	$(l_0 + l_1 \ll 1 + l_2 + 2) \gg 2$
p_{30}, p_{42}	p_{30}	p_{30}, p_{42}, p_{54}, p_{66}	$((l_1 + l_2 \ll 1 + l_3 + 2) \gg 2 + (l_2 + l_3 \ll 1 + l_4 + 2) \gg 2) \gg 1$
p_{31}, p_{43}	p_{31}	p_{31}, p_{43}, p_{55}, p_{67}	$(l_1 + l_2 \ll 1 + l_3 + 2) \gg 2$
p_{40}, p_{52}	-	p_{40}, p_{52}, p_{64}, p_{76}	$((l_2 + l_3 \ll 1 + l_4 + 2) \gg 2 + (l_3 + l_4 \ll 1 + l_5 + 2) \gg 2) \gg 1$
p_{41}, p_{53}	-	p_{41}, p_{53}, p_{65}, p_{77}	$(l_2 + l_3 \ll 1 + l_4 + 2) \gg 2$
p_{50}, p_{62}	-	p_{50}, p_{62}, p_{74}	$((l_3 + l_4 \ll 1 + l_5 + 2) \gg 2 + (l_4 + l_5 \ll 1 + l_6 + 2) \gg 2) \gg 1$
p_{51}, p_{63}	-	p_{51}, p_{63}, p_{75}	$(l_3 + l_4 \ll 1 + l_5 + 2) \gg 2$
p_{60}, p_{72}	-	p_{60}, p_{72}	$((l_4 + l_5 \ll 1 + l_6 + 2) \gg 2 + (l_5 + l_6 \ll 1 + l_7 + 2) \gg 2) \gg 1$
p_{61}, p_{73}	-	p_{61}, p_{73}	$(l_4 + l_5 \ll 1 + l_6 + 2) \gg 2$
p_{70}	-	p_{70}	$((l_5 + l_6 \ll 1 + l_7 + 2) \gg 2 + (l_6 + l_7 \ll 1 + l_8 + 2) \gg 2) \gg 1$
p_{71}	-	p_{71}	$(l_5 + l_6 \ll 1 + l_7 + 2) \gg 2$

12.4.1.8 Mode 7: vertical-right prediction

t_k, k = 0 to (N-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
p_{00}	p_{00}	p_{00}	$((t_0 + t_1 \ll 1 + t_2 + 2) \gg 2 + (t_1 + t_2 \ll 1 + t_3 + 2) \gg 2) \gg 1$
p_{10}	p_{10}	p_{10}	$(t_1 + t_2 \ll 1 + t_3 + 2) \gg 2$
p_{01}, p_{20}	p_{01}, p_{20}	p_{01}, p_{20}	$((t_1 + t_2 \ll 1 + t_3 + 2) \gg 2 + (t_2 + t_3 \ll 1 + t_4 + 2) \gg 2) \gg 1$
p_{11}, p_{30}	p_{11}, p_{30}	p_{11}, p_{30}	$(t_2 + t_3 \ll 1 + t_4 + 2) \gg 2$
p_{02}, p_{21}, p_{40}	p_{02}, p_{21}	p_{02}, p_{21}, p_{40}	$((t_2 + t_3 \ll 1 + t_4 + 2) \gg 2 + (t_3 + t_4 \ll 1 + t_5 + 2) \gg 2) \gg 1$
p_{12}, p_{31}, p_{50}	p_{12}, p_{31}	p_{12}, p_{31}, p_{50}	$(t_3 + t_4 \ll 1 + t_5 + 2) \gg 2$
p_{03}, p_{22}, p_{41}, p_{60}	p_{03}, p_{22}	p_{03}, p_{22}, p_{41}, p_{60}	$((t_3 + t_4 \ll 1 + t_5 + 2) \gg 2 + (t_4 + t_5 \ll 1 + t_6 + 2) \gg 2) \gg 1$
p_{13}, p_{32}, p_{51}, p_{70}	p_{13}, p_{32}	p_{13}, p_{32}, p_{51}, p_{70}	$(t_4 + t_5 \ll 1 + t_6 + 2) \gg 2$
p_{23}, p_{42}, p_{61}	p_{04}, p_{23}	p_{04}, p_{23}, p_{42}, p_{61}	$((t_0 + t_1 \ll 1 + t_2 + 2) \gg 2 + (t_1 + t_2 \ll 1 + t_3 + 2) \gg 2) \gg 1$
p_{33}, p_{52}, p_{71}	p_{14}, p_{33}	p_{14}, p_{33}, p_{52}, p_{71}	$(t_5 + t_6 \ll 1 + t_7 + 2) \gg 2$
p_{43}, p_{62}	p_{05}, p_{24}	p_{05}, p_{24}, p_{43}, p_{62}	$((t_5 + t_6 \ll 1 + t_7 + 2) \gg 2 + (t_6 + t_7 \ll 1 + t_8 + 2) \gg 2) \gg 1$
p_{53}, p_{72}	p_{15}, p_{34}	p_{15}, p_{34}, p_{53}, p_{72}	$(t_6 + t_7 \ll 1 + t_8 + 2) \gg 2$

p_{63}	p_{06}, p_{25}	p_{06}, p_{25}, p_{44}, p_{63}	$((t_6 + t_7<<1 + t_8 + 2) >> 2 + (t_7 + t_8<<1 + t_9 + 2) >> 2) >> 1$
p_{73}	p_{16}, p_{35}	p_{16}, p_{35}, p_{54}, p_{73}	$(t_7 + t_8<<1 + t_9 + 2) >> 2$
-	p_{07}, p_{26}	p_{07}, p_{26}, p_{45}, p_{64}	$((t_7 + t_8<<1 + t_9 + 2) >> 2 + (t_8 + t_9<<1 + t_{10} + 2) >> 2) >> 1$
-	p_{17}, p_{36}	p_{17}, p_{36}, p_{55}, p_{74}	$(t_8 + t_9<<1 + t_{10} + 2) >> 2$
-	p_{27}	p_{27}, p_{46}, p_{65}	$((t_8 + t_9<<1 + t_{10} + 2) >> 2 + (t_9 + t_{10}<<1 + t_{11} + 2) >> 2) >> 1$
-	p_{37}	p_{37}, p_{56}, p_{75}	$(t_9 + t_{10}<<1 + t_{11} + 2) >> 2$
-	-	p_{47}, p_{66}	$((t_9 + t_{10}<<1 + t_{11} + 2) >> 2 + (t_{10} + t_{11}<<1 + t_{12} + 2) >> 2) >> 1$
-	-	p_{57}, p_{76}	$(t_{10} + t_{11}<<1 + t_{12} + 2) >> 2$
-	-	p_{67}	$((t_{10} + t_{11}<<1 + t_{12} + 2) >> 2 + (t_{11} + t_{12}<<1 + t_{13} + 2) >> 2) >> 1$
-	-	p_{77}	$(t_{11} + t_{12}<<1 + t_{13} + 2) >> 2$

12.4.1.9 Mode 8: horizontal-up prediction

l_k, k = 0 to (M-1) shall be available.

4x8 block samples	8x4 block samples	8x8 block samples	predicted by
p_{00}	p_{00}	p_{00}	$((l_0 + l_1<<1 + l_2 + 2) >> 2 + (l_1 + l_2<<1 + l_3 + 2) >> 2) >> 1$
p_{01}	p_{01}	p_{01}	$(l_1 + l_2<<1 + l_3 + 2) >> 2$
p_{10}, p_{02}	p_{10}, p_{02}	p_{10}, p_{02}	$((l_1 + l_2<<1 + l_3 + 2) >> 2 + (l_2 + l_3<<1 + l_4 + 2) >> 2) >> 1$
p_{11}, p_{03}	p_{11}, p_{03}	p_{11}, p_{03}	$(l_2 + l_3<<1 + l_4 + 2) >> 2$
p_{20}, p_{12}	p_{20}, p_{12}, p_{04}	p_{20}, p_{12}, p_{04}	$((l_2 + l_3<<1 + l_4 + 2) >> 2 + (l_3 + l_4<<1 + l_5 + 2) >> 2) >> 1$
p_{21}, p_{13}	p_{21}, p_{13}, p_{05}	p_{21}, p_{13}, p_{05}	$(l_3 + l_4<<1 + l_5 + 2) >> 2$
p_{30}, p_{22}	p_{30}, p_{22}, p_{14}, p_{06}	p_{30}, p_{22}, p_{14}, p_{06}	$((l_3 + l_4<<1 + l_5 + 2) >> 2 + (l_4 + l_5<<1 + l_6 + 2) >> 2) >> 1$
p_{31}, p_{23}	p_{31}, p_{23}, p_{15}, p_{07}	p_{31}, p_{23}, p_{15}, p_{07}	$(l_4 + l_5<<1 + l_6 + 2) >> 2$
p_{40}, p_{32}	p_{32}, p_{24}, p_{16}	p_{40}, p_{32}, p_{24}, p_{16}	$((l_4 + l_5<<1 + l_6 + 2) >> 2 + (l_5 + l_6<<1 + l_7 + 2) >> 2) >> 1$
p_{41}, p_{33}	p_{33}, p_{25}, p_{17}	p_{41}, p_{33}, p_{25}, p_{17}	$(l_5 + l_6<<1 + l_7 + 2) >> 2$
P_{50}, p_{42}	p_{34}, p_{26}	p_{50}, p_{42}, p_{34}, p_{26}	$((l_5 + l_6<<1 + l_7 + 2) >> 2 + (l_6 + l_7<<1 + l_8 + 2) >> 2) >> 1$
P_{51}, p_{43}	p_{35}, p_{27}	p_{51}, p_{43}, p_{35}, p_{27}	$(l_6 + l_7<<1 + l_8 + 2) >> 2$
P_{60}, p_{52}	p_{36}	p_{60}, p_{52}, p_{44}, p_{36}	$((l_6 + l_7<<1 + l_8 + 2) >> 2 + (l_7 + l_8<<1 + l_9 + 2) >> 2) >> 1$
P_{61}, p_{53}	p_{37}	p_{61}, p_{53}, p_{45}, p_{37}	$(l_7 + l_8<<1 + l_9 + 2) >> 2$
P_{70}, p_{62}	-	p_{70}, p_{62}, p_{54}, p_{46}	$((l_7 + l_8<<1 + l_9 + 2) >> 2 + (l_8 + l_9<<1 + l_{10} + 2) >> 2) >> 1$
P_{71}, p_{63}	-	p_{71}, p_{63}, p_{55}, p_{47}	$(l_8 + l_9<<1 + l_{10} + 2) >> 2$
P_{72}	-	p_{72}, p_{64}, p_{56}	$((l_8 + l_9<<1 + l_{10} + 2) >> 2 + (l_9 + l_{10}<<1 + l_{11} + 2) >> 2) >> 1$
P_{73}	-	p_{73}, p_{65}, p_{57}	$(l_9 + l_{10}<<1 + l_{11} + 2) >> 2$
-	-	p_{74}, p_6	$((l_9 + l_{10}<<1 + l_{11} + 2) >> 2 + (l_{10} + l_{11}<<1 + l_{12} + 2) >> 2) >> 1$
-	-	p_{75}, p_{67}	$(l_{10} + l_{11}<<1 + l_{12} + 2) >> 2$
-	-	p_{76}	$((l_{10} + l_{11}<<1 + l_{12} + 2) >> 2 + (l_{11} + l_{12}<<1 + l_{13} + 2) >> 2) >> 1$
-	-	p_{77}	$(l_{11} + l_{12}<<1 + l_{13} + 2) >> 2$

12.4.2 Scanning method for ABT blocks

Scanning patterns for blocks of size 4x4, 4x8, 8x4, and 8x8 coefficients are given below. For blocks decoded in frame mode, the zig-zag scans are used. For block decoded in field mode, the field scans are used. The zig-zag scan for 4x4 blocks corresponds to the zig-zag scan specified in Figure 8-12.

12.4.2.1 Zig-zag scan

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

Figure 12-3 – 4x4 zig-zag scan

0	2	3	9
1	4	8	10
5	7	11	17
6	12	16	18
13	15	19	25
14	20	24	26
21	23	27	30
22	28	29	31

Figure 12-4 – 4x8 zig-zag scan

0	1	5	6	13	14	21	22
2	4	7	12	15	20	23	28
3	8	11	16	19	24	27	29
9	10	17	18	25	26	30	31

Figure 12-5 – 8x4 zig-zag scan

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

Figure 12-6 – 8x8 zig-zag scan

12.4.2.2 Field scan

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

Figure 12-7 – 4x4 field scan

0	4	12	20
1	5	13	21
2	6	14	22
3	11	19	27
7	15	23	28
8	16	24	29
9	17	25	30
10	18	26	31

Figure 12-8 – 4x8 field scan

0	2	6	10	14	18	22	26
1	5	9	13	17	21	25	29
3	7	11	15	19	23	27	30
4	8	12	16	20	24	28	31

Figure 12-9 – 8x4 field scan

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

Figure 12-10 – 8x8 field scan

12.4.3 Scaling and inverse transform for ABT blocks

The scaling and inverse transform of residual blocks of block size larger than 4x4 is specified below. The scaling and inverse transform for 4x4 blocks is specified in subclause 8.6. For 8x8 blocks, the coefficients $R_{ij}^{(m)}$, used in the formulas below, are specified as:

$$R_{ij}^{(m)} = V_m^{8x8} \tag{12-3}$$

where the subscript of V^{8x8} is the row index of the vector specified as:

$$V^{8x8} = \begin{bmatrix} 15 \\ 17 \\ 19 \\ 22 \\ 24 \\ 27 \end{bmatrix}. \tag{12-4}$$

For 4x8 blocks, the coefficients $R_{ij}^{(m)}$, used in the formulas below, are specified as:

$$R_{ij}^{(m)} = \begin{cases} V_{m0}^{8x4,4x8} & \text{for} \quad i = 0,...,7;\ j = 0,2 \\ V_{m1}^{8x4,4x8} & \text{for} \quad i = 0,...,7;\ j = 1,3 \end{cases} \tag{12-5}$$

where the first and second subscripts of $V^{8x4,4x8}$ are row and column indices, respectively, of the matrix specified as:

$$V^{8x4,4x8} = \begin{bmatrix} 9 & 11 \\ 10 & 12 \\ 11 & 14 \\ 12 & 16 \\ 14 & 17 \\ 15 & 20 \end{bmatrix}. \tag{12-6}$$

For 8x4 blocks, the coefficients $R_{ij}^{(m)}$, used in the formulas below, are specified as:

$$R_{ij}^{(m)} = \begin{cases} V_{m0}^{8x4,4x8} & \text{for} \quad i = 0,2;\ j = 0,\ldots,7 \\ V_{m1}^{8x4,4x8} & \text{for} \quad i = 1,3;\ j = 0,\ldots,7 \end{cases} \tag{12-7}$$

where the first and second subscripts of $V^{8x4,4x8}$ are row and column indices, respectively, of the matrix specified in Equation 12-6.

The coefficient levels are multiplied with the scaling value R

$$w_{ij} = [c_{ij} \cdot R_{ij}^{(QP\%6)}] << (QP/6 - 2), \quad i = 0,\ldots,N,\ j = 0,\ldots,M \tag{12-8}$$

After constructing an entire MxN block of scaled transform coefficients and assembling these into a MxN matrix W of elements w_{ij} illustrated as

$$W = \begin{bmatrix} w_{00} & \cdots & w_{0(N-1)} \\ \vdots & \ddots & \vdots \\ w_{(M-1)0} & \cdots & w_{(M-1)(N-1)} \end{bmatrix} \tag{12-9}$$

W is inverse transformed horizontally. If N = = 4, the one-dimensional inverse transform is performed as specified in subclause 8.6.2.3. If N = = 8, the inverse transform is specified by Equation 12-10,

$$Z' = \begin{bmatrix} w_{00} & \cdots & w_{07} \\ \vdots & \ddots & \vdots \\ w_{(M-1)0} & \cdots & w_{(M-1)7} \end{bmatrix} \begin{bmatrix} 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 \\ 19 & 15 & 9 & 3 & -3 & -9 & -15 & -19 \\ 17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 \\ 9 & 3 & -19 & -15 & 15 & 19 & -3 & -9 \\ 13 & -13 & -13 & 13 & 13 & -13 & -13 & 13 \\ 15 & -19 & -3 & 9 & -9 & 3 & 19 & -15 \\ 7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 \\ 3 & -9 & 15 & -19 & 19 & -15 & 9 & -3 \end{bmatrix} \tag{12-10}$$

The result is rounded

$$Z_{ij} = \mathrm{sign}(Z'_{ij})\left[\mathrm{abs}(Z'_{ij}) + 2^{B_{\mathrm{shift}}-1}\right] >> B_{\mathrm{shift}} \tag{12-11}$$

where B_{shift} = = 7 for 8x8 blocks, and B_{shift} = = 2 for 4x8 or 8x4 blocks. If N = = 4, the second one-dimensional inverse transform is performed as specified in subclause 8.6.2.3. If N = = 8, the second inverse transform is specified by Equation 12-12 below,

$$X' = \begin{bmatrix} 13 & 19 & 17 & 9 & 13 & 15 & 7 & 3 \\ 13 & 15 & 7 & 3 & -13 & -19 & -17 & -9 \\ 13 & 9 & -7 & -19 & -13 & -3 & 17 & 15 \\ 13 & 3 & -17 & -15 & 13 & 9 & -7 & -19 \\ 13 & -3 & -17 & 15 & 13 & -9 & -7 & 19 \\ 13 & -9 & -7 & 19 & -13 & 3 & 17 & -15 \\ 13 & -15 & 7 & -3 & -13 & 19 & -17 & 9 \\ 13 & -19 & 17 & -9 & 13 & -15 & 7 & -3 \end{bmatrix} \begin{bmatrix} z_{00} & \cdots & z_{0(N-1)} \\ \vdots & \ddots & \vdots \\ z_{70} & \cdots & z_{7(N-1)} \end{bmatrix} \tag{12-12}$$

After the second (vertical) transform in Equation 12-12 step the final reconstructed sample residual values X'' shall be obtained as specified in Equation 8-59.

Finally, the reconstructed sample residual values X'' from Equation 8-59 are added to the prediction values P_{ij} from motion-compensated prediction or spatial prediction and clipped to the range of 0 to 255 to form the final decoded sample result prior to application of the deblocking filter as specified in subclause 8.6.3.

Deleted: compensated

12.4.4 Modifications for the deblocking filter

If ABT is used, the boundary strength shall be Bs = 0 for all 4x4 luma block edges inside an ABT block. The index into the threshold Table (Table 8-3) is increased by I_{QP}. For ABT blocks, I_{QP} depends on the sizes of the neighbouring blocks as specified in Table 12-4. I_{QP} = 0 for non-ABT blocks.

Table 12-4 – I_{QP} values

I_{QP}		Block q			
		4x4	4x8	8x4	8x8
Block p	4x4	0	1	1	2
	4x8	1	2	2	3
	8x4	1	2	2	3
	8x8	2	3	3	3

The index used to access the α-table, as well as the C0-Table that is used in the default filter mode, is computed as:

$$\mathrm{Index}_A = \mathrm{Clip3}(0, 51, QP_{av} + I_{QP} + \mathrm{Filter\_Offset\_A})$$

The index used to access the β-Table is computed as:

$$\mathrm{Index}_B = \mathrm{Clip3}(0, 51, QP_{av} + I_{QP} + \mathrm{Filter\_Offset\_B}),$$

with QP_{av}, Filter_Offset_A, and Filter_Offset_B as specified in subclause 8.7.2. The values for the thresholds (α and β) are specified in Table 8-3.

12.5 ABT entropy coding

12.5.1 ABT variable length coding

12.5.1.1 Mapped Exp-Golomb entropy coding

The ABT intra macroblock types in Intra slices and the intra_block_typeABT syntax elements in Inter slices are to Exp-Golomb codeword numbers as specified in Table 12-5.

Table 12-5 – Assignment of Exp-Golomb codeword numbers for ABT syntax elements

code_number	mb_type	intra_block_typeABT
0	ABTIntra_4x4	4x4
1	ABTIntra_4x8	4x8
2	ABTIntra_8x4	8x4
3	ABTIntra_8x8	8x8

12.5.1.2 VLC entropy coding of ABT coefficients

12.5.1.2.1 Decoding num_coeff_abt

For ABT Intra blocks, num_coeff_abt is specified. The structure of the codewords for num_coeff and escape_run, specified in subclause 12.4.2.4???, is indicated in Table 12-6. The info bits xi, i=0 to n can take values 0 or 1.

Table 12-6 – Code structure for ABT num_coeff_abt and escape_run

codeword	length L
1 x1 x0	3
0 1 x2 x1 x0	5
0 0 1 x3 x2 x1 x0	7
0 0 0 1 x4 x3 x2 x1 x0	9
0 0 0 0 1 x5 x4 x3 x2 x1 x0	11

The value of num_coeff is specified as the code number of the decoded codeword. The code number is specified as

$$\text{code\_number} = 2^{(L+1)/2} - 4 + \text{INFO} \qquad (12\text{-}13)$$

For a codeword with info bits xi, i=0 to n, INFO is specified as

$$\text{INFO} = \sum_{i=0}^{n} \text{x}i \cdot 2^{i}\ . \qquad (12\text{-}14)$$

12.5.1.2.2 2D (level,run) symbols

The code structure used for decoding (level,run) symbols depends on block type. The structure of the codes is specified in Table 12-7. For all block types, the codewords with code numbers 0 to 59 are used, with code number 59 being the escape symbol. INFO is specified in Equation 12-14

Table 12-7 – Code structure for ABT (level, run) symbols

Block type	Code structure	length L	code_number
Intra 8x8, 8x4, 4x8, 4x4 Inter 8x8	1 x1 x0	3	$2^{(L+2)/2} - 4$ + INFO
	0 1 x2 x1 x0	5	
	0 0 1 x3 x2 x1 x0	7	
	0 0 0 x4 x3 x2 x1 x0	8	
Inter 8x4, 4x8	1 x0	2	$2^{(L+1)/2} - 2$ + INFO
	0 1 x1 x0	4	
	0 0 1 x2 x1 x0	6	
	0 0 0 1 x3 x2 x1 x0	8	
	0 0 0 0 x4 x3 x2 x1 x0	9	
Inter 4x4	1	1	0
	0 1 x0	3	$2^{L/2} - 1$ + INFO
	0 0 1 x1 x0	5	
	0 0 0 1 x2 x1 x0	7	
	0 0 0 0 1 x3 x2 x1 x0	9	
	0 0 0 0 0 x4 x3 x2 x1 x0	10	$2^{(L+1)/2} - 1$ + INFO

12.5.1.2.3 Assignment of level and run to code numbers

For positive level, the assignment of code numbers to run and level is specified in Table 12-9. Run and negative levels are assigned as follows

$$\text{code\_number(-abs(level), run) = code\_number(abs(level), run) + 1.} \tag{12-15}$$

12.5.1.2.4 escape_level and escape_run

The code structure for escape_level is specified in Table 12-8. The code number of a decoded codeword is

$$\text{code\_number} = 2^{(L+2)/2} - 8 + \text{INFO}, \tag{12-16}$$

with INFO as specified in Equation 12-14. The assignment of code numbers to escape_level is specified as follows:

```
if( ( code_number % 2) > 0 )
  escape_level = -(code_number/2)
else
  escape_level = code_number/2                                   (12-17)
```

The code structure for escape_run is specified in Table 12-6. The value of escape_run is specified as the code number of the decoded codeword. The code number for escape_run decoding is specified in Equation 12-13.

Table 12-8 – Code structure for escape_level

Codeword	length L
1 x2 x1 x0	4
0 1 x3 x2 x1 x0	6
0 0 1 x4 x3 x2 x1 x0	8
0 0 0 1 x5 x4 x3 x2 x1 x0	10
0 0 0 0 1 x6 x5 x4 x3 x2 x1 x0	12
0 0 0 0 0 1 x7 x6 x5 x4 x3 x2 x1 x0	14
0 0 0 0 0 0 1 x8 x7 x6 x5 x4 x3 x2 x1 x0	16
0 0 0 0 0 0 0 1 x9 x8 x7 x6 x5 x4 x3 x2 x1 x0	18

Table 12-9 – Assignment of Inter and Intra level and run to code numbers.

	Inter								Intra																				
	EOB	level>0							level>0 (QP < 26)							level>0 (26 <= QP < 34)							level>0 (QP >= 34)						
run		1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1	2	3	4	5	6	7
0	0	1	5	13	21	31	39	47	0	2	6	8	12	18	20	0	2	8	12	18	22	30	0	4	12	20	32	42	56
1	-	3	15	33	51	-	-	-	4	14	24	30	38	44	49	4	16	26	36	48	56	-	2	16	30	48	-	-	-
2	-	7	25	53	-	-	-	-	10	28	40	50	-	-	-	6	24	42	-	-	-	-	6	24	46	-	-	-	-
3	-	9	35	-	-	-	-	-	16	34	54	-	-	-	-	10	34	-	-	-	-	-	8	34	-	-	-	-	-
4	-	11	45	-	-	-	-	-	22	42	-	-	-	-	-	16	44	-	-	-	-	-	10	40	-	-	-	-	-
5	-	17	55	-	-	-	-	-	26	56	-	-	-	-	-	20	5	-	-	-	-	-	14	50	-	-	-	-	-
6	-	19	-	-	-	-	-	-	32	-	-	-	-	-	-	28	-	-	-	-	-	-	18	-	-	-	-	-	-
7	-	23	-	-	-	-	-	-	36	-	-	-	-	-	-	32	-	-	-	-	-	-	22	-	-	-	-	-	-
8	-	27	-	-	-	-	-	-	46	-	-	-	-	-	-	38	-	-	-	-	-	-	26	-	-	-	-	-	-

9	-	29	-	-	-	-	-	-	52	-	-	-	-	-	-	40	-	-	-	-	-	-	28	-	-	-	-	-	-
10	-	37	-	-	-	-	-	-	-	-	-	-	-	-	-	46	-	-	-	-	-	-	36	-	-	-	-	-	-
11	-	41	-	-	-	-	-	-	-	-	-	-	-	-	-	52	-	-	-	-	-	-	38	-	-	-	-	-	-
12	-	43	-	-	-	-	-	-	-	-	-	-	-	-	-	54	-	-	-	-	-	-	44	-	-	-	-	-	-
13	-	49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	52	-	-	-	-	-	-
14	-	57	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	54	-	-	-	-	-	-

12.5.2 ABT CABAC

12.5.2.1 Fixed-length (FL) binarization for mb_type

Table 12-10shows the binarization scheme used for decoding of macroblock types in I-slices. In case of mb_type = 6 for P slices or mb_type = 23 for B slices, intra_block_typeABT has to be decoded as additional information related to the chosen intra mode for these macroblock types. This shall be done in the same way as specified for mb_type in I slices.

Table 12-10 – Binarization for macroblock type

Slice type	Code number for mb_type	Binarization							
I slice	0 (ABTIntra_4x4)	1	0						
	1 (ABTIntra_4x8)	0	0						
	2 (ABTIntra_8x4)	0	1						
	3 (ABTIntra_8x8)	1	1						

12.5.2.2 Context definition and assignment

Table 12-11provides the context identifier associated to the syntax element macroblock type. A detailed description of the corresponding context variables is given in the subsequent subclauses. For the syntax elements related to decoding of transform coefficients, each of the context identifiers utilizes a separate set of ranges depending on whether the additional context categories 5 – 7 given in Table 12-14are used, which is only the case if MbABTFlag=1.

Table 12-11 – Macroblock type and associated context identifier

Syntax element	Context identifier	Type of Binarization	max_idx_ctx_id	Range of context label
Macroblock type	ctx_mb_type_I_ABT	Table 10-16	2	0 – 4
Transform coefficients	ctx_cbp4	-/-	-/-	75 – 94, 267 – 274
	ctx_sig	-/-	-/-	95 – 155, 275 – 319
	ctx_last	-/-	-/-	156 – 216, 320 – 346
	ctx_abs_level	UEG0, UCoff=14	2	217 – 266, 347 – 376

12.5.2.2.1 Assignment of context labels

Table 12-12and Table 12-13contain context identifiers along with their corresponding range of context labels. The association of context labels (modulo some offset) and bin numbers shows which context variable uses a fixed model and which one implies a choice of different models. The latter are characterized by those entries where a set of different context labels are given for a specific bin_no (Table 12-12) or block type dependent context_category (Table 12-13); these are the context variables, which need to be specified further in the following subclauses 12.5.2.2.2and 12.5.2.2.3.

Table 12-12 – Context identifiers and associated context labels

Context identifier	Range of context label	Offset for context label	max_idx_ctx_id	bin_no				
				1	2	3	4	≥ 5
ctx_mb_type_I_ABT	0 – 4	0	2	0,1,2	3,4	-/-	-/-	-/-
ctx_abs_level (context_category 5 – 7)	347 – 376	347+ 10*(context_category–5)	2	0 to 4	5 to 9	5 to 9	5 to 9	5 to 9

Table 12-13 – Context identifiers and associated context labels (continued)

Context identifier	Offset (range) of context label for context_category 5 – 7	context_category of block_type							
		0	1	2	3	4	5	6	7
ctx_cbp4	267 (267 – 274)	0 – 3	4 – 7	8 – 11	12 – 15	16 – 19	-/-	0 – 3	4 – 7
ctx_sig	275 (275 – 319)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60	0 – 14	15 –29	30 – 44
ctx_last	320 (320 – 346)	0 – 14	15 – 28	29 – 43	44 –46	47 – 60	0 – 9	10 –19	20 – 29

12.5.2.2.2 Context definitions using preceding bin values

For the context identifier ctx_mb_type_I_ABT, the choice of the model for the 2nd bin depends on the value of the first bin as specified in Equation 12-18:

$$\text{ctx\_mb\_type\_I\_ABT}[2] = (b1 == 0)\ ?\ 3 : 4 \tag{12-18}$$

Table 12-14 – Additional context categories for the different block types

block_type	Maximum number of coefficients	context_category
Luma block for INTRA 8x8 mode	64	5:Luma-8x8
Luma block for INTER 8x8 mode	64	
Luma block for INTRA 8x4 mode	32	6:Luma-8x4
Luma block for INTER 8x4 mode	32	
Luma block for INTRA 4x8 mode	32	7:Luma-4x8
Luma block for INTRA 4x8 mode	32	

12.5.2.2.3 Additional context definitions for information related to transform coefficients

As specified in subclause 9.2.2.4, three different additional context identifiers are used for conditioning of information related to transform coefficients. All these three types depend on context categories of different block types denoted by the variable *context_category*. The definition of these context categories is given in Table 9-24and Table 12-14. Note that the context categories 5 – 7 are only used in the case when MbABTFlag = 1. The context identifiers ctx_sig and ctx_last are related to the binary valued information of SIG and LAST; the definition of the related context variables includes an additional dependency on the scanning position *scanning_pos* within the regarded block:

$$ctx\_sig[scanning\_pos] = Map\_sig(\ scanning\_pos), \tag{12-19}$$

$$ctx\_last[scanning\_pos] = Map\_last(scanning\_pos). \qquad (12\text{-}20)$$

The definition of Map_sig and Map_last in Equations 12-19 and 12-20 depends on the block type. For context_category 0 – 4 the corresponding maps are given by

$$Map\_sig(scanning\_pos) = Map\_list(scanning\_pos) = scanning\_pos, \quad \text{if } context\_category = 0,\ldots,4, \qquad (12\text{-}21)$$

where scanning_pos denotes the position related to the zig-zag scan. For context categories 5 – 7, which are only in use if MbABTFlag = 1, two cases are distinguished. In frame coding mode, where the zig-zag scan is used, Map_sig and Map_last are given by the definition in Table 12-15; for field coding mode, Map_sig and Map_last related to the alternative scans are given in Tabel 12-16. In each case, the offset for the context category given in Table 12-17has to be added for calculating the context label of each scanning position.

For abs_level_m1, the decoding process is specified in subclause 9.2.2.4.

Table 12-15 – *Map_sig* and *Map_last* for zig-zag scanning order used for the additional ABT block sizes 8x8, 8x4 and 4x8

Scanning position	8x8		Scanning position	8x8		Scanning position	8x4 and 4x8	
	Map_sig	Map_last		Map_sig	Map_last		Map_sig	Map_last
0	0	0	32	7	3	0	0	0
1	1	1	33	6	3	1	1	1
2	2	1	34	11	3	2	2	1
3	3	1	35	12	3	3	3	1
4	4	1	36	13	3	4	4	1
5	5	1	37	11	3	5	5	1
6	5	1	38	6	3	6	7	1
7	4	1	39	7	3	7	8	1
8	4	1	40	8	4	8	9	2
9	3	1	41	9	4	9	10	2
10	3	1	42	14	4	10	11	2
11	4	1	43	10	4	11	9	2
12	4	1	44	9	4	12	8	2
13	4	1	45	8	4	13	6	2
14	5	1	46	6	4	14	7	2
15	5	1	47	11	4	15	8	2
16	4	2	48	12	5	16	9	3
17	4	2	49	13	5	17	10	3
18	4	2	50	11	5	18	11	3
19	4	2	51	6	5	19	9	3
20	3	2	52	9	6	20	8	4
21	3	2	53	14	6	21	6	4
22	6	2	54	10	6	22	12	4
23	7	2	55	9	6	23	8	4

24	7	2	56	11	7	24	9	5
25	7	2	57	12	7	25	10	5
26	8	2	58	13	7	26	11	6
27	9	2	59	11	7	27	9	6
28	10	2	60	14	8	28	13	7
29	9	2	61	10	8	29	13	7
30	8	2	62	12	8	30	14	8
31	7	2	63			31		

Table 12-16 – *Map_sig* and *Map_last* for field-based scanning order used for the additional ABT block sizes 8x8, 8x4 and 4x8

Scanning position	8x8		Scanning position	8x8		Scanning position	8x4 and 4x8	
	Map_sig	Map_last		Map_sig	Map_last		Map_sig	Map_last
0	0	0	32	9	3	0	0	0
1	1	1	33	9	3	1	1	1
2	1	1	34	10	3	2	2	1
3	2	1	35	10	3	3	3	1
4	2	1	36	8	3	4	4	1
5	3	1	37	11	3	5	5	1
6	3	1	38	12	3	6	6	1
7	4	1	39	11	3	7	3	1
8	5	1	40	9	4	8	4	2
9	6	1	41	9	4	9	5	2
10	7	1	42	10	4	10	6	2
11	7	1	43	10	4	11	3	2
12	7	1	44	8	4	12	4	2
13	8	1	45	13	4	13	7	2
14	4	1	46	13	4	14	6	2
15	5	1	47	9	4	15	8	2
16	6	2	48	9	5	16	9	3
17	9	2	49	10	5	17	7	3
18	10	2	50	10	5	18	6	3
19	10	2	51	8	5	19	8	3
20	8	2	52	13	6	20	9	4
21	11	2	53	13	6	21	10	4
22	12	2	54	9	6	22	11	4
23	11	2	55	9	6	23	12	4

24	9	2	56	10	7	24	12	5
25	9	2	57	10	7	25	10	5
26	10	2	58	14	7	26	11	6
27	10	2	59	14	7	27	13	6
28	8	2	60	14	8	28	13	7
29	11	2	61	14	8	29	14	7
30	12	2	62	14	8	30	14	8
31	11	2	63			31		

12.5.2.3 Initialisation of context models

The initialization procedure for the context models is specified in subclause 9.2.3. In this subclause, the initialization parameters for the additional context models in subclause 12.4.2are specified.

Table 12-17 – Initialisation parameters for context identifier *ctx_mb_type_I_ABT*

Context label	ctx_mb_type_I_ABT	
	m	n
0	-8	53
1	2	50
2	17	20
3	2	50
4	2	50

Table 12-18 – Initialisation parameters for context identifiers *ctx_cbp4, ctx_sig, ctx_last, ctx_abs_level* for context category 5 – 7

Context label	Context category 5				Context label	Context category 6				Context label	Context category 7			
	I-slices		P,B-slices			I-slices		P,B-			I-slices		P,B-slices	
	m	n	m	n		m	n	m	n		m	n	m	n
ctx_cbp4														
					267	-4	63	-2	61	271	-1	63	-3	63
					268	-1	70	-7	75	272	-7	74	-15	75
					269	-7	68	-12	70	273	-1	70	-13	80
					270	-5	76	-20	86	274	-5	76	-21	88
ctx_sig														
275	-1	59	-4	44	290	-8	69	-3	48	305	-8	68	-3	49
276	-7	55	-3	34	291	-12	67	-5	47	306	-11	68	-6	47
277	-6	58	-2	35	292	-11	68	-3	47	307	-11	64	-3	48
278	-7	53	-4	33	293	-12	63	-7	47	308	-11	56	-7	46
279	-9	52	-5	31	294	-12	66	-7	48	309	-13	63	-6	47
280	-5	48	-2	31	295	-12	66	-3	46	310	-11	67	-3	45

281	-14	59	-7	36	296	-7	60	-7	48	311	-8	63	-8	49
282	-8	53	-1	31	297	-12	63	-4	45	312	-12	65	-6	46
283	-10	54	-4	33	298	-11	64	-2	45	313	-11	63	-3	46
284	-5	47	4	29	299	-14	64	-5	46	314	-13	60	-2	45
285	-4	43	0	29	300	-12	57	-8	43	315	-11	53	-7	45
286	-13	56	2	31	301	-16	57	0	35	316	-13	52	-2	37
287	-9	49	0	31	302	-9	53	5	40	317	-8	50	0	38
288	-8	47	5	23	303	7	54	-3	57	318	8	52	-3	54
289	3	44	15	28	304	2	67	5	63	319	1	67	7	64
ctx_last														
320	12	29	17	27	329	9	25	24	10	338	8	25	25	9
321	5	29	23	17	330	7	25	25	7	339	8	24	25	7
322	9	28	24	21	331	13	22	27	12	340	15	21	26	13
323	18	22	22	28	332	18	19	24	19	341	21	17	25	19
324	19	23	23	31	333	20	22	24	24	342	25	22	21	29
325	23	23	23	36	334	25	23	25	32	343	28	23	25	33
326	26	22	17	43	335	21	30	22	36	344	22	31	13	44
327	14	41	17	49	336	22	38	22	45	345	15	46	13	50
328	40	31	2	58	337	13	55	-3	61	346	10	59	2	57
ctx_abs_level														
347	-9	55	-3	43	357	-11	63	-6	51	367	-11	63	-6	51
348	-1	30	-1	14	358	-3	34	-4	21	368	-3	34	-4	21
349	-2	34	0	16	359	-5	39	-6	28	369	-5	39	-6	28
350	-2	36	2	18	360	-5	41	-4	31	370	-5	41	-4	31
351	-1	37	1	23	361	-4	44	-5	37	371	-4	44	-5	37
352	-4	40	-7	36	362	-7	46	-10	41	372	-7	46	-10	41
353	-1	45	-2	43	363	-7	54	-9	49	373	-7	54	-9	49
354	-6	53	-4	47	364	-5	56	-7	51	374	-5	56	-7	51
355	-8	55	-5	49	365	-6	58	-8	53	375	-6	58	-8	53
356	-11	64	1	52	366	-11	69	-11	60	376	-11	69	-11	60

Annex A
Profile and level definitions

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

A.1 Introduction

[Ed. Notes: BEGIN EDITORIAL NOTES

Editorial notes on Annex A: (from Dave Lindbergh, later modified):

1. Depending on final terminology, it may be necessary to change “picture” to “frame”; check terms. “Max picture size” is meant to refer to a complete picture (both fields) if interlaced. The same goes for other uses of the word “picture” in this annex.
2. Yellow highlighted text in this subclause requires further consideration regarding proper terminology.
3. **PROBLEM: The meeting agreed that the HRD/VBV buffer size should be 2.56 seconds at the maximum bitrate for Levels 1, 1.1, and 2, and 1.0 seconds for all other Levels. The consequence of this would be that the HRD/VBV size for Level 1.2 would be GREATER THAN that for Level 2. I have taken the liberty to fix this by changing Max Video Bitrate for Level 1.2 from 1,000,000 bps to 768,000 bps. This solution seems better than changing the buffer size to be equal to the Level 2 amount (which would result in < 1 second of buffer). THIS MUST BE REVIEWED BY THE RAPPORTUERS and eventually the group.** Personally, I would prefer an even lower Level 1.2 bitrate limit than 768 kbps (probably 256, 384 or 512 kbps), but the change I’ve made here is the minimum possible to prevent perverse HRD requirements.

CHANGE LIST:

- Fixed level 4 max picture size to 8192 macroblock, in both Table A-1and subclause A.7.
- Changed wording to make clear 172 Hz frame rate limit is normative.
- Removed requirement that all decoders support Baseline.
- Updated Baseline Profile per JVT-D154 agreements, improved clarity of definition with “shall”.
- Added “X” profile, including language requiring support of Baseline profile
- Updated Main profile per JVT-D154, removed list of “not included” features
- Inserted new Table 1??? (from JVT-D154) and changed text to match it
- Wrote new description of “reference frame memory” limit; this replaces old “number of reference frames” subclause

END EDITORIAL NOTES]

Deleted: MB

Profiles and Levels specify the capabilities needed to decode the coded data, and may 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 set of algorithmic features and limits which shall be supported by all decoders conforming to that Profile. Note that encoders are not required to make use of any particular set of features supported in a Profile.

Deleted: defines

Each Level specifies a set of limits on the values which may be taken by the parameters 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 and memory capability, in units based on video decoding, rather than on specific implementation platforms.

Deleted: defines

A.2 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 constraints Profiles and Levels specified in this Annex. For each such Profile, the Level supported for that Profile shall also be expressed. Such expression may be in the form of coded values equivalent to a specific Profile and specific Level from this Annex.

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 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.3 Baseline profile

A.3.1 Features

All decoders supporting the Baseline Profile shall be capable of decoding bitstreams which use the following features:

a) I and P picture types
b) In-loop deblocking filter
c) Frame pictures with mb_level_aff = 0
d) 1/4-sample motion compensation
e) Tree-structured motion segmentation down to 4x4 block size
f) VLC-based entropy coding
g) Arbitrary slice order (ASO): In Baseline profile, the decoding order of slices within a picture may not follow the constraint that first_mb_in_slice is monotonically increasing within the NAL unit stream for a picture.
h) Flexible macroblock ordering (FMO, maximum 8 slice groups): In Baseline profile, num_slice_groups_minus1<8.
i) Redundant slices
j) 4:2:0 Chrominance format

Decoders supporting the Baseline Profile Level 2.1 and above shall also be capable of decoding bitstreams using:

k) Field pictures.

Conformance to the Baseline Profile is indicated by setting the syntax element profile_idc equal to 66.

A.3.2 Limits

All decoders supporting this Profile shall be capable of decoding bitstreams which:

a) use 15 or fewer Reference Frames,
b) have a compression ratio per picture of 4:1 or greater,
c) use 64 or fewer Picture Parameter Sets, and,
d) use 16 or fewer Independent Sequence Parameter Sets.

A.4 X profile

A.4.1 Features

All decoders supporting the X Profile shall be capable of decoding bitstreams which use the following features:

a) Bi-predictive slices
b) SP and SI slices
c) Data partitioned slices
d) Weighted prediction
e) All features included in the Baseline Profile

All video decoders supporting the X Profile shall also support the Baseline Profile. The Level number supported for the Baseline Profile shall not be less than the Level number supported for the X Profile.

Conformance to the X Profile is indicated by setting the syntax element profile_idc equal to 88.

A.4.2 Limits

All decoders supporting this Profile shall be capable of decoding bitstreams which:

a) use 15 or fewer Reference Frames,
b) have a compression ratio per picture of 4:1 or greater,
c) use 64 or fewer Picture Parameter Sets, and,
d) use 16 or fewer Sequence Parameter Sets.

A.5 Main profile

A.5.1 Features

All decoders supporting the Main Profile shall be capable of decoding bitstreams which use the following features:

a) Bi-predictive slices
b) CABAC
c) Weighted prediction
d) Adaptive block-size transforms (ABT)
e) All features included in the Baseline Profile except:
   1. Arbitrary Slice Order (ASO): In Main profile, the decoding order of slices within a picture shall follow the constraint that first_mb_in_slice shall be monotonically increasing within the NAL unit stream for a picture.
   2. Flexible Macroblock Order (FMO): In Main profile, num_slice_groups_minus1 shall be zero.
   3. Redundant Slices

Decoders supporting Level 2.1 and above shall also be capable of decoding bitstreams using:

f) Interlaced pictures, frame/field adaptive at picture level and macroblock level.

Conformance to the Main profile is indicated by setting the syntax element profile_idc equal to 77.

A.5.2 Limits

All decoders supporting this Profile shall be capable of decoding bitstreams which:
a) use 15 or fewer Reference Frames,
b) have a compression ratio per picture of 4:1 or greater,
c) use 64 or fewer Picture Parameter Sets, and,
d) use 16 or fewer Sequence Parameter Sets.

A.6 Level definitions

A.6.1 General

Level limits are expressed in units of whole luma macroblocks. If a particular picture height or width is not an exact multiple of a whole macroblock, that dimension shall be considered as rounded up to the next whole macroblock for the purposes of conformance with this subclause.

The definition of support for a given Level is that any picture size/frame rate combination shall be decoded where the:

a) sample processing rate (in whole macroblocks/second) is <= the Level limit given, and,

b) picture size (Height * Width, in whole macroblocks) is <= the Level limit given, and,

c) required reference memory is <= the Level limit given, and,

d) maximum video bit rate of the bitstream is <= the Level limit given, and,

e) required HRD/VBV buffer size is <= the Level limit given, and,

f) horizontal and vertical motion vector range does not exceed the Level limits given, and,

g) picture Height and picture Width (in whole macroblocks) are <= sqrt(LevelLimitMaxPictureSize * 8), and,

h) frame rate is not greater than 172 Hz.

The definition of each Level includes the requirements of all lower numbered Levels in Table A-1. Decoders supporting a given Level shall also be capable of decoding bitstreams using all lower numbered Levels.

Note that display of decoded video is outside the scope of this Recommendation | International Standard; some decoder implementations may not include displays at all, and display limitations do not necessarily cause interoperability failures.

"Picture size" means the total number of macroblocks in the complete picture (both even and odd fields if interlaced).

A.6.2 Level limits

Table A-1below gives the parameter limits for each Level. Conformance to a particular Level shall be indicated 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 #	Max Sample Processing Rate (MB/s)	Max Picture Size (MBs)	Reference Memory (1024 bytes)	Max Video Bitrate (1000 bits/sec)	Max HRD/VBV Buffer Size (bits)	Horizontal MV Range (full pels)	Vertical MV Range (full pels)	Minimum luma Bi-predictive block size
1	1 485	99	148.5	64	163 840	[-2048, 2047.75]	[-64,+63.75]	8x8
1.1	2 970	396	891.0	128	327 680	[-2048, 2047.75]	[-128,+127.75]	8x8
1.2	5 940	396	891.0	768	1 966 080	[-2048, 2047.75]	[-128,+127.75]	8x8
2	11 880	396	891.0	2 000	2 000 000	[-2048, 2047.75]	[-128,+127.75]	8x8
2.1	19 800	792	1 782.0	4 000	4 000 000	[-2048, 2047.75]	[-256,+255.75]	8x8
2.2	20 250	1 620	3 037.5	4 000	4 000 000	[-2048, 2047.75]	[-256,+255.75]	8x8
3	40 500	1 620	3 037.5	8 000	8 000 000	[-2048, 2047.75]	[-256,+255.75]	8x8
3.1	108 000	3 600	6 750.0	20 000	20 000 000	[-2048, 2047.75]	[-512,+511.75]	8x8
3.2	216 000	5 120	7 680.0	20 000	20 000 000	[-2048, 2047.75]	[-512,+511.75]	8x8
4	245 760	8 192	12 288.0	20 000	20 000 000	[-2048, 2047.75]	[-512,+511.75]	8x8
5	491 520	19 200	28 800.0	TBD	(1s @ max bps)	[-2048, 2047.75]	TBD	8x8

Levels with non-integer Level numbers in Table A-1are refered to as "intermediate Levels". All Levels have the same status, but note that some applications may choose to use only the integer-numbered Levels.

Informative subclause A.7 shows the effect of these limits on frame rates for several example picture formats.

A.6.3 Reference memory constraints on modes

"Reference Memory" means the decoder memory pool which is used to store reference frames and post-decoder frame buffers. Note that P pictures require one reference frame, and B pictures require two reference frames; any remaining reference memory may be used either for multiple reference frames or for post-decoder frame buffers.

Decoders shall support all bitstreams within a given Profile and Level where the required reference memory does not exceed the limit given for the Level.

The reference memory required for a given mode shall be calculated as:

bytes = PictureSize * NumberOfReferenceFrames * ChromaFormatParameter * 256

The PictureSize parameter is in units of whole macroblocks.

The parameter ChromaFormatParameter shall take values according to Table A-2:

Table A-2 – ChromaFormatParameter values

Chrominance Format	ChromaFormatParameter
monochrome	1
4:2:0	1.5
4:2:2	2
4:4:4	3

A.7 Effect of level limits on frame rate (informative)

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

Level number:					**1**	**1.1**	**1.2**	**2**	**2.1**	**2.2**	**3**	**3.1**	**3.2**	**4**	**5**
Max picture size (macroblocks):					**99**	**396**	**396**	**396**	**792**	**1,620**	**1,620**	**3,600**	**5,120**	**8,192**	**19,200**
Max macroblocks/second:					**1,485**	**2,970**	**5,940**	**11,880**	**19,800**	**20,250**	**40,500**	**108,000**	**216,000**	**245,760**	**491,520**
Max picture size (samples):					25,344	101,376	101,376	101,376	202,752	414,720	414,720	921,600	1,310,720	2,097,152	4,915,200
Max samples/second (1000s):					380	760	1,521	3,041	5,069	5,184	10,368	27,648	55,296	62,915	125,829
Format	**Sample Width**	**Sample Height**	**MB Wide**	**MB High**											
SQCIF	**128**	**96**	8	6	30.9	61.9	123.8	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	11	9	15.0	30.0	60.0	120.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	**320**	**240**	20	15	-	9.9	19.8	39.6	66.0	67.5	135.0	172.0	172.0	172.0	172.0
SIF	**352**	**240**	22	15	-	9.0	18.0	36.0	60.0	61.4	122.7	172.0	172.0	172.0	172.0
CIF	**352**	**288**	22	18	-	7.5	15.0	30.0	50.0	51.1	102.3	172.0	172.0	172.0	172.0
2SIF	**352**	**480**	22	30	-	-	-	-	30.0	30.7	61.4	163.6	172.0	172.0	172.0
HHR	**352**	**576**	22	36	-	-	-	-	25.0	25.6	51.1	136.4	172.0	172.0	172.0
VGA	**640**	**480**	40	30	-	-	-	-	-	16.9	33.8	90.0	172.0	172.0	172.0
4SIF	**704**	**480**	44	30	-	-	-	-	-	15.3	30.7	81.8	163.6	172.0	172.0
NTSC SD	**720**	**480**	45	30	-	-	-	-	-	15.0	30.0	80.0	160.0	172.0	172.0
4CIF	**704**	**576**	44	36	-	-	-	-	-	12.8	25.6	68.2	136.4	155.2	172.0
PAL SD	**720**	**576**	45	36	-	-	-	-	-	12.5	25.0	66.7	133.3	151.7	172.0
SVGA	**800**	**600**	50	38	-	-	-	-	-	-	-	56.8	113.7	129.3	172.0
XGA	**1024**	**768**	64	48	-	-	-	-	-	-	-	35.2	70.3	80.0	160.0
720p	**1280**	**720**	80	45	-	-	-	-	-	-	-	30.0	60.0	68.3	136.5
4VGA	**1280**	**960**	80	60	-	-	-	-	-	-	-	-	45.0	51.2	102.4
SXGA	**1280**	**1024**	80	64	-	-	-	-	-	-	-	-	42.2	48.0	96.0
16SIF	**1408**	**960**	88	60	-	-	-	-	-	-	-	-	-	46.5	93.1
16CIF	**1408**	**1152**	88	72	-	-	-	-	-	-	-	-	-	38.8	77.6
4SVGA	**1600**	**1200**	100	75	-	-	-	-	-	-	-	-	-	32.8	65.5
1080i	**1920**	**1080**	120	68	-	-	-	-	-	-	-	-	-	30.1	60.2
2Kx1K	**2048**	**1024**	128	64	-	-	-	-	-	-	-	-	-	30.0	60.0
4XGA	**2048**	**1536**	128	96	-	-	-	-	-	-	-	-	-	-	40.0
16VGA	**2560**	**1920**	160	120	-	-	-	-	-	-	-	-	-	-	25.6

Note 1 This is a variable-picture-size specification. The specific picture sizes in this Table are illustrative examples only.

Note 2 XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, HHR aka 2CIF aka 1/2 D1, aka 1/2 ITU-R BT.601.

Note 3 Frame rates given are correct for progressive scan modes, and for interlaced if "MB High" column value is even.

Annex B
Byte stream format

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

B.2 Introduction

This annex specifies 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.

Deleted: defines

The byte stream format consists of a sequence of byte_stream_unit( ) structures. Each byte_stream_unit( ) contains one start code prefix (SCP) and one nal_unit( ). Optionally, at the discretion of the encoder, the byte_stream_unit( ) may also contain additional "stuffing" zero-valued bytes as specified in this clause.

There are two types of start code prefixes:

– A short SCP, consisting of one byte having the value zero (0x00) followed by one byte having the value one (0x01), and

– A long SCP, consisting of two bytes having the value zero (0x00) followed by one byte having the value one (0x01).

The long SCP provides a mechanism for decoder byte-alignment recovery in the event of loss of decoder synchronization.

B.3 Byte stream NAL unit syntax

byte_stream_unit() {	**Category**	**Mnemonic**
while (next_bits(16) ! = 0x0001 && next_bits(24) ! = 0x000001)		
zero_byte		f(8) = 0x00
if(next_bits() = = 0x000001)		
zero_byte		f(8) = 0x00
zero_byte		f(8) = 0x00
one_byte		f(8) = 0x01
nal_unit( )		
}		

B.4 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units.

zero_byte is a single byte (8 bits) having the value zero (0x00). Optionally, at the discretion of the encoder, the beginning of a byte_stream_unit() may contain more zero_byte syntax elements than required in this subclause.

The minimum required number of zero_byte syntax elements depends on the nal_unit_type as specified in Table 7-1, in order to ensure the use of the long SCP for certain nal_unit_type values. This ensures use of the long SCP in the byte_stream_unit() for these nal_unit() structures. The use of the long SCP for nal_unit() structures with other values of nal_unit_type is optional. At least two zero_byte syntax elements shall be present in each byte_stream_unit() for the following values of nal_unit_type:

– 0x06: Supplemental enhancement information,

– 0x07: Sequence parameter set,

– 0x08: Picture parameter set, and

– 0x09: Picture delimiter.

one_byte is a single byte (8 bits) having the value one (0x01). A sequence of two zero_byte syntax elements followed by a one_byte is a long SCP, and one zero_byte followed by a one_byte is a short SCP.

The use of the byte sequence 0x00 0x02 is reserved for future use by ITU-T | ISO/IEC. [Ed. Note: Move this to a better place.]

B.5 Decoder byte-alignment recovery (informative)

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

If the decoder does not have byte alignment with the encoder's byte stream, the decoder can examine the incoming bit stream for the binary pattern '00000000 00000000 00000001' (23 consecutive zero-valued bits followed by a non-zero bit). The bit immediately following this pattern is the first bit of a whole 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 byte sequences 0x00 0x01 and 0x00 0x03.

If the byte sequence 0x00 0x01 is detected, this represents a SCP. If the previous byte was 0x00, the SCP is a long SCP. Otherwise, it is a short SCP.

If the byte sequence 0x00 0x03 is detected, the decoder discards the byte 0x03 as shown in the rbsp_extraction() syntax diagram.

NOTE - Many systems are inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this subclause.

NOTE - The byte alignment detection procedure described in this subclause is equivalent to searching a byte sequence for 0x00 0x00, starting at any alignment position. Detecting this pattern indicates that the next non-zero byte contains the end of a SCP, 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)

C.1 Hypothetical reference decoder and buffering verifiers

The hypothetical reference decoder (HRD) represents a set of normative requirements on coded bitstreams or packet streams. These constraints must be enforced by an encoder, and can be assumed by a decoder or multiplexor to be true. It is possible to verify the conformance of a bitstream or packet stream to the requirements of this subclause by examining the bitstream or packet stream only.

This subclause specifies the normative requirements of the HRD. Subclause C.2provides additional information that is important for a full understanding of the HRD operation.

Deleted: defines

Two types of streams may be subject to the HRD requirements of this Recommendation | International Standard; a stream of VCL NAL Units and a bitstream. Figure C-1shows how each of these is constructed from the RBSP. In other words, a given set of HRD parameters may pertain to the VCL data only or to the multiplexed combination of VCL and NAL. This is signalled through video usability information (subclauses E.2 and E.3).

.

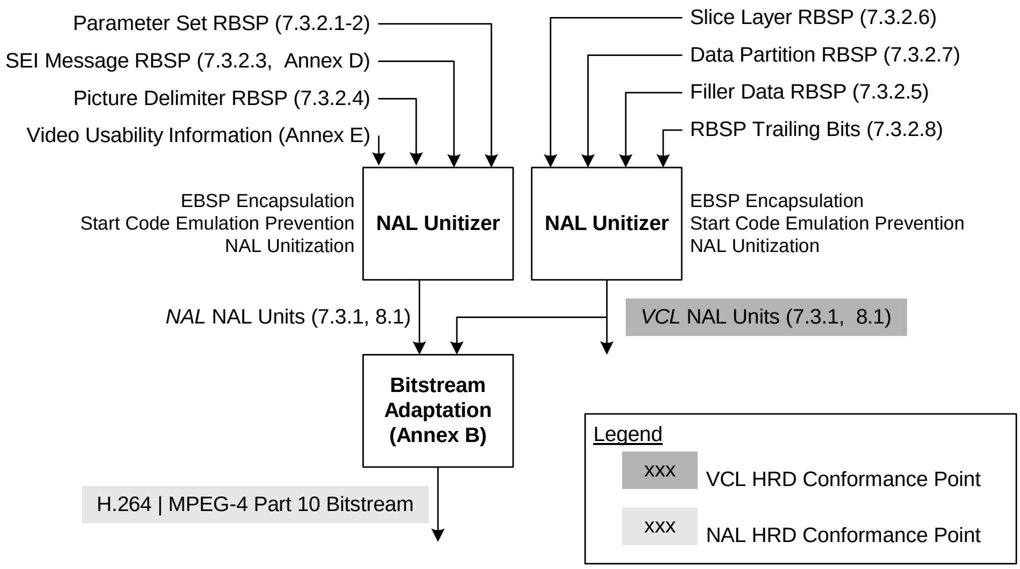


Figure C-1 – Structure of Byte streams and NAL unit streams and HRD Conformance Points

The HRD can contain any combination of the following buffering verifiers, as shown in Figure C-2:

- One or more pre-decoder buffers, each of which is either variable bit rate (VBR) or constant bit rate (CBR)
- At most one reference and post-decoder buffer attached to the output of one of the pre-decoder buffers

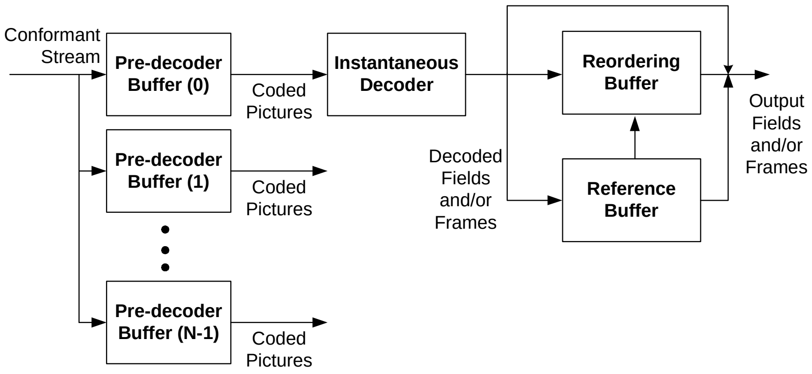


Figure C-2 – HRD Buffer Verifiers

The multiple buffering verifiers exist because a bit stream or packet stream may conform to multiple pre-decoder buffers, as detailed in subclause C.2.2.

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 pre-decoder buffer just prior to or after removal of a transmitted picture is not necessarily an integer. Furthermore, 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, each of these may vary from the signalled or specified value.

This hypothetical reference decoder uses two time bases. One time base is a 90 kHz clock, and is only in operation for a short time after the reception of a Buffering Period SEI message. The second time base uses the num_units_in_tick and time_scale syntax in the Sequence Parameter Set to derive the time interval between picture removals from the buffers (and in some cases between picture arrivals to the pre-decoder buffer).

In the following description, let t_c = num_units_in_tick ÷ time_scale be the *clock tick* associated with the second clock. The clock tick is a time interval no larger than the shortest possible inter-picture capture interval in seconds. 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 buffer arrival bit rate.

The following statements are normative requirements on the composition of a conforming bitstream. If multiple sequence parameter sets pertain to the bit stream or packet stream, they must contain consistent HRD information. In the case that any HRD buffers are signalled in the sequence parameter set(s), then the following rules dictate the insertion of SEI messages in the bit stream or packet stream.

1. At each decoder refresh point (IDR, ODR or GDR), a buffering period SEI message shall follow the last NAL Unit of the last picture before a decoder refresh and precede the first NAL Unit of the first picture after the decoder refresh. Note that in the case of an IDR, this SEI message will precede the indication of the decoder refresh point.
2. An HRD picture SEI message must follow the last NAL Unit of each picture and precede the first NAL Unit of the next picture. Each of these SEI messages pertains to the picture that follows it.

C.1.1 Operation of VCL video buffering verifier (VBV) pre-decoder buffer

This specification applies independently to each pre-decoder buffer VUI sequence parameters within the sequence parameter set.

C.1.1.1 Timing of bitstream or packet stream arrival

The buffer is initially empty. The first bit of the first transmitted picture begins to enter the buffer at *initial arrival time* $t_{ai}(0)=0$ at the bit rate bit_rate[k] associated with the pre-decoder buffer (see subclause 8.3.3). The last bit of the first transmitted picture finishes arriving at *final arrival time*

$$t_{af}(0) = b(0) \div \text{bit\_rate}[k], \qquad \text{(C-1)}$$

where $b(n)$ is the size in bits of the n-th transmitted picture. 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 pre-decoder buffer:

$$t_{af}(n) = t_{ai}(n) + b(n) \div \text{bit\_rate}[k]. \qquad \text{(C-2)}$$

For each subsequent picture, the initial arrival time of picture n is the later of $t_{af}(n\text{-}1)$ and the sum of all preceding **pre_dec_removal_delay** times, as indicated in Equation C-3.

$$t_{ai}(n) = \max\{\ t_{af}(n\text{-}1),\ t_c \times \sum_{m=0}^{n-1} \text{pre\_dec\_removal\_delay}(m)\ \} \qquad \text{(C-3)}$$

See subclauses D.2.5and D.3.5 for the syntax and semantics of the pre_dec_removal_delay times. When the encoder is producing a bit rate lower than the bit rate associated with a pre-decoder buffer, this rule may delay the entry of some pictures into the pre-decoder buffer, producing periods during which no data enters.

Deleted: C.3.5???

C.1.1.2 Timing of coded picture removal

For the first picture and all pictures that are the first complete picture after receiving a buffering period SEI message, the coded data associated with the picture is removed from the pre-decoder buffer at a *removal time* equal to the following:

$$t_r(0) = \text{initial\_pre\_dec\_removal\_delay} \div 90000 \qquad \text{(C-4)}$$

where initial_pre_dec_removal_delay is the pre-decoder removal delay 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 $n\text{-}1$; the delay is equal to the time indicated in the pre_dec_removal_delay syntax element present in the HRD picture SEI message.

$$t_r(n) = t_r(n\text{-}1) + t_c \times \text{pre\_dec\_removal\_delay}(n) \qquad \text{(C-5)}$$

At this time, the coded data for the next transmitted picture is removed from the pre-decoder buffer.

In the case that the amount of coded data for picture n, $b(n)$, is so large that it prevents removal at the computed removal time, the coded data is removed at the *delayed removal time*, $t_{r,ld}(n, m^*)$, given by

$$t_{r,d}(n,m^*) = t_r(0) + t_c \times m^*, \qquad \text{(C-6)}$$

where m^* is such that $t_{r,d}(n, m^*\text{-}1) < t_{af}(n) \le t_{r,ld}(n, m^*)$. This is an aspect of low-delay operation (see subclause C.2.1.2). This delayed removal time is the next time instant after the final arrival time $t_{af}(n)$ which is delayed with respect to $t_r(0)$ by an integer multiple of t_c.

C.1.1.3 Conformance constraints on coded bitstreams or packet streams

A transmitted or stored stream of coded data conforming to this Recommendation | International Standard fulfils the following requirements.

- ***Removal time consistency.*** For each picture, the removal times $t_r(n)$ computed using different buffering periods as starting points for conformance verification shall be consistent to within the accuracy of the two clocks used (90 kHz clock used for initial removal time and t_c clock used for subsequent removal time calculations). This can be ensured at the encoder by computing the pre-decoder removal delay (initial_pre_dec_removal_delay) for a buffering period SEI message from the arrival and removal times computed using Equations C-3 and C-5. Any small deviations between the values computed in the different ways shall not cause violation of any of the following constraints.

- ***Underflow and Overflow Prevention.*** The buffer must 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 pre-decoder buffer at the picture's computed removal time tr(n). In other words, its final arrival time must be no later than its removal time: taf(n) ≤ tr(n). Further, the removal time tr(n) must be no later than the time-equivalent of the buffer size te[pre_dec_buffer_size[k]]. Note that this prevents overflow.

- ***Big Picture Removal Time, Overflow Prevention and Resynchronisation of Underflow Prevention.*** If the final arrival time $t_{af}(n)$ of picture n 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.

- ***Constant Bit Rate Constraint.*** If **vbr_cbr_flag[**k**]** = = 1, data shall arrive continuously at the input to the pre-decoder buffer. This is equivalent to ensuring that $t_{af}(n\text{-}1) \geq t_c \times \sum_{m=0}^{n-1}$ pre_dec_removal_delay(m).

- ***Time Duration Constraint.*** For each picture immediately preceding a Buffering Period SEI message, the sum of pre-decoder removal delays from the start of the sequence up to that point of time shall be no further from the accumulated sequence duration as represented by the sum of prev_buf_period_duration than the removal_time_tolerance in the relevant sequence parameter set.

  If the picture immediately preceding the Buffering Period SEI message is the n-th picture in transmitted order, and the buffering period SEI message is the k-th such message, then this constraint amounts to the following:

$$\left| \sum_{m=0}^{n-1} \text{pre\_dec\_removal\_delay(m)} - \sum_{m=0}^{k-1} \text{prev\_buf\_period\_duration(m)} \right| \leq \text{removal\_time\_tolerance} \qquad \text{(C-7)}$$

- ***Maximum Decoder Frame Rate.*** The interval between consecutive removal times shall not be lower than the minimum picture interval, specified as the inverse of the maximum picture rate (See A.5.1).

C.1.2 Operation of the post-decoder buffer verifier

[Ed.Note: This subclause must be revised. Subject to review by the ad hoc team.]

C.1.2.1 Arrival timing

A reconstructed picture is added to the post-decoder buffer at the same time when the corresponding coded picture is removed from the pre-decoder buffer.

C.1.2.2 Removal timing

Data is not removed from the post-decoder buffer during a period called the initial post-decoder buffering period. The period starts when the first picture is added to the post-decoder buffer.

When the initial post-decoder buffering period has expired, the playback timer is started from the earliest display time of the pictures residing in the post-decoder buffer at that time.

A picture is virtually displayed when the playback timer reaches the scheduled presentation time of the picture.

A picture memory is marked unused in the post-decoder buffer when it is virtually displayed and when it is no longer needed as a reference picture.

C.1.2.3 Conformance constraints

The occupancy of the post-decoder buffer shall not exceed the default or signalled buffer size.

Each picture shall be available in the post-decoder buffer before or on its presentation time.

C.2 Informative description of the HRD

Subclause C.1 contains the normative requirements imposed by a set of buffering verifiers. This subclause provides explanatory text describing in more detail the operation and capabilities of these buffers.

An HRD represents a means to communicate how the bit rate is controlled in the process of compression. The HRD contains a pre-decoder buffer (or VBV Buffer) through which compressed data flows with a precisely specified arrival and removal timing, as shown in Figure C-3. An HRD may be designed for variable or constant bit rate operation, and for low-delay or delay-tolerant behavior. The HRD described in this document handles all cases.

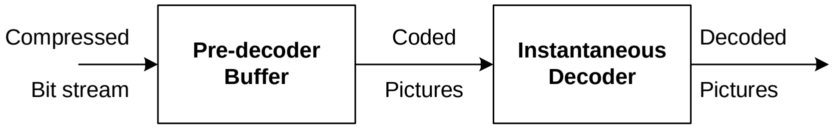


Figure C-3 – A Hypothetical Reference Decoder

Compressed data representing a sequence of *coded pictures* flows into the pre-decoder buffer according to a specified *arrival schedule*. All compressed bits associated with a given coded picture are removed from the pre-decoder buffer by the instantaneous decoder at the specified *removal time* of the picture.

The pre-decoder buffer *overflows* if the buffer becomes full and more bits are arriving. The buffer *underflows* if the removal time for a picture occurs before all compressed bits representing the picture have arrived.

C.2.1 Constrained arrival time leaky bucket (CAT-LB) model

The hypothetical reference decoder (HRD) is a mathematical model of a decoder and its input buffer. The *k*-th pre-decoder buffer of the HRD is characterized by the pre-decoder peak rate bit_rate[*k*] (in bits per second), the buffer size pre_dec_buffer_size[*k*] (in bits), the sequence initial pre-decoder buffer removal delay (in seconds), as well as picture removal delays for each picture. The first three of these parameters represent levels of resources (transmission capacity, buffer capacity, and delay) used to decode a bitstream.

The term "leaky bucket" arises from the analogy of the encoder as a system that "dumps" water in discrete chunks into a bucket that has a hole in it. The departure of bits from the encoder buffer corresponds to water leaking out of the bucket. Here, the decoder buffer is described, which has an inverse behaviour where bits flow *in* at a constant rate, and are removed in chunks.

The leaky bucket described here is called a *constrained arrival time* leaky bucket because the arrival times of all pictures after the first are constrained to arrive at the buffer input no earlier than the difference in hypothetical encoder processing times between that picture and the first picture. In other words, if a picture is encoded exactly seven seconds after the first picture was encoded, then its bits are guaranteed not to start arriving in the buffer prior to seven seconds after the bits of the first picture started arriving. It is possible to know this encoding time difference because it is sent in the bitstream as the picture removal delay.

C.2.1.1 Operation of the CAT-LB HRD

The HRD input buffer has capacity pre_dec_buffer_size[*k*] bits. Initially, the buffer begins empty. The lifetime in the buffer of the coded bits associated with picture *n* is characterized by the *arrival interval* $\{t_{ai}(n), t_{af}(n)\}$ and the *removal time* $t_r(n)$. The end-points of the arrival interval are known as the *initial arrival time* and the *final arrival time*.

At time $t_{ai}(0) = 0$, the buffer begins to receive bits at the rate bit_rate[*k*]. The removal time $t_r(0)$ for the first picture is computed from the pre-decoder removal delay initial_pre_dec_removal_delay (see Buffering Period SEI Message) associated with the buffer by the following:

$$t_r(0) = 90{,}000 \times \text{initial\_pre\_dec\_removal\_delay}. \quad \text{(C-8)}$$

Removal times $t_r(1)$, $t_r(2)$, $t_r(3)$, …, for subsequent pictures (in transmitted order) are computed with respect to $t_r(0)$, as follows. Let the *clock tick* t_c be specified by

$$t_c = \text{num\_units\_in\_tick} \div \text{time\_scale} \quad \text{(C-9)}$$

For instance, if time_scale = 60,000 and num_units_in_tick = 1,001, then

$$t_c = 1{,}001 \div 60{,}000 = 16.68333\ldots \text{ milliseconds.} \quad \text{(C-10)}$$

In the HRD picture SEI message for each picture, there is a pre_dec_removal_delay syntax element. This indicates the number of clock ticks to delay the removal of picture *n* after removing picture *n-1*. Thus, the removal time is simply

$$t_r(n) = t_r(n-1) + t_c \times \text{pre\_dec\_removal\_delay}(n) \quad \text{(C-11)}$$

Note that this recursion can be used to show that

$$t_r(n) = t_r(0) + t_c \times \sum_{m=1}^{n} [\text{pre\_dec\_removal\_delay}(m)], \quad \text{(C-12)}$$

The calculation of arrival times is more complex, because of the arrival time constraint. The initial arrival time of picture *n* is equal to the final arrival time of picture *n-1*, unless that time precedes the earliest arrival time, computed by

$$t_{ai,earliest}(n) = t_c \times \sum_{m=1}^{n} [\text{pre\_dec\_removal\_delay}(m)] \quad \text{(C-13)}$$

Let $b(n)$ be the number of bits associated with picture *n*. The duration of the picture arrival interval is always the time-equivalent of the picture size in bits, at the rate bit_rate[k].

$$t_{af}(n) - t_{ai}(n) \equiv te[b(n)] = b(n) \div \text{bit\_rate[k]} \quad \text{(C-14)}$$

Figure C-4 demonstrates a segment of the pre-decoder buffer fullness plot for a CAT-LB with the parameters given in Table C-1and picture sizes given by the first column of Table C-2. Note that Table C-2lists for each picture the values for many times of interest in the buffering process. In addition to quantities specified above, the second column of Table C-2contains t_e, which represents a hypothetical encoding time equal to the earliest possible initial arrival time of the picture.

Table C-1 - Attributes of an example CAT-LB HRD

Attribute	Value	Units
time_scale	1	units per second
num_units_in_tick	1	units per tick
bit_rate	1000	bits per second
pre_dec_buffer_size	10	bits
initial_delay	10	seconds

Table C-2 - Picture sizes, and encoding, arrival and removal times for the example CAT-LB HRD

b	t_e	t_{ai}	t_{af}	t_{ai}-t_e	t_r	t_r - t_{ai}	t_r - t_e
5,000	0	0	5	0	10	10	10
1,000	1	5	6	4	11	6	10
1,000	2	6	7	4	12	6	10
1,000	3	7	8	4	13	6	10
1,000	4	8	9	4	14	6	10
1,000	5	9	10	4	15	6	10
500	6	10	10.5	4	16	6	10
500	7	10.5	11	3.5	17	6.5	10
500	8	11	11.5	3	18	7	10
500	9	11.5	12	2.5	19	7.5	10
500	10	12	12.5	2	20	8	10
500	11	12.5	13	1.5	21	8.5	10
500	12	13	13.5	1	22	9	10
500	13	13.5	14	0.5	23	9.5	10
500	14	14	14.5	0	24	10	10
500	15	15	15.5	0	25	10	10
500	16	16	16.5	0	26	10	10
500	17	17	17.5	0	27	10	10
3,000	18	18	21	0	28	10	10

3,000	19	21	24	2	29	8	10
3,000	20	24	27	4	30	6	10
3,000	21	27	30	6	31	4	10
2,000	22	30	32	8	32	2	10
300	23	32	32.3	9	33	1	10
300	24	32.3	32.6	8.3	34	1.7	10
300	25	32.6	32.9	7.6	35	2.4	10
300	26	32.9	33.2	6.9	36	3.1	10
300	27	33.2	33.5	6.2	37	3.8	10
300	28	33.5	33.8	5.5	38	4.5	10
300	29	33.8	34.1	4.8	39	5.2	10
300	30	34.1	34.4	4.1	40	5.9	10
300	31	34.4	34.7	3.4	41	6.6	10
300	32	34.7	35	2.7	42	7.3	10
300	33	35	35.3	2	43	8	10
300	34	35.3	35.6	1.3	44	8.7	10
300	35	35.6	35.9	0.6	45	9.4	10
300	36	36	36.3	0	46	10	10
300	37	37	37.3	0	47	10	10
300	38	38	38.3	0	48	10	10
300	39	39	39.3	0	49	10	10
300	40	40	40.3	0	50	10	10
300	41	41	41.3	0	51	10	10
300	42	42	42.3	0	52	10	10
500	43	43	43.5	0	53	10	10
500	44	44	44.5	0	54	10	10
500	45	45	45.5	0	55	10	10
500	46	46	46.5	0	56	10	10
500	47	47	47.5	0	57	10	10
500	48	48	48.5	0	58	10	10
500	49	49	49.5	0	59	10	10
500	50	50	50.5	0	60	10	10
500	51	51	51.5	0	61	10	10
500	52	52	52.5	0	62	10	10

Legend:

t_e Encoding time

t_{ai} Initial arrival time

t_{af} Final arrival time

t_r Removal time

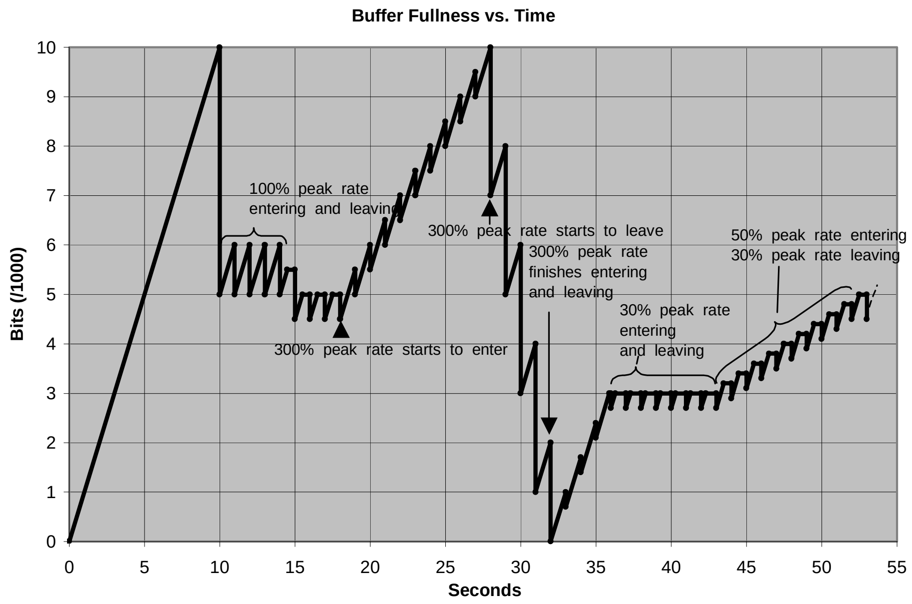


Figure C-4 – Buffer fullness plot for example HRD in Table C-2 with picture sizes given in Table C-3

As can be seen from Table C-2, the initial picture is large, and is followed by five pictures at exactly the buffer arrival rate *R*. This is followed by twelve pictures at half the rate, four pictures at three times the rate and one picture at twice the rate. Following this are two segments with pictures at 30% and 50% of the rate, respectively. In Figure C-4, the time interval from 10 seconds to 18 seconds illustrates the behaviour when the bit rate is constant and at or below the rate *R*. In fact, whenever the arrival bit rate remains less than *R* for a time, the lower points of the fullness curve will not change. Further, the fullness at the peak in such a segment will be proportional to the fraction of the peak rate being consumed by the pictures. From seconds 18 to 28, we see the temporary effect of an increase in arrival rate to above *R*. Once those large pictures start to exit the buffer, the bit rate of pictures leaving the buffer exceeds *R*, and the fullness decreases. This process terminates at second 32, when the big pictures have exited and the series of smaller pictures starts entering the buffer. During seconds 36-43, the 30% peak rate pictures are entering and leaving the buffer, and during seconds 43-52, 30% peak rate pictures are leaving while 50% peal rate pictures are entering. Hence the buffer fullness rises. Once 50% peak rate pictures begin to leave, the fullness stabilizes at 50% full. Note that this pre-decoder buffer stabilizes at a fullness that is proportional to the ratio of the short-term average bit rate to the arrival bit rate, rather than at 100%..

In general, the curve of buffer fullness vs. time is given by the following expression:

$$BF(t) = \sum_{n} \left[I(t_{af}(n) \le t < t_r(n)) \times b(n) + I(t_{ai}(n) < t < t_{af}(n)) \times be(t - t_{ai}(n)) \right] \quad \text{(C-15)}$$

This expression uses indicator functions $I(\cdot)$ with time-related logical assertions as arguments to sum only those pictures that are completely in the buffer at time *t*, plus the appropriate portion of the picture currently entering the buffer, if one is. The indicator function $I(x)$ is '1' if *x* is true and '0' otherwise.

C.2.1.2 Low-delay operation

Low-delay operation is obtained by selecting a low value for the initial pre-decoder removal delay. This results in true low delay through the buffer because, under normal operation, no removal delay ($t_r(n)$-$t_{ai}(n)$) can exceed the initial removal delay $t_r(0)$. To see this, consider that the maximum removal delay for picture *n* occurs when the initial arrival time is equal to the earliest arrival time. Therefore, the maximum removal delay is given by $t_r(n)$ - $t_{ai,earliest}(n)$. But,

$$t_r(n) = t_r(0) + t_c \times \sum_{m=1}^{n} [\text{pre\_dec\_removal\_delay(m)}],$$

and

$$t_{\mathrm{ai,earliest}}(n) = t_c \times \sum_{m=1}^{n} \; [\text{pre\_dec\_removal\_delay(m)}],$$

so

$$t_r(n) - t_{\mathrm{ai,earliest}}(n) = t_r(0). \qquad \text{(C-16)}$$

Thus setting an initial low delay creates a steady-state low-delay condition.

However, in low-delay operation, it is useful to be able to process the occasional large picture whose size is so large than that it cannot be removed by its indicated removal time. Such a large picture can arise at a scene change, for example. This would ordinarily lead to an "underflow" condition. When a large picture is encountered, the rules for removal are relaxed to prevent this. The picture is removed at the *delayed removal time*, $t_{\mathrm{r,ld}}(n, m^*)$, given by

$$t_{\mathrm{r,ld}}(n,m^*) = t_r(0) + t_c \times m^*, \qquad \text{(C-17)}$$

where m^* is such that $t_{\mathrm{r,ld}}(n, m^*\text{-}1) < t_{\mathrm{ai}}(n) + \mathrm{te}[b(n)] \leq t_{\mathrm{r,ld}}(n, m^*)$. Note that the buffer must be large enough that this large picture can be accommodated without overflow. Immediately after such a picture is received the removal time of the next picture must be such that low-delay operation is resumed. An encoder can facilitate this by skipping a number of pictures immediately after the large picture, if necessary.

C.2.1.3 Bitstream / packet stream constraints

The buffer must not be allowed to underflow or overflow. Furthermore, all pictures except the isolated big pictures must be completely in the buffer before their computed removal times. Isolated big pictures are allowed to arrive later than their computed removal times, but must still obey the overflow constraint. In CBR mode, there must be no gaps in bit arrival.

C.2.1.3.1 Underflow

The underflow constraint, $BF(t) \geq 0$ for all t, is satisfied if the final arrival time of each picture precedes its removal time.

$$t_{\mathrm{af}}(n) \leq t_r(n) \qquad \text{(C-18)}$$

This puts an upper bound on the size of picture n. The picture size can be no larger than the bit-equivalent of the time interval from the start of arrival to the removal time.

$$b(n) \leq be[t_r(n) - t_{\mathrm{ai}}(n)] \qquad \text{(C-19)}$$

Since the initial arrival time $t_{\mathrm{ai}}(n)$ is in general a function of the sizes and removal delays of previous pictures, the constraint on $b(n)$ will vary over time as well.

C.2.1.3.2 Overflow

Overflow is avoided provided the buffer fullness curve $BF(t)$ never exceeds the buffer size B.

The constraints that the initial pre-decoder removal delay must be no larger than the time-equivalent of the buffer size, $t_r(0) \leq te(B)$, and that under normal operation no removal delay can exceed the initial one guarantee that no overflow occurs in normal operation. To avoid overflow of an isolated big picture, the picture size is constrained by

$$b(n) \leq be[B - t_{\mathrm{ai}}(n)] \qquad \text{(C-20)}$$

C.2.1.3.3 Constant bitrate (CBR) operation

The CAT-LB model operates in constant bit rate mode if one further constraint is applied - that data must constantly arrive at the input of the buffer. This ensures that the average rate is equal to the buffer rate R. This model behaves like an MPEG-1 CBR model with variable frame rate. This condition is ensured if the final arrival time of picture n is no earlier than the earliest initial arrival time of picture $n+1$.

$$t_{af}(n) \geq t_{ai,earliest}(n+1) = t_c \times \sum_{m=1}^{n} \text{[pre\_dec\_removal\_delay(m)]} \quad \text{(C-21)}$$

This time constraint puts a lower bound on $b(n)$.

C.2.1.4 Rate control considerations

An encoder employs rate control as a means to constrain the varying bit rate characteristics of the coded bitstream or packet stream in order to produce high quality coded pictures at the target bit rate(s). A rate control algorithm may target a variable bit rate (VBR) or a constant bit rate (CBR). It may even target both a high peak rate using a VBR scheme and an average rate using a CBR scheme. Further, as shown in subclause C.2.2, multiple VBR rates can be targeted.

Rate control must ensure conformance with the pre-decoder buffers. This is related to the first goal of rate control, but is not necessarily the same. In this subclause, the way the pre-decoder buffers influences rate control is discussed. In a VBR pre-decoder buffers, the buffer must not overflow or underflow, but gaps may appear in the arrival rate. In order to meet these constraints, the encoder must ensure that for all t, the following inequalities remain true:

$$0 \leq BF(t) \leq B, \text{ for all } t. \quad \text{(C-22)}$$

Using Equation D-14, this becomes:

$$0 \leq \sum_{n} [I(t_{af}(n) \leq t < t_r(n)) \times b(n) + I(t_{ai}(n) < t < t_{af}(n)) \times be(t\text{-}t_{ai}(n))] \leq B, \text{ for all } t. \quad \text{(C-23)}$$

The buffer fullness $B(t)$ is a piecewise non-decreasing function of time, with each non-decreasing interval bounded by two consecutive removal times. Therefore, it is sufficient to guarantee the conformance at the interval endpoints; *i.e.* at the removal times. In particular, if underflow is prevented at the start of an interval (just after removal of a picture), it is completely prevented. The same holds for overflow at the end of the interval, just prior to picture removal. Therefore, the points of interest are the removal times. In the buffer at $t_r^-(n)$ and contributing to Equation C.22 are picture n and possibly some additional pictures up to picture $m>n$ (with the last picture possibly only partially in the buffer). All pictures earlier than picture n have been removed. At $t_r^+(n)$, picture n has been removed.

Thus when encoding picture n, one rate control task is to allocate bits to picture n and the others in the immediate future in such a way that overflow is prevented at $t_r^-(n)$, and underflow is prevented at $t_r^+(n)$. Most immediately, as long as $b(n)$ is small enough so that $te[b(n)] \leq t_r(n) - t_{ai}(n)$, both overflow at $t_r^-(n)$ and underflow at $t_r^+(n)$ are prevented. This is usually a very high limit, and a rate control method will most likely further limit $b(n)$ through its bit allocation process.

C.2.2 Multiple leaky bucket description

C.2.2.1 Schedule of a bitstream

The sequence of removal time and picture size pairs $\{(t_r(n), b(n)), n=0,1,\ldots\}$ is called the schedule of a bitstream. The schedule of a bitstream is intrinsic to the bitstream, and completely characterizes the instantaneous coding rate of the bitstream over its lifetime. Although a bitstream may conform to VBVs with different peak bit rates and different pre-decoded buffer sizes, the schedule of the bitstream is independent of the VBV.

C.2.2.2 Containment in a leaky bucket

A leaky bucket with leak rate R_1, bucket size B_1, and initial bucket fullness $B_1–F_1$ is said to contain a bitstream with schedule $\{(t_r(n), b(n)), n=0,1,\ldots\}$ if the bucket does not overflow under the following conditions. At t_0, d_0 bits are inserted into the leaky bucket on top of the $B_1–F_1$ bits already in the bucket, and the bucket begins to drain at rate R1 bits per second. If the bucket empties, it remains empty until the next insertion. At time t_i, $i \geq 1$, d_i, bits are inserted into the bucket, and the bucket continues to drain at rate R_1 bits per second. In other words, for $i \geq 0$, the state of the bucket just prior to time t_i is

$$b_0 = B_1–F_1 \quad \text{(C-24)}$$

$$b_{i+1} = \max\{0, b_i + d_i – R_1(t_{i+1}–t_i)\}. \quad \text{(C-25)}$$

The leaky bucket does not overflow if $b_i + d_i \leq B_1$ for all $i \geq 0$.

Equivalently, the leaky bucket contains the bitstream if the graph of the schedule of the bitstream lies between two parallel lines with slope R_1, separated vertically by B_1 bits, possibly sheared horizontally, such that the upper line begins at F_1 at time t_0, as illustrated in Figure C-5. Note from Figure C-5 that the same bitstream is containable in more than one leaky bucket. Indeed, a bitstream is containable in an infinite number of leaky buckets.

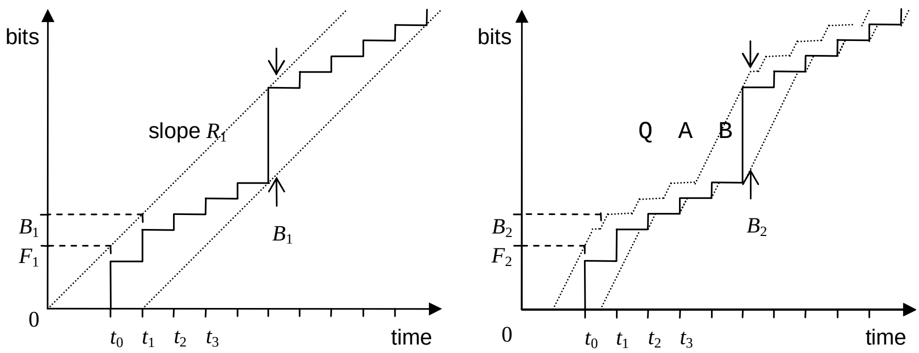


Figure C-5 – Illustration of the leaky bucket concept

If a bitstream is contained in a leaky bucket with parameters (R_1,B_1,F_1), then when it is input with peak rate R_1 to a hypothetical reference decoder with parameters R=R_1, B=B_1, and F=F_1, then the HRD buffer does not overflow or underflow.

C.2.2.3 Minimum buffer size and minimum peak rate

If a bitstream is contained in two leaky buckets with parameters (R_1,B_1,F_1) and (R_2,B_2,F_2), then it is also contained in any leaky bucket with parameters (R,B,F) where a) $R_1 \le R \le R_2$, b) $B \ge B_{min}(R)$, and c) $F \ge F_{min}(R)$ and $B_{min}(R)$ and $F_{min}(R)$ are specified by

$$B_{min}(R) = \alpha B_n + (1 - \alpha)B_{n+1}, \qquad \text{(C-26)}$$

$$F_{min}(R) = \alpha F_n + (1 - \alpha)F_{n+1}, \qquad \text{(C-27)}$$

and

$$\alpha = (R_{n+1} - R) \div (R_{n+1} - R_n). \qquad \text{(C-28)}$$

For $R \le R_1$,

$$B_{min}(R) = B_1 + (R_1 - R)T \qquad \text{(C-29)}$$

$$F_{min}(R) = F_1, \qquad \text{(C-30)}$$

where $T = t_{L-1} - t_0$ is the duration of the bitstream (i.e., the difference between the decoding times for the first and last pictures in the bitstream). And for $R \ge R_N$,

$$B_{min}(R) = B_N \qquad \text{(C-31)}$$

$$F_{min}(R) = F_N\,. \qquad \text{(C-32)}$$

Thus, the leaky bucket parameters can be linearly interpolated and extrapolated.

Alternatively, when the bitstream is communicated to a decoder with buffer size B, it is decodable provided $\ge R_{min}(B)$ and $F \ge F_{min}(B)$, where for $B_n \ge B \ge B_{n+1}$,

$$R_{min}(B) = \alpha R_n + (1 - \alpha)R_{n+1} \qquad \text{(C-33)}$$

$$F_{min}(B) = \alpha F_n + (1 - \alpha)F_{n+1} \qquad \text{(C-34)}$$

$$\alpha = (B - B_{n+1}) \div (B_n - B_{n+1}). \qquad \text{(C-35)}$$

For $B \ge B_1$,

$$R_{min}(B) = R_1 - (B - B_1) \div T \tag{C-36}$$

$$F_{min}(B) = F_1. \tag{C-37}$$

For $B \le B_N$, the stream may not be decodable.

In summary, the bitstream is guaranteed to be decodable in the sense that the HRD buffer does not overflow or underflow, provided that the point (R,B) lies on or above the lower convex hull of the set of points $(0,B_1+R_1T)$, (R_1,B_1), …, (R_N,B_N), as illustrated in Figure C-6. The minimum start-up delay necessary to maintain this guarantee is $F_{min}(R) \div R$.

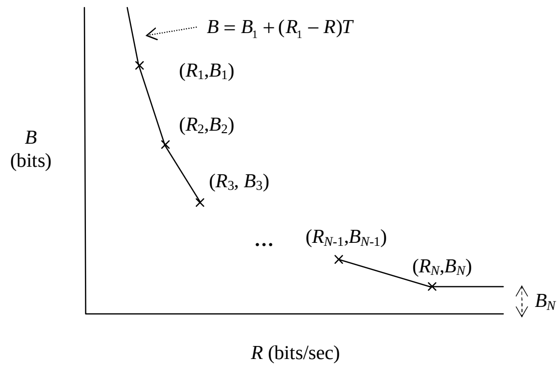


Figure C-6 – Further illustration of the leaky bucket concept

An HRD with buffer size B and initial decoder buffer fullness F with peak input rate R shall perform the tests $B \ge B_{min}(R)$ and $F \ge F_{min}(R)$, as specified above, for any conforming bitstream with LB parameters $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, and shall decode the bitstream provided that $B \ge B_{min}(R)$ and $F \ge F_{min}(R)$.

C.2.2.4 Encoder considerations

The encoder can create a bitstream that is contained by some given N leaky buckets, or it can simply compute N sets of leaky bucket parameters after the bitstream is generated, or a combination of these. In the former, the encoder enforces the N leaky bucket constraints during rate control. Conventional rate control algorithms enforce only a single leaky bucket constraint. A rate control algorithm that simultaneously enforces N leaky bucket constraints can be obtained by running a conventional rate control algorithm for each of the N leaky bucket constraints, and using as the current quantisation parameter (QP) the maximum of the QP's recommended by the N rate control algorithms.

Additional sets of leaky bucket parameters can always be computed after the fact (whether rate controlled or not), from the bitstream schedule for any given R_n, from the iteration specified in subclause C.2.2.2.

Annex D
Supplemental enhancement information

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

D.1 Introduction

This annex specifies 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 or display of video. 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 conformance to this Recommendation | International Standard.

Deleted: defines

Deleted: defines

Deleted: defines

D.2 SEI payload syntax

sei_payload(PayloadType, PayloadSize) {	**Category**	**Descriptor**
if(PayloadType = = 1)		
temporal_reference(PayloadSize)	7	
else if(PayloadType = = 2)		
clock_timestamp(PayloadSize)	7	
else if(PayloadType = = 3)		
panscan_rect(PayloadSize)	7	
else if(PayloadType = = 4)		
buffering_period(PayloadSize)	7	
else if(PayloadType = = 5)		
hrd_picture(PayloadSize)	7	
else if(PayloadType = = 6)		
filler_payload(PayloadSize)	7	
else if(PayloadType = = 7)		
user_data_registered_itu_t_t35(PayloadSize)	7	
else if(PayloadType = = 8)		
user_data_unregistered(PayloadSize)	7	
else if(PayloadType = = 9)		
random_access_point(PayloadSize)	7	
else if(PayloadType = = 10)		
ref_pic_buffer_management_repetition(PayloadSize)	7	
else if(PayloadType = = 11)		
spare_picture(PayloadSize)	7	
else if(PayloadType = = 12)		
scene_information(PayloadSize)	7	
else if(PayloadType = = 13)		
subseq_information(PayloadSize)	7	
else if(PayloadType = = 14)		
subseq_layer_characteristics(PayloadSize)	7	
else if(PayloadType = = 15)		
subseq_characteristics(PayloadSize)	7	
else		
reserved_sei_message(PayloadSize)	7	
if(!byte_aligned( )) {		
bit_equal_to_one	7	f(1)
while(!byte_aligned( ))		
bit_equal_to_zero	7	f(1)
}		
}		

D.2.1 Temporal reference SEI message syntax

temporal_reference(PayloadSize) {	**Category**	**Descriptor**
progressive_scan	7	u(1)
bottom_field_flag /* zero if progressive_scan is 1 */	7	u(1)
six_reserved_one_bits	7	f(6)
temporal_ref_value	7	u(v)
}		

D.2.2 Clock timestamp SEI message syntax

clock_timestamp(PayloadSize) {	**Category**	**Descriptor**
progressive_scan	7	u(1)
bottom_field_flag /* zero if progressive_scan is 1 */	7	u(1)
six_reserved_one_bits	7	f(6)
counting_type	7	u(5)
full_timestamp_flag	7	u(1)
discontinuity_flag	7	u(1)
count_dropped	7	u(1)
nframes	7	u(8)
if(full_timestamp_flag) {		
seconds_value /* 0,…,59 */	7	u(6)
minutes_value /* 0,…,59 */	7	u(6)
hours_value /* 0,…,23 */	7	u(5)
bit_count = 41		
} else {		
seconds_flag	7	u(1)
bit_count = 25		
if(seconds_flag) {		
seconds_value /* range 0,…,59 */	7	u(6)
minutes_flag	7	u(1)
bit_count += 7		
if(minutes_flag) {		
minutes_value /* 0,…,59 */	7	u(6)
hours_flag	7	u(1)
bit_count += 7		
if(hours_flag) {		
hours_value /* 0,…,23 */	7	u(5)
bit_count += 5		
}		
}		
}		
while(!byte_aligned( )) {		
bit_equal_to_one	7	f(1)
bit_count++		
}		
if(PayloadSize–(bit_count>>3) > 0)		
time_offset	7	i(v)
}		

D.2.3 Pan-scan rectangle SEI message syntax

pan_scan_rect(PayloadSize) {	**Category**	**Descriptor**
pan_scan_rect_id	7	e(v)
pan_scan_rect_left_offset	7	e(v)
pan_scan_rect_right_offset	7	e(v)
pan_scan_rect_top_offset	7	e(v)
pan_scan_rect_bottom_offset	7	e(v)
}		

D.2.4 Buffering period SEI message syntax

buffering_period(PayloadSize) {	**Category**	**Mnemonic**
seq_parameter_set_id	7	ue(v)
if(nal_hrd_flag = = 1) {		
for(k = 0; k <= pdb_count; k++)		
initial_pre_dec_removal_delay[k **]**	7	u(16)
}		
if(vcl_hrd_flag = = 1 **)** {		
for(k = 0; k <= pdb_count; k++)		
initial_pre_dec_removal_delay[k **]**	7	u(16)
}		
prev_buf_period_duration	7	ue(v)
}		

D.2.5 HRD picture SEI message syntax

hrd_picture(PayloadSize)	**Category**	**Descriptor**
pre_dec_removal_delay	7	ue(v)

D.2.6 Filler payload SEI message syntax

filler_payload(PayloadSize) {	**Category**	**Descriptor**
for(k = 0; k < PayloadSize; k++)		
filler_byte	7	f(8) = 0xFF
}		

D.2.7 User data registered by ITU-T Recommendation T.35 SEI message syntax

user_data_registered_itu_t_t35(PayloadSize) {	**Category**	**Descriptor**
itu_t_t35_country_code	7	b(8)
if(country_code ! = 0xFF)		
i = 1;		
else {		
itu_t_t35_country_code_extension_byte	7	b(8)
i = 2;		
}		
do {		
itu_t_t35_payload_byte	7	b(8)
i++		
} while(i < PayloadSize)		
}		

D.2.8 User data unregistered SEI message syntax

user_data_arbitrary(PayloadSize) {	**Category**	**Descriptor**
i = 0		
do {		
user_data_arbitrary_payload_byte	7	b(8)
i++		
} while(i < PayloadSize)		
}		

D.2.9 Random access point SEI message syntax

random_access_point(PayloadSize) {	**Category**	**Descriptor**
preroll_count	7	ue(v)
postroll_count	7	ue(v)
exact_match_flag	7	u(1)
broken_link_flag	7	u(1)
}		

D.2.10 Reference picture buffer management Repetition SEI message syntax

ref_pic_buffer_management_repetition(PayloadSize) {	**Category**	**Descriptor**
original_frame_num	7	u(v)
ref_pic_buffer_management( )		
}		

D.2.11 Spare picture SEI message syntax

spare_picture(PayloadSize) {	**Category**	**Descriptor**
delta_frame_num	7	ue(v)
num_spare_pics_minus1	7	ue(v)
for(i = 0; i < num_spare_pics_minus1+1; i++) {		
delta_spare_frame_num	7	ue(v)
ref_area_indicator	7	ue(v)
if(ref_area_indicator = = 1)		
for(j = 0; j < number_of_mbs_in_pic; j++)		
ref_mb_indicator	7	u(1)
else if(ref_area_indicator = = 2) {		
MbCnt = 0		
do {		
zero_run_length	7	ue(v)
MbCnt = MbCnt + zero_run_length + 1		
} while(MbCnt <= MaxMbAddress)		
}		
}		

Deleted: MAX_MB_ADDRESS

D.2.12 Scene information SEI message syntax

[Ed.Note: function more_sei_payload_data( ) should be specified in an approprite location.]

scene_information(PayloadSize) {	**Category**	**Descriptor**
scene_id	7	u(8)
scene_transition_type	7	ue(v)
if(scene_transition_type > 3)		
second_scene_id	7	u(8)
}		

D.2.13 Sub-sequence information SEI message syntax

subseq_information(PayloadSize) {	**Category**	**Descriptor**
subseq_layer_num	7	ue(v)
subseq_id	7	ue(v)
last_picture_flag	7	u(1)
if(more_sei_payload_data( ))		
stored_frame_cnt	7	ue(v)
}		

D.2.14 Sub-sequence layer characteristics SEI message syntax

subseq_layer_characteristics(PayloadSize) {	**Category**	**Descriptor**
do {		
average_bit_rate	7	u(16)
average_frame_rate	7	u(16)
} while(more_sei_payload_data( ))		
}		

D.2.15 Sub-sequence characteristics SEI message syntax

subseq_characteristics(PayloadSize) {	**Category**	**Descriptor**
subseq_layer_num	7	ue(v)
subseq_id	7	ue(v)
duration_flag	7	u(1)
if (duration_flag)		
subseq_duration	7	u(32)
average_rate_flag	7	u(1)
if (average_rate_flag) {		
average_bit_rate	7	u(16)
average_frame_rate	7	u(16)
}		
num_referenced_subseqs	7	ue(v)
for (n = 0; n < num_referenced_subseqs; n++) {		
ref_subseq_layer_num	7	ue(v)
ref_subseq_id	7	ue(v)
}		
}		

D.2.16 Reserved SEI message syntax

reserved_sei_message(PayloadSize) {	**Category**	**Descriptor**
for(i=0; i<PayloadSize; i++)		
reserved_sei_message_payload_byte	7	b(8)
}		

D.3 SEI payload semantics

D.3.1 Temporal reference SEI message semantics

progressive_scan: This parameter indicates whether the current picture has progressive or interlaced scan timing.

bottom_field_flag: When progressive_scan is 0, this parameter indicates whether the temporal reference is for the top (0) or bottom (1) field. Shall be 0 if progressive_scan is 1.

six_reserved_one_bits: Reserved for future backward-compatible use by ITU-T | ISO/IEC. Shall be equal to the binary string '111111' unless and until specified otherwise by ITU-T | ISO/IEC. A decoder conforming to this Recommendation | International Standard shall ignore the value of these bits.

temporal_ref_value: This parameter indicates a number of clock ticks as a multiplier of num_units_in_tick for the current time_scale. It is used for conveying local relative timing information.

The number of bytes used by temporal_ref_value shall remain constant for the video stream and shall be equal to PayloadSize – 1 bytes. For a temporal_ref_value encoded using *n* bytes, the temporal_reference contains the remainder of a clock tick counter modulo 256^n.

D.3.2 Clock timestamp SEI message semantics

The contents of the clock timestamp SEI message specify a time_offset which indicates the display or capture time computed as

$$\text{equivalent\_timestamp} = ((HH * 60 + MM) * 60 + \text{SS}) * \text{time\_scale} + NF * \text{num\_units\_in\_tick} + TO, \quad \text{(D-1)}$$

in units of ticks of a clock with clock frequency equal to time_scale Hz.

progressive_scan: This parameter indicates whether the current picture is in progressive or interlaced scan format.

bottom_field_flag: When progressive_scan is 0, this parameter indicates whether the temporal reference is for the top (0) or bottom (1) field. Shall be 0 if progressive_scan is 1.

six_reserved_one_bits: Reserved for future use by ITU-T | ISO/IEC. Shall be equal to the binary string '111111'. A decoder conforming to this Recommendation | International Standard shall ignore the value of these bits.

counting_type: A 5-bit parameter that specifies the method of dropping values of the nframes parameter as specified in Table D-1.

Table D-1 – Definition of counting_type values

Value (binary)	Interpretation
00000	no dropping of nframes count values and no use of time_offset
00001	no dropping of nframes count values
00010	dropping of individual zero values of nframes count
00011	dropping of individual max_pps values of npictures count
00100	dropping of the two lowest (value 0 and 1) nframes counts when seconds_value is zero and minutes_value is not an integer multiple of ten
00101	dropping of unspecified individual nframes count values
00110	dropping of unspecified numbers of unspecified nframes count values
00111 - 11111	reserved

full_timestamp_flag indicates whether the nframes parameter is followed by seconds_value or seconds_flag.

discontinuity_flag indicates whether the time difference between the current value of equivalent_timestamp and the value of equivalent_timestamp computed from the last previously-transmitted clock timestamp can be interpreted as a true time difference. A value of 0 indicates that the difference represents a true time difference.

count_dropped indicates the skipping of a count using the counting method specified by counting_type.

nframes indicates the value of *NF* used to compute the equivalent_timestamp. Shall be less than

$$\text{max\_fps} = \text{Ceil}(\text{time\_scale} \div \text{num\_units\_in\_tick}) \quad \text{(D-2)}$$

If counting_type is '00010' and count_dropped is 1, nframes shall be 1 and the value of nframes for the last previous picture in display order shall not be equal to 0 unless discontinuity_flag is equal to 1.

If counting_type is '00011' and count_dropped is 1, nframes shall be 0 and the value of nframes for the last previous picture in display order shall not be equal to max_fps – 1 unless discontinuity_flag is equal to 1.

If counting_type is '00100' and count_dropped is 1, nframes shall be 2 and the indicated value of *SS* shall be zero and the indicated value of *MM* shall not be an integer multiple of ten and nframes for the last previous picture in display order shall not be equal to 0 or 1 unless discontinuity_flag is equal to 1.

If counting_type is '00101' or '110' and count_dropped is 1, nframes shall not be equal to one plus the value of nframes for the last previous picture in display order modulo max_fps unless discontinuity_flag is equal to 1.

seconds_flag indicates whether seconds_value is present when full_timestamp_flag is 0.

seconds_value indicates the value of *SS* used to compute the equivalent_timestamp. Shall not exceed 59. If not present, the last previously-transmitted seconds_value shall be used as *SS* to compute the equivalent_timestamp.

minutes_flag indicates whether seconds_value is present when full_timestamp_flag is 0 and seconds_flag is 1.

minutes_value indicates the value of *MM* used to compute the equivalent_timestamp. Shall not exceed 59. If not present, the last previously-transmitted minutes_value shall be used as *MM* to compute the equivalent_timestamp.

hours_flag indicates whether seconds_value is present when full_timestamp_flag is 0 and seconds_flag is 1 and minutes_flag is 1.

hours_value indicates the value of *HH* used to compute the equivalent_timestamp. Shall not exceed 23. If not present, the last previously-transmitted hours_value shall be used as *HH* to compute the equivalent_timestamp.

bit_equal_to_one is a single bit which shall be equal to 1.

time_offset indicates the value of *TO* used to compute the equivalent_timestamp. The number of bytes used to represent time_offset shall be equal to PayloadSize – (bit_count >> 3), where bit_count is computed as specified in subclause D.2.2. If time_offset is not present, the value 0 shall be used as *TO* to compute the equivalent_timestamp.

D.3.3 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 $1/16^{th}$ sample spacing relative to the luma sampling grid.

pan_scan_rect_id contains an identifying number which may be used as specified externally 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_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 $1/16^{th}$ sample spacing relative to the luma sampling grid, the location of the pan-scan rectangle.

The pan-scan rectangle is specified, in units of $1/16^{th}$ sample spacing relative to the luma sampling grid, as the area of the rectangle with horizontal coordinates from 16 * cropping_rect_left + pan_scan_rect_left_offset to 16 * [16 * (pic_width_in_mbs_minus1 + 1) – cropping_rect_right] + pan_scan_rect_right_offset – 1 and with vertical coordinates from 16 * cropping_rect_top + pan_scan_rect_top_offset to 16 * [16 * (pic_height_in_mbs_minus1 + 1) – cropping_rect_bottom] + 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.

D.3.4 Buffering period SEI message semantics

A Buffering Period is specified as the set of pictures between two instances of the Buffering Period SEI message. The seq_parameter_set_id indicates the sequence parameter set that contains the sequence level HRD attributes.

seq_parameter_set_id indicates the sequence parameter set that contains the sequence level HRD attributes.

initial_pre_dec_removal_delay: This syntax element represents the delay between the time of arrival in the pre-decoder buffer of the first bit of the coded data associated with the first picture following the Buffering Period SEI message (including all NAL data in the case that the HRD pertains to the NAL) and the time of removal of the coded data associated with the picture from the pre-decoder buffer. It is in units of a 90 kHz clock. The initial_pre_dec_removal_delay syntax element is used in conjunction with the pre-decoder buffers as specified in Annex C. A value of zero is forbidden.

prev_buf_period_duration: This syntax element represents the duration of the subset of the video sequence contained in the previous Buffering Period. The interpretation of the syntax element is as a number of clock ticks (see Annex D). The prev_buf_period_duration syntax element is used in conjunction with the pre-decoder buffers as specified in Annex C. A value of zero is forbidden.

D.3.5 HRD picture SEI message semantics

pre_dec_removal_delay: This syntax element indicates how many clock ticks (see Annex C) to wait after removal from the HRD pre-decoder buffer of the previous picture before removing from the buffer the picture data immediately following the SEI message which contains the element. This value is also used to calculate an earliest possible time of arrival of picture data into the pre-decoder buffer, as specified in Annex C.

D.3.6 Filler payload SEI message semantics

This message contains a series of PayloadSize bytes of value 0xFF, which can be discarded.

filler_byte shall be a byte having the value 0xFF.

D.3.7 User data registered by ITU-T Recommendation T.35 SEI message semantics

This message contains registered user data as specified by ITU-T Recommendation T.35.

itu_t_t35_country_code shall be a byte having a value specified as a country code by ITU-T Recommendation T.35.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as an extended country code by ITU-T Recommendation T.35.

itu_t_t35_payload_byte shall be a byte containing user data registered as specified by ITU-T Recommendation T.35.

D.3.8 User data arbitrary SEI message semantics

This message contains arbitrary user data, the contents of which are not specified by this Recommendation | International Standard.

user_data_arbitrary_payload_byte shall be a byte having a value not specified by this Recommendation | International Standard.

NOTE - Users of this Recommendation | International Standard should exercise care in the use of the user data arbitrary SEI message to avoid the carriage of data content in a form likely to conflict with the data content format of other users (e.g., avoiding conflict by using a fixed multi-byte prefix identifier within the payload content).

D.3.9 Random access point SEI message semantics

The random access point SEI message indicates the recovery point of decoder output after starting decoding from a random access entry point. All decoded pictures at or subsequent to the recovery point in output order are indicated to be correct or approximately correct in content. Decoded pictures produced by starting the decoding process at the entry point may not be correct in content until the indicated recovery point, and the decoding process starting at the entry point and ending at the recovery point may contain references to pictures not available in the multi-picture buffer.

The entry point is indicated as a pre-roll count relative to the position of the SEI message in units of coded frame numbers prior to the frame number of the current picture. The recovery point is indicated as a post-roll count in units of coded pictures subsequent to the current picture at the position of the SEI message.

preroll_count indicates the entry point for the decoding process. Decoding should have started at or prior to the stored picture having the frame number equal to the frame number of the next slice minus the preroll_count in modulo MAX_FN arithmetic.

postroll_count indicates the recovery point of output. All decoded pictures in output order are indicated to be correct or approximately correct in content after the stored picture having the frame number equal to the frame number of the next slice incremented by postroll_count in modulo MAX_FN arithmetic.

exact_match_flag indicates whether decoded pictures at and subsequent to the recovery point in output order obtained by starting the decoding process at the specified entry point shall be an exact match to the pictures that would be produced by a decoder starting at the last prior IDR point in the NAL unit stream. The value 0 indicates that the match may not be exact and the value 1 indicates that the match shall be exact.

If decoding starts from an entry point indicated in a random access point SEI message, all references to unavailable stored pictures shall be inferred as references to sample values given by Y=Cb=Cr=128 (mid-level grey) for purposes of determining the conformance of the value of exact_match_flag.

broken_link_flag indicates the presence or absence of a splicing point in the NAL unit stream at the location of the random access point SEI message. If broken_link_flag is equal to 1, pictures produced by starting the decoding process at the last previous IDR point may contain undesirable visual artifacts due to splicing operations and should not be displayed until the indicated random access recovery point in output order. If broken_link_flag is equal to 0, no indication is given regarding any potential presence of visual artifacts.

If a sub-sequence information SEI message is transmitted in conjunction with a random access point SEI message in which broken_link_flag is equal to 1 and if subseq_layer_num is 0, subseq_id should be different from the latest subseq_id for subseq_layer_num equal to 0 that was decoded prior to the entry point. If broken_link_flag is equal to 0, the subseq_id in sub-sequence layer 0 should remain unchanged.

A buffering period SEI message should be transmitted at the location of the random access entry point indicated in the random access point SEI message in order to establish initialisation of the HRD buffer model.

D.3.10 Reference picture buffer management Repetition SEI message semantics

The Reference picture buffer management repetition SEI message is used to repeat memory management control operation commands that were located earlier in decoding order.

original_frame_num identifies the picture were the repeated memory management control operation originally occurred.

ref_pic_buffer_managament( ) shall contain a copy of the reference picture buffer management syntax elements of the picture whose frame_num was original_frame_num.

D.3.11 Spare picture SEI message semantics

The spare picture SEI message indicates that certain macroblocks, called spare decoded macroblocks, in one or more decoded stored pictures resemble the co-located macroblocks in a certain decoded picture, called the target picture, so much that any of these spare decoded macroblocks can be used to replace a co-located incorrect decoded macroblock in the target picture in the multi-frame buffer and in decoder output. Decoded pictures that contain spare macroblocks are called spare pictures.

The picture that contains the next slice or data partition in decoding order is herein referred to as the current picture. The frame_num of the current picture is herein denoted as CurrFrameNum.

delta_frame_num identifies the target picture whose spare pictures and macroblocks are specified later in the message. Let TargetFrameNum be the frame_num of the target picture, and the target picture is the stored picture having the TargetFrameNum. TargetFrameNum is calculated as follows

$$\begin{array}{l} \text{TargetFrameNum} = \text{CurrFrameNum} - \text{delta\_frame\_num} \\ \text{if(TargetFrameNum} < 0 \text{)} \\ \qquad \text{TargetFrameNum} = \text{MAX\_FN} + \text{TargetFrameNum} \end{array} \tag{D-3}$$

num_spare_pics_minus1 specifies the number of pictures which contain spare picture or macroblocks for the target picture.

delta_spare_frame_num specifies to which spare picture the following spare picture information in the current loop count belongs. For the first spare picture of the message, CandidateSpareFrameNum is equal to TargetFrameNum – 1 if TargetFrameNum is greater than 0 and MAX_FN – 1 otherwise. For later spare pictures, CandidateSpareFrameNum is the SpareFrameNum of the previous loop round minus 1 if SpareFrameNum is greater than 0 and MAX_FN – 1 otherwise. For each loop round, SpareFrameNum is calculated as follows:

$$\begin{array}{l} \text{SpareFrameNum} = \text{CandidateSpareFrameNum} - \text{delta\_spare\_frame\_num} \\ \text{if(SpareFrameNum} < 0 \text{)} \\ \qquad \text{SpareFrameNum} = \text{MAX\_FN} + \text{SpareFrameNum} \end{array} \tag{D-4}$$

ref_area_indicator specifies how the locations of spare macroblocks are coded. ref_area_indicator 0 indicates that all macroblocks of the spare picture are spare macroblocks.. ref_area_indicator 1 indicates an uncompressed spare macroblock map. ref_area_indicator 2 indicates a compressed spare macroblock map. A spare macroblock map consists of flags for each macroblock location of a picture. A flag shall be 0 if the macroblock location in the spare picture is a spare macroblock and 1 otherwise.

If ref_area_indicator is 1, there is a **ref_mb_indicator** for each macroblock address of the spare macroblock map in raster scan order. ref_mb_indicator 0 indicates that the macroblock is a spare macroblock, and ref_mb_indicator 1 indicates that the macroblock is not a spare macroblock.

If ref_area_indicator is 2, a spare macroblock map between a spare picture and the target picture is compressed. The coded macroblock map for loop_count equal to 0 is the spare macroblock map between the target picture and the first spare picture. A coded macroblock map for loop_count greater than 0 is generated by applying an exclusive or operation between the previous spare macroblock map and the current spare macroblock map. The coded macroblock map is scanned in counter-clockwise box-out order as specified in subclause 8.3.4.1. The number of consecutive zeros in the scanning order is indicated in **zero_run_length**.

D.3.12 Scene information SEI message semantics

A scene is herein specified as a set of pictures in decoding order captured with one camera. The scene information SEI message is used to label scenes with identifiers. The message concerns the next slice or data partition in decoding order.

scene_id: Pictures in a scene shall share the same value of scene_id. Consecutive scenes in decoding order should not have the same value of scene_id. If the next slice or data partition in decoding order belongs to a picture that includes contents from two scenes, scene_id is the scene identifier of the former scene in decoding order.

The following values of **scene_transition_type** are valid:

Table D-2 – Scene transition types.

Value	Description
0	No transition
1	Fade-out
2	Fade-in
3	Unspecified transition from or to constant color
4	Dissolve
5	Wipe
6	Unspecified mixture of two scenes
Other values	Reserved

If scene_transition_type is greater than 3, the next slice or data partition in decoding order belongs to a picture that includes contents from two scenes.

second_scene_id is present if the next slice or data partition in decoding order belongs to a picture that includes contents from two scenes. second_scene_id is the scene identifier of the latter scene in decoding order.

D.3.13 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 coded data stream. 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. The subjective quality increases 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 other sub-sequences and previous long-term reference pictures.

The sub-sequence information SEI message concerns the next slice or data partition in decoding order. The picture which the next slice or data partition belongs to is herein referred to as the target picture.

subseq_layer_num indicates the sub-sequence layer number of the target picture.

subseq_id identifies the sub-sequence within a layer. Consecutive sub-sequences within a particular layer in decoding order shall have a different subseq_id from each other.

last_picture_flag equal to 1 signals that the target picture is the last picture of the sub-sequence (in decoding order).

stored_frame_cnt is 0 for the first stored picture of the sub-sequence. For each coded frame belonging to the sub-sequence in decoding order, stored_frame_cnt shall be incremented by 1, in modulo MAX_FN operation, relative to the previous stored frame that belongs to the sub-sequence.

D.3.14 Sub-sequence layer characteristics SEI message semantics

The sub-sequence layer characteristics SEI message indicates the characteristics of sub-sequence layers.

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 signals the characteristics of sub-sequence layer 0. The second pair, if present, signals the characteristics of sub-sequence layers 0 and 1 jointly. Each pair in decoding order signals the characteristics for a range of sub-sequence layers from layer number 0 to the layer number that is incremented by one from the previous upper limit of layer numbers. The values are in effect from the point they are decoded until an update of the values is decoded.

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 calculated according to the decoding time of the NAL units. Value zero means an unspecified bit rate. [Ed. Note: It is unclear a) how bit rate is specified (i.e. how is the duration derived in order to compute rate from a number of bits), and b) what is the intended use of this bit rate.]

average_frame_rate gives the average frame rate in frames/(256 seconds) of the sub-sequence layer. Value zero indicates an unspecified frame rate.

D.3.15 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.

This message applies to the next sub-sequence in decoding order having the indicated **subseq_layer_num** and **subseq_id**. This sub-sequence is herein called the target sub-sequence.

duration_flag equal to zero indicates that the duration of the target sub-sequence is not specified.

subseq_duration indicates the duration of the target sub-sequence in clock ticks of a 90-kHz clock.

average_rate_flag equal to zero indicates that the average bit rate and the average frame rate of the target sub-sequence are unspecified.

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 calculated according to the decoding time of the NAL units. [Ed. Note: See Previous editors note.]

average_frame_rate gives the average frame rate in frames/(256 seconds) of the current sub-sequence.

num_referenced_subseqs gives the number of sub-sequences which contain pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence.

ref_subseq_layer_num and **ref_subseq_id** identify a sub-sequence that contains pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence.

D.3.16 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)

E.1 Introduction

This Annex specifies those parts of the sequence parameter set and the picture 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 sequence parameters for conformance to this Recommendation | International Standard.

For each 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 the parameters become effective.

E.2 VUI syntax

E.2.1 VUI sequence parameters syntax

vui_seq_parameters( ) {	**Category**	**Descriptor**
aspect_ratio_info_flag	0	u(1)
if(aspect_ratio_info_flag) {		
aspect_ratio_info	0	b(8)
if(aspect_ratio_info = = "Extended SAR") {		
sar_width	0	u(8)
sar_height	0	u(8)
}		
}		
video_signal_type_flag	0	u(1)
if(video_signal_type_flag) {		
video_format	0	u(3)
video_range_flag	0	u(1)
colour_description_flag	0	u(1)
if(colour_description_flag) {		
colour_primaries	0	b(8)
transfer_characteristics	0	b(8)
matrix_coefficients	0	b(8)
}		
}		
chroma_location_flag	0	u(1)
if (chroma_location_flag) {		
chroma_location_frame	0	e(v)
chroma_location_field	0	e(v)
}		
timing_information_flag		
if(timing_information_flag) {		
num_units_in_tick	0	u(32)
time_scale	0	u(32)
fixed_frame_rate_flag	0	u(1)
}		
nal_hrd_flag	0	u(1)
if(**nal_hrd_flag** = = 1)		
hrd_parameters( )		
vcl_hrd_flag	0	u(1)
if(**vcl_hrd_flag** = = 1)		
hrd_parameters( )		
if((**nal_hrd_flag** = = 1 || (**vcl_hrd_flag** = = 1)) {		
low_delay_hrd	0	u(1)
removal_time_tolerance	0	ue(v)
}		
bitstream_restriction_flag	0	u(1)
if(bitstream_restriction_flag) {	0	u(1)
motion_vectors_over_pic_boundaries_flag	0	u(1)
minimum_compression_per_pic_reversed	0	e(v)
minimum_compression_per_macroblock_reversed	0	e(v)
log2_maximum_mv_length_vertical	0	e(v)
log2_maximum_mv_length_horizontal	0	e(v)

}		
}		

E.2.2 HRD parameters syntax

hrd_parameters( ) { /* coded picture buffer parameters */		
pdb_cnt	0	ue(v)
bit_rate_scale	0	u(4)
coded_pic_buffer_size_scale	0	u(4)
for(k=1; k<=pdb_cnt; k++) {		
bit_rate_value[k **]**	0	ue(v)
coded_pic_buffer_size_value[k **]**	0	ue(v)
vbr_cbr_flag[k]	0	u(1)
}		
}		

E.2.3 VUI picture parameters syntax

vui_pic_parameters( ) {	**Category**	**Descriptor**
frame_cropping_flag	1	ue(v)
if(frame_cropping_flag) {		
frame_cropping_rect_left_offset	1	ue(v)
frame_cropping_rect_right_offset	1	ue(v)
frame_cropping_rect_top_offset	1	ue(v)
frame_cropping_rect_bottom_offset	1	ue(v)
}		
}		

E.3 VUI semantics

E.3.1 VUI sequence parameters semantics

aspect_ratio_info_flag: A flag that, when 1, signals the presence of the aspect_ratio_info. If the flag is 0, then the following default values shall apply: aspect_ratio_info = 0.

aspect_ratio_info is an eight-bit integer which specifies the value of sample aspect ratio. Table E-1shows the meaning of the code. If aspect_ratio_info indicates Extended SAR, sample_aspect_ratio is represented by sar_width and sar_height. The sar_width and sar_height shall be relatively prime. If aspect_ratio_info is zero or if either sar_width or sar_height are zero, the sample aspect ratio shall be considered unspecified or specified externally.

Deleted: defines

Table E-1 – Meaning of sample aspect ratio

aspect_ratio_info	Sample aspect ratio
0000 0000	Unspecified or specified externally
0000 0001	1:1 (“Square”)
0000 0010	12:11 (625-type for 4:3 picture)
0000 0011	10:11 (525-type for 4:3 picture)
0000 0100	16:11 (625-type stretched for 16:9 picture)
0000 0101	40:33 (525-type stretched for 16:9 picture)
0000 0110	24:11 (Half-wide 4:3 for 625)
0000 0111	20:11 (Half-wide 4:3 for 525)
0000 1000	32:11 (Half-wide 16:9 for 625)

0000 1001	80:33 (Half-wide 16:9 for 525)
0000 1010	18:11 (2/3-wide 4:3 for 625)
0000 1011	15:11 (2/3-wide 4:3 for 525)
0000 1100	24:11 (2/3-wide 16:9 for 625)
0000 1101	20:11 (2/3-wide 16:9 for 525)
0000 1110	16:11 (3/4-wide 4:3 for 625)
0000 1111	40:33 (3/4-wide 4:3 for 525)
0001 0000	64:33 (3/4-wide 16:9 for 625)
0001 0001	160:99 (3/4-wide 16:9 for 525)
0001 0010 to 1111 1110	Reserved
1111 1111	Extended SAR

sar_width is an 8-bit unsigned integer which indicates the horizontal size of sample aspect ratio. A zero value is forbidden.

sar_height is an 8-bit unsigned integer which indicates the vertical size of sample aspect ratio. A zero value is forbidden.

video_signal_type_flag: A flag that, when 1, signals the presence of video signal information. If video_signal_type_flag is 0, then the following default values shall apply: video_format = '101', video_range = 0, colour_description = 0.

video_format: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this Recommendation | International Standard. Its meaning is specified in Table E-2. If the video_signal_type( ) is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

Table E-2 – Meaning of video_format

video_format	Meaning
000	Component
001	PAL
010	NTSC
011	SECAM
100	MAC
101	Unspecified video format
110	Reserved
111	Reserved

video_range_flag indicates the nominal black level and range of the luminance and chrominance signals as derived from E'_Y, E'_{PB}, and E'_{PR} analogue component signals as follows:

If video_range_flag=0:

$$Y = \text{round}(219 * E'_Y + 16)$$
$$Cb = \text{round}(224 * E'_{PB} + 128)$$
$$Cr = \text{round}(224 * E'_{PR} + 128)$$

If video_range_flag=1:

$$Y = \text{round}(255 * E'_Y)$$
$$Cb = \text{round}(255 * E'_{PB} + 128)$$
$$Cr = \text{round}(255 * E'_{PR} + 128)$$

If video_signal_type_flag is zero, video_range shall be inferred to have value 0 (a nominal range of Y from 16 to 235).

colour_description_flag which if set to '1' indicates the presence of colour_primaries, transfer_characteristics and matrix_coefficients in the bitstream.

colour_primaries: This 8-bit integer describes the chromaticity coordinates of the source primaries, and is specified in Table E-3.

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 video Image characteristics are unknown.
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	SMPTE 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	SMPTE 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)
9-255	Reserved

If video_signal_type_flag is zero or colour_description is zero, the chromaticity is unspecified or specified externally.

transfer_characteristics: This 8-bit integer describes the opto-electronic transfer characteristic of the source picture, and is specified in Table E-4.

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 $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
2	Unspecified video Image characteristics are unknown.
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	SMPTE 170M $V = 1{,}099\ L_C^{0{,}45} - 0{,}099$ for $1 \geq L_C \geq 0{,}018$ $V = 4{,}500\ L_C$ for $0{,}018 > L_C \geq 0$
7	SMPTE 240M (1987) $V = 1{,}1115\ L_C^{0{,}45} - 0{,}1115$ for $L_C \geq 0{,}0228$ $V = 4{,}0\ L_C$ for $0{,}0228 > L_C$
8	Linear transfer characteristics i.e. $V = L_C$
9	Logarithmic transfer characteristic (100:1 range) $V = 1.0 - \log_{10}(Lc)/2$ for $1 = L_C = 0.01$ $V = 0.0$ for $0.01 > L_C$
10	Logarithmic transfer characteristic (316.22777:1 range) $V = 1.0 - \log_{10}(Lc)/2.5$ for $1 = L_C = 0.0031622777$ $V = 0.0$ for $0.0031622777 > L_C$
11-255	Reserved

If video_signal_type_flag is zero or colour_description is zero, the transfer characteristics are unspecified or are specified externally.

matrix_coefficients: This 8-bit integer describes the matrix coefficients used in deriving luminance and chrominance signals from the green, blue, and red primaries, as specified in Table E-5.

Using this table:

E'_Y is analogue with values between 0 and 1
E'_R, E'_G, and E'_B are analogue with values between 0 and 1
E'_{PB} and E'_{PR} are analogue between the values -0,5 and 0,5
White is specified as $E'_R = E'_G = E'_B = 1$
White equivalently given by $E'_Y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$
$E'_Y = K_R * E'_R + (1 - K_R - K_B) * E'_G + K_B * E'_B$
$E'_{PB} = 0.5(E'_R - E'_Y) \div (1-K_R)$
$E'_{PB} = 0.5 (E'_B - E'_Y) \div (1-K_B)$

Table E-5 – Matrix Coefficients

Value	Matrix
0	Reserved
1	ITU-R Recommendation BT.709 K_G = 0,7152; K_R = 0,2126
2	Unspecified video Image characteristics are unknown.
3	Reserved
4	FCC K_G = 0,59; K_R =0,30
5	ITU-R Recommendation BT.470-2 System B, G: K_G = 0,587; K_R = 0,299
6	SMPTE 170M K_G = 0,587; K_R = 0,299
7	SMPTE 240M (1987) K_G = 0,701; K_R = 0,212
8-255	Reserved

If video_signal_type_flag is zero or colour_description is zero, the matrix coefficients are assumed to be unspecified or specified externally.

chroma_location_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_location_frame = 0, chroma_location_field = 0.

chroma_location_frame specifies the 4:2:0 sampling structure according to Table E-66 and Figure E-1.

Table E-6 – Chroma Sampling Structure Frame

Value	Sampling Structure
0	unspecified
1	Frame according to Figure E-1 Chroma Sample Mode 1
2	Frame according to Figure E-1 Chroma Sample Mode 2
3	Frame according to Figure E-1 Chroma Sample Mode 3

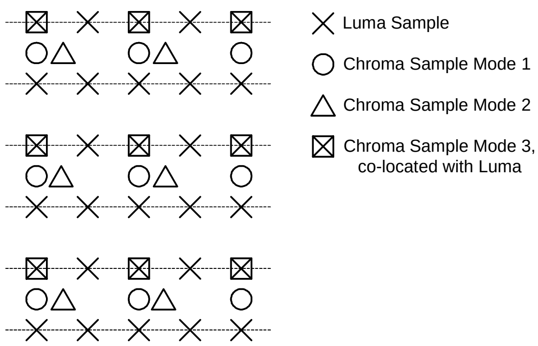


Figure E-1 – Luma and chroma sample types

chroma_location_field specifies the 4:2:0 sampling structure according to Table E-7and Figure E-2.

Table E-7 – Chroma Sampling Structure Frame

Value	Sampling Structure
0	unspecified
1	Frame according to Figure E-2 Chroma Sample Mode 1
2	Frame according to Figure E-2 Chroma Sample Mode 2
3	Frame according to Figure E-2 Chroma Sample Mode 3

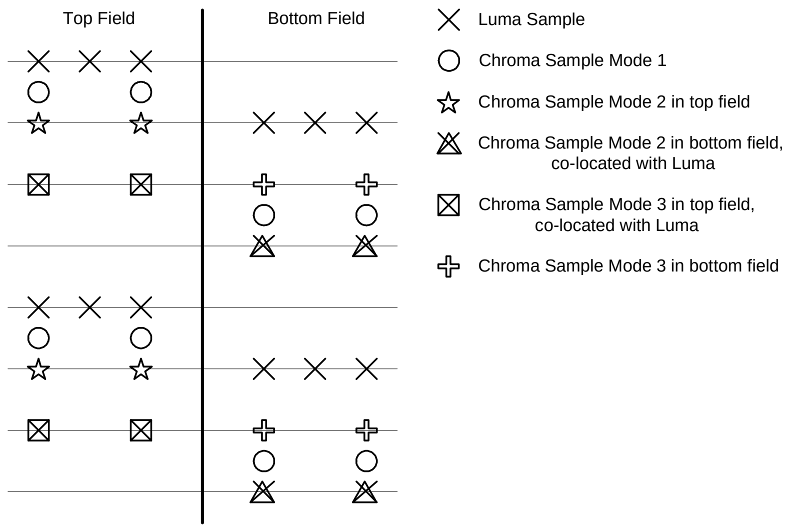


Figure E-2 – Luma and chroma association

timing_information_flag: A flag that, when 1, signals the presence of time unit information. If timing_information_flag is set to 0, then the following default values shall apply: num_units_in_tick = 0, time_scale = 0, fixed_frame_rate = 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 (30 000) ÷ 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 which 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 0, the duration of the clock tick specified above is unspecified.

fixed_frame_rate_flag: 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 as specified in Annex C is a constant. If equal to 0, the temporal distances between HRD output times of consecutive frames or fields in output order as specified in Annex C may not be constant.

nal_hrd_flag: If nal_hrd_flag == '1', the multiplexed NAL and VCL stream complies with a hypothetical reference decoder (HRD) as specified in Annex C. In this case, the HRD parameters follow the nal_hrd_flag in the sequence parameter set syntax. If nal_hrd_flag == '0', the multiplexed NAL and VCL stream is not guaranteed to comply with an HRD. No default values are specified.

NOTE - If nal_hrd_flag == 0 the maximum buffer sizes and bit rates specified in Annex A apply.

vcl_hrd_flag: If vcl_hrd_flag == '1', the VCL bitstream complies with a hypothetical reference decoder (HRD) as specified in Annex C. In this case, the HRD parameters follow the vcl_hrd_flag in the sequence parameter set syntax. If vcl_hrd_flag == '0', the VCL bitstream is not guaranteed to comply with an HRD.

NOTE - If vcl_hrd_flag == 0 the maximum buffer sizes and bit rates specified in Annex A apply.

removal_time_tolerance: This syntax element indicates the number of clock ticks (see Annex C) of deviation allowed between the pre-decoder buffer removal times (see subclauses C.2.5??? and C.3.5???) and the accumulated Buffering Period capture time (see subclauses C.2.4??? and C.3.4???). It is encoded as a universal VLC, with all values allowed. A value of '0' implies that, at each measurement point (*i.e.* each picture preceding a Buffering Period SEI message), the removal time shall exactly match the capture time.

bitstream_restriction_flag: A flag that, when 1, signals the presence of bitstream restriction information. If bitstream_restriction_flag is set to 0, then the following default values shall apply: motion_vectors_over_pic_boundaries_flag = 1, minimum_compression_per_pic_reversed = 4, minimum_compression_per_macroblock_reversed = 1, log2_maximum_mv_length_vertical = 16, log2_maximum_mv_length_horizontal = 16.

motion_vectors_over_pic_boundaries_flag equal to 0 indicates that no motion vector refers to samples outside the picture boundaries. motion_vectors_over_pic_boundaries_flag equal to 1 indicates that motion vectors may refer to samples outside the picture boundaries.

minimum_compression_per_pic_reversed and **minimum_compression_per_macroblock_reversed** advise the decoder about the minimum compression ratio (corresponding to a maximum coded picture or macroblock size respectively). A value of n for either of the two indicates a minimum compression ratio of 1:n. Annex A specifies the numbering range for both values.

Deleted: defines

log2_maximum_mv_length_vertical and **log2_maximum_mv_length_horizontal** indicate the maximum value of the absolute of a non-predicted vertical or horizontal motion vector component, in units of either ¼ or 1/8 sample, depending on the value of motion_vector_resolution. A value of n asserts that no absolute value of a motion vector component is bigger than 2**n ¼ pel or $1/8^{th}$ pel units. Note: the high default value is restricted in Annex A for some profile/level combinations. Furthermore, the maximum vector length is restricted by the picture size.

E.3.2 HRD parameters semantics

pdb_cnt: This syntax element indicates the number of pre-decoder buffers (PBDs) in the HRD. A value of pdb_cnt equal to '0' is not allowed.

bit_rate_scale: Together with bit_rate_value[k], this syntax element specifies the maximum input bit rate of the k-th PDB in an HRD.

Deleted: defines

bit_rate_value[k**]**: Together with bit_rate_scale, this syntax element specifies the maximum input bit rate of the k-th PDB in an HRD. The actual bit rate in bits per second is given by:

Deleted: defines

$$\text{bit\_rate[k]} = \text{bit\_rate\_value[k]} * 2^{(6 + \text{bit\_rate\_scale})}. \qquad \text{(E-1)}$$

coded_pic_buffer_size_value is used together with coded_pic_buffer_size_scale[k] to define the maximum input bit rate of the k-th PDB in an HRD.

coded_pic_buffer_size_scale[k**]** is used together with coded_pic_buffer_size_value to define the pre-decoder buffer size of the k-th PDB in an HRD. The actual buffer size in bits is given by

$$\text{coded\_pic\_buffer\_size[k]} = \text{coded\_pic\_buffer\_size\_value[k]} * 2^{(4 + \text{coded\_pic\_buffer\_size\_scale})}. \qquad \text{(E-2)}$$

vbr_cbr_flag: If equal to '0', this syntax element indicates that the pre-decoder buffer operates in variable bit rate (VBR) mode. If equal to '1', it indicates constant bit rate (CBR) operation.

low_delay_hrd: If low_delay_hrd is equal to '0', the HRD operates in delay-tolerant mode. If low_delay_hrd is equal to '1', the HRD operates in low-delay mode. In low-delay mode, only one HRD buffer may be selected and big pictures which violate the HRD removal time rules at the pre-decoder buffer are permitted. It is expected that such big pictures occur only occasionally, but not mandatory.

E.3.3 VUI picture parameters semantics

frame_cropping_flag when 1, signals the presence of bitstream restriction information. If frame_cropping_flag is set to 0, then the following default values shall apply frame_cropping_rect_left = 0, frame_cropping_rect_right = 0, frame_cropping_rect_top = 0, frame_cropping_rect_bottom = 0.

frame_cropping_rect_left, frame_cropping_rect_right, frame_cropping_rect_top, frame_cropping_rect_bottom define the area of the luma picture internal array which shall be the output of the decoding process. The decoded values of these offsets consist of non-negative integer values, and the output of the decoding process is specified as the area within the rectangle containing luma samples with horizontal coordinates from cropping_rect_left to 16*(pic_width_in_mbs_minus1 + 1)-(cropping_rect_right + 1) and with vertical coordinates from cropping_rect_top to 16*(pic_height_in_mbs_minus1 + 1)-(cropping_rect_bottom + 1), inclusive.

Page xv: [1] Deleted	wiegand	9/19/2002 4:55 PM

Obtaining good image quality at the bit rates of interest demands very high compression, which is not achievable with intra picture coding alone. The need for random access, however, is best satisfied with pure intra picture coding. The choice of the techniques is based on the need to balance a high image quality and compression capability with the requirement to allow random access into the coded video data stream.

A

Page xv: [2] Deleted	wiegand	9/19/2002 4:53 PM

Motion compensation is used both for causal prediction of a current picture from one or more previous pictures, and for non-causal prediction from future pictures in decoder output order.

Page xv: [3] Deleted	wiegand	9/19/2002 4:42 PM

coded pictures (I-pictures) are coded

Page xv: [4] Deleted	wiegand	9/19/2002 4:44 PM

Bi-predictive pictures (B-pictures) provide the highest degree of compression but require a higher degree of memory access capability in the decoding process, as each block of sample values in a B picture may be predicted using a weighted average of two blocks of motion-compensated sample values.

Page xv: [5] Deleted	wiegand	9/19/2002 4:45 PM

Figure Intro-1 illustrates one limited and example of the relationship among the three different picture types. Significantly different inter-picture dependency relationships are also allowed at the discretion of the encoder within limits specified by the profile and level.

Page xv: [6] Deleted	wiegand	9/19/2002 4:43 PM

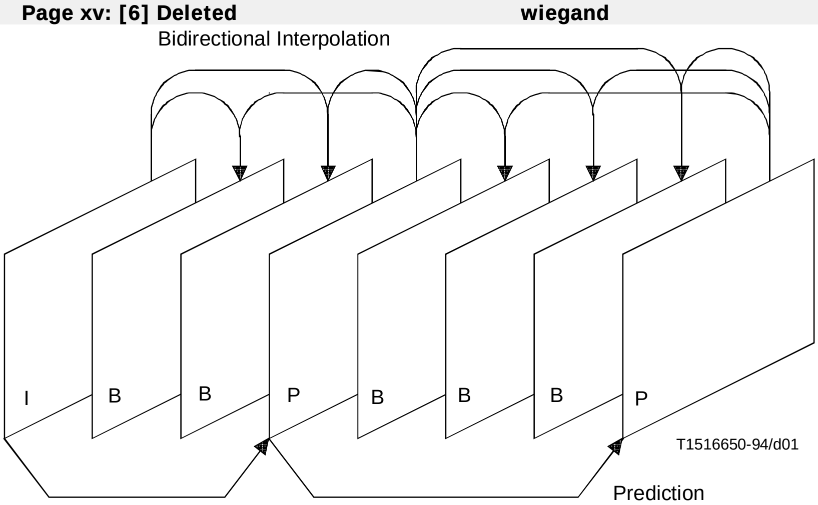


Figure Intro. 1 – Example of temporal picture structure

[Ed.Note: This needs to be replaced by a Fig that reflects temporal dependencies of this rec / is.]

Page xv: [7] Change	wiegand	9/19/2002 5:50 PM

Formatted Bullets and Numbering

Page 45: [8] Deleted	wiegand	9/19/2002 5:41 PM

					na

Page 74: [9] Deleted	wiegand	9/19/2002 2:10 PM

8.5.3.1Mode 0: vertical prediction

$F(0,-1) = (P(0,-1) + P(1,-1) + 1) >> 1$ (8-25)

$F(x,-1) = (P(x-1,-1) + 2 \times P(x,-1) + P(x+1,-1) + 2) >> 2$, x=1,…,6(8-26)

$F(7,-1) = (P(6,-1) + P(7,-1) + 1) >> 1$ (8-27)

Pred(x, y) = *F*(x, -1), x, y=0..7 (8-28)

8.5.3.2Mode 1: horizontal prediction

$F(-1,0) = (P(-1,0) + P(-1,1) + 1) >> 1$ (8-29)

$F(-1,y) = (P(-1,y-1) + 2 \times P(-1,y) + P(-1,y+1) + 2) >> 2$, y=1,…,6 (8-30)

$F(-1,7) = (P(-1,6) + P(-1,7) + 1) >> 1$ (8-31)

Pred(x, y) = *F*(-1, y), x, y=0..7 (8-32)

8.5.3.3Mode 2: DC prediction

If all samples P(-1,*n*) and P(*n*,-1) used in Equation 8-33 are available, the prediction is formed as

Pred(x, y) = $\left(\left(\sum_{n=0}^{7}(P(-1,n)+P(n,-1))\right)+8\right) >> 4$ x,y=0..7, (8-33)

If the 8 samples P(-1,*n*) are not available, the prediction is formed as

Pred(x,y) = $\left[\left(\sum_{n=0}^{7}P(n,-1)\right)+4\right]$ >> 3 x,y=0..7, (8-33a)

If the 8 samples P(*n*,-1) are not available, the prediction is formed as

Pred(x,y) = $\left[\left(\sum_{n=0}^{7}P(n,-1)\right)+4\right]$ >> 3 x,y=0..7, (8-33b)

If all 16 samples are not available, the prediction Pred(x,y) for all samples x,y=0..7is 128.

8.5.3.4Mode 3: plane prediction

For the plane mode, the prediction is formed as:

Pred(i,j) = Clip1((a + b·(i-3) + c·(j-3) +16) >> 5), i,j=0,…,7 (8-34)

where:

a = 16·(P(-1,7) + P(7,-1)) (8-35)

b = (17*H+16)>>5) (8-36)

c = (17*V+16)>>5) (8-37)

and H and V are specified as:

$$H = \sum_{i=1}^{4} i \cdot (P(3+i,-1) - P(3-i,-1)) \qquad (8\text{-}38)$$

$$V = \sum_{j=1}^{4} j \cdot (P(-1,3+j) - P(-1,3-j)) \qquad (8\text{-}39)$$

Page 81: [10] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [10] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [11] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [12] Deleted	**wiegand**	**9/19/2002 2:42 PM**

72

Page 81: [12] Deleted	**wiegand**	**9/19/2002 2:42 PM**

73

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [13] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [14] Formatted	**wiegand**	**9/19/2002 2:46 PM**

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [14] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [15] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [15] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [15] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [15] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [16] Deleted	wiegand	9/19/2002 2:42 PM

71

Page 81: [16] Deleted	wiegand	9/19/2002 2:42 PM

75

Page 81: [17] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [17] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [17] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [17] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [17] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [18] Deleted	wiegand	9/19/2002 2:43 PM

Page 81: [18] Deleted	wiegand	9/19/2002 2:31 PM

76

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [19] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [20] Deleted	wiegand	9/19/2002 2:42 PM

77

Page 81: [20] Deleted	wiegand	9/19/2002 2:42 PM

78

Page 81: [21] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [21] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [21] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [21] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [22] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [23] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [23] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [23] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [23] Formatted	wiegand	9/19/2002 2:46 PM

Font: Not Italic

Page 81: [24] Deleted	wiegand	9/19/2002 2:43 PM

76

Page 81: [24] Deleted	wiegand	9/19/2002 2:43 PM

80