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

3rd Meeting: Fairfax, Virginia, USA, 6-10 May, 2002

Document **JVT-C167**
File: JVT-C167.doc
Generated: 2002-05-10

Title: Joint Committee Draft (CD)

Status: Approved Output Document

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: Text of Committee Draft of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC)

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 ITU-T (the ITU Telecommunication Standardisation Sector) is a permanent organ of the International Telecommunications Union (ITU). The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to developing telecommunication standards on a world-wide basis. The World Telecommunication Standardisation Conference, which meets every four years, establishes the program of work arising from the review of existing questions and new questions among other things. The approval of new or revised Recommendations by members of the ITU-T is covered by the procedure laid down in the ITU-T Resolution No. 1 (Helsinki 1993). The proposal for Recommendation is accepted if 70% or more of the replies from members indicate approval.

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 specification is a committee draft that is being submitted for approval to the ITU-T, ISO/IEC JTC1/SC29. It was prepared jointly by ITU-T SG16 Q.6 also known as VCEG (Video Coding Experts Group) and by SC29/WG11, also known as MPEG (Moving Pictures Expert Group). VCEG was formed in 1997 to develop 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 specification Annex A, Annex B, and Annex C contain normative requirements and are an integral part of this specification.

Introduction

Introduction.

1 Scope

This document specifies ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 video coding. The basic configuration of the algorithm is similar to H.263 and ISO/IEC 14496-2.

2 Normative References

Normative References.

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-field picture**: A field structure B-Picture.

3.3 **B-frame picture**: A frame structure B picture.

3.4 **B picture**: A predictive-coded picture; A picture that is coded in a manner in which some macroblocks may use a weighted average of two distinct motion-compensated prediction values for the prediction of the macroblock sample values.

3.5 **backward motion vector**: A motion vector used for backward prediction.

3.6 **backward prediction**: Interframe prediction of the content of a B picture using a reference picture index into the backward reference set.

3.7 **backward reference set**: The ordered set of reference frames or fields defined for use in interframe backward prediction for a B picture. The backward reference set is one of two ordered sets of reference pictures used by a B picture, with the other being the forward reference set.

3.9 **bitstream (stream)**: An ordered series of bits that forms the coded representation of the encoded data.

3.10 **block**: An N-column by M-row array of samples, or NxM transform coefficients (source, quantised, or scaled).

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

3.12 **byte**: 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.

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

3.14 **channel**: A digital medium that stores or transports a bitstream constructed according to this Specification.

3.15 **chroma component**: An array or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chroma signals are Cr and Cb.

3.16 **chroma format**: Defines the number of chroma blocks in a macroblock.

3.17 **coded frame**: A coded frame is a coded I frame, a coded P frame or a coded B frame.

3.18 **coded I frame**: An I-frame picture or a pair of field pictures, where the first field picture is an I picture and the second field picture is an I picture or a P picture.

3.19 **coded B frame**: A B-frame picture or a pair of B-field pictures.

3.20 **coded order**: The order in which the pictures are transmitted and decoded. This order is not necessarily the same as the display order.

3.21 **coded P frame**: A P-frame picture or a pair of P-field pictures.

3.19 **coded picture**: A coded picture is made of a picture header, the optional extensions immediately following it, and the following picture data. A coded picture may be a coded frame or a coded field.

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

3.21 **coded video bitstream**: A coded representation of a series of one or more pictures as defined in this Specification.

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

3.24 **compression**: Reduction in the number of bits used to represent an item of data.

3.26 **constant bit rate**: Operation where the bit rate is constant from start to finish of the coded bitstream.

3.28 **data partitioning**: A method of grouping syntax elements

3.29 **DC coefficient**: The transform coefficient for which the frequency index is zero in both dimensions.

3.30 **decoder input buffer**: The first-in first-out (FIFO) buffer specified in the video buffering verifier.

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

3.32 **decoding (process)**: The process defined in this Specification that reads an input coded bitstream and produces decoded pictures or audio samples.

3.34 **display aspect ratio**: The ratio height/width (in SI units) of the intended display.

3.35 **display order**: The order in which the decoded pictures are intended to be displayed.

3.36 **display process**: The (non-normative) process by which reconstructed frames are displayed.

3.37 **editing**: The process by which one or more coded bitstreams are manipulated to produce a new coded bitstream. Conforming edited bitstreams must meet the requirements defined in this Specification.

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

3.39 **encoding (process)**: A process, not specified in this Specification, that reads a stream of input pictures and produces a valid coded bitstream as defined in this Specification.

3.42 **field (structure) picture**: A field structure picture is a coded picture with picture_structure is equal to “Top field” or “Bottom field”.

3.44 **field**: A “field” is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.

3.45 **field-based prediction**: A prediction mode using only one field of the reference frames.

3.46 **flag**: A variable which can take one of only the two values defined in this Specification.

3.48 **forward motion vector**: A motion vector used for forward prediction.

3.49 **forward prediction**: Interframe prediction of the content of a P, B, or SP picture using a reference picture index into the forward reference set. All interframe prediction used for P and SP pictures is considered forward prediction. The forward reference set is one of two ordered sets of reference pictures used by a B picture, with the other being the backward reference set.

3.50 **forward reference set**: The ordered set of reference frames or fields defined for use in interframe forward prediction for a P, B, or SP picture.

3.51 **forward transform**: A part of a (non-normative) example encoding process by which blocks of spatial-domain samples are converted into frequency-domain transform coefficients for subsequent quantisation.

3.52 **frame**: A frame contains the lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.

3.53 **frame (structure) picture**: A frame structure picture is a coded picture with picture_structure is equal to “Frame”.

3.56 **frame reordering**: The process of reordering the reconstructed frames when the coded order is different from the display order.

3.57 **frame-based prediction**: A prediction mode using both fields of the reference frame.

3.58 **future reference frame (field)**: A future reference frame (field) is a reference frame (field) that occurs at a later time than the current picture in display order.

3.59 **group of pictures**: a group of pictures starting at an instantaneous decoder refresh point.

3.60 **header**: A block of data in the coded bitstream containing the coded representation of a number of data elements pertaining to the coded data that follow the header in the bitstream.

3.61 **I-field picture**: A field structure I-Picture.

3.62 **I-frame picture**: A frame structure I picture.

3.62 **independent group of pictures (independent GOP)**: A group of pictures that can be decoded independently from previous or later pictures. The start of an independent GOP is explicitly signalled.

3.62 **instantaneous decoder refresh**: Decoding of a frame that results into a completely refreshed picture. Later coded frames can be decoded without referencing to data prior to the frame.

3.63 **interlace**: The property of conventional television frames where alternating lines of the frame represent different instances in time. In an interlaced frame, one of the fields is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.

3.64 **intra coding**: Coding of a macroblock or picture that uses information only from that macroblock or picture.

3.65 **I picture, intra-coded picture**: A picture coded using information only from itself.

3.66 **scaling**: The process of rescaling the transform coefficient levels after their representation in the bitstream has been decoded.

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

3.67 **layer:** One of a set of conceptual entities in a non-branching hierarchical relationship. Higher layers contain lower layers. There is a set of system layers and a set of coding layers. The system layers are the system layer, video coding layer (VCL), network abstraction layer (NAL), and transport layers. The coding layers are the sequence, GOP, picture, picture layer reference picture selection (picture layer RPSL), slice, slice layer reference picture selection (slice layer RPSL), macroblock, 8x8 block and 4x4 block layers.

3.68 **level**: A defined set of constraints on the values which may be taken by the parameters of this Specification. 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 absolute value of a non-zero coefficient (see “run”). In a third context, a level is the lowest coding layer in which a syntax element appears.

3.68 **long SCP:** A start code prefix which uniquely indicates the start of a picture or higher coding layer start code. It is one byte longer than a short SCP.

3.69 **luma component**: An array or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luma is Y.

3.70 **macroblock**: The 16x16 samples of luma data and the two corresponding 8x8 samples of chroma.

3.71 **Mbit**: 1 000 000 bits.

3.75 **motion compensation**: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference frames or reference fields containing previously decoded sample values that are used to form the prediction error.

3.76 **motion estimation**: The process of estimating motion vectors during the encoding process.

3.77 **motion vector**: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture or field to the coordinates in a reference frame or reference field.

3.78 **non-intra coding**: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

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

3.80 **parameter**: A variable within the syntax of this Specification which may take one of a range of values. A variable which can take one of only two values is called a flag.

3.81 **parity (of field)**: The parity of a field can be top or bottom.

3.82 **past reference frame (field)**: A past reference frame(field) is a reference frame(field) that occurs at an earlier time than the current picture in display order.

3.83 **P-field picture**: A field structure P picture.

3.84 **P-frame picture**: A frame structure P picture.

3.87 P picture, predictive-coded picture: A picture that is coded using motion compensated prediction from previously-decoded reference fields or frames, using at most one motion vector and reference picture to predict the value of each individual region.

3.86 picture: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular arrays of 8-bit numbers representing the luma and two chroma signals. A “coded picture” is defined in 3.21. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.

3.88 prediction error: The difference between the actual value of a sample or data element and its predictor.

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

3.90 predictor: A linear combination of previously decoded sample values or data elements.

3.91 profile: A defined subset of the syntax of this specification.

3.92 progressive: The property of video frames that indicates that all of the samples of the frame represent the same instant in time.

3.94 quantiser scale: A scale factor coded in the bitstream and used by the decoding process for scaling.

3.95 random access: The process of beginning to read and decode the coded bitstream at an arbitrary point.

3.96 raster scan order: A mapping of a regular rectangular two-dimensional pattern to a one-dimensional scanning order such that the first entries in the scan order are from the first row of the pattern scanned from left to right, followed similarly by the second, third, etc. rows of the pattern each scanned from left to right.

3.97 reconstructed frame: A reconstructed frame consists of three rectangular arrays of 8-bit numbers representing the luma and two chroma signals. A reconstructed frame is obtained by decoding a coded frame.

3.98 reconstructed picture: A reconstructed picture is obtained by decoding a coded picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a reconstructed frame (when decoding a field picture). If the coded picture is a field picture, then the reconstructed picture is the top field or the bottom field of the reconstructed frame.

3.99 reference field: A reference field is one field of a reconstructed frame. Reference fields are used for forward and backward prediction when P-pictures and B-pictures are decoded. Note that when field P-pictures are decoded, prediction of the second field P picture of a coded frame uses the first reconstructed field of the same coded frame as a reference field.

3.100 reference frame: A reference frame is a non-disposable reconstructed frame.

3.101 reordering delay: A delay in the decoding process that is caused by frame reordering in order to display pictures in a different order than the order specified for the decoding process.

3.102 reserved: The term “reserved” when used in the clauses defining the coded bitstream indicates that some values of a particular syntax element may be used in extensions of this Specification by ITU-T | ISO/IEC, and that these values shall not be used unless so specified.

3.103 run: The number of zero coefficients preceding a non-zero coefficient, in the scan order. The absolute value of the non-zero coefficient is called “level”.

3.104 sample aspect ratio: (abbreviated to SAR). This specifies the distance between samples. It is defined (for the purposes of this Specification) as the vertical displacement of the lines of luma samples in a frame divided by the horizontal displacement of the luma samples. Thus its units are (metres per line) ÷ (metres per sample).

3.105 saturation: Limiting a value that exceeds a defined range by setting its value to the maximum or minimum of the range as appropriate.

3.105 short SCP: A start code prefix which uniquely indicates the start of a slice layer start code. It is one byte shorter than a long SCP.

3.107 skipped macroblock: A macroblock for which no data is encoded other than an indication that the macroblock is to be decoded as "skipped".

3.108 slice: An integer number of macroblocks belonging to the same slice group. Within the slice group, the macroblocks are ordered in raster scan order.

3.108 slice group: one or more uncoded macroblock, not necessarily in raster scan order, that share the same slice group_id in the mb_allocation_map.

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

3.110 spatial prediction: A prediction derived from the content of the current decoded frame.

3.111 start code prefix (SCP): One of a set of unique codes embedded in the coded bitstream that are used for identifying the beginning of a start code. Emulation of start code prefixes is prevented with a defined procedure.

3.113 temporal prediction: prediction derived from pictures other than the current decoded picture.

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

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

3.118 variable length coding (VLC): A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

3.119 video buffering verifier (VBV): A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

3.121 XYZ profile decoder: decoder able to decode bitstreams conforming to the specifications of the XYZ profile (with XYZ being any of the defined Profile names).

3.122 zig-zag scanning order: A specific sequential ordering of transform coefficients from (approximately) the lowest spatial frequency to the highest.

3.123 Symbol: Syntax element, or part thereof, to be coded. If a symbol is non-binary, it is converted into a sequence of binary decisions called *Bins*.

3.124 Context Modelling:The choice and definition of prior transmitted symbols that are to be used in the conditional coding of a symbol.

3.125 Context Variable: Defined for each symbol by an equation containing the recently transmitted symbols defined in the Context Modelling above.

3.126 Context: The numerical value of the Context Variable when coding a particular symbol at a particular point, block, macroblock, etc. in the picture.

3.127 Probability Model: The set of probability distributions to be used by the arithmetic coding engine when coding a symbol. The *Context* determines which probability distribution is to be used when coding 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*. For binary data, each probability distribution contains only two numbers, e.g. *p* and 1-*p*.

4 Abbreviations

4.1 **LSB**: Least Significant Bit

4.2 **MSB**: Most Significant Bit

4.3 **NAL**: Network Abstraction Layer

4.4 **SCP**: Start Code Prefix

4.5 **MB**: Macroblock

4.6 **VCL**: Video Coding Layer

5 Conventions

The following mathematical and logical operators are defined as follows

$a >> b$: Arithmetic right shift of a two's complement integer representation of *a* by *b* binary digits

$a << b$: Arithmetic left shift of a two's complement integer representation of *a* by *b* binary digits

If *b* is negative, $a << b$ is interpreted as $a >> (–b)$

a % *b*: Remainder of *a* divided by *b*, defined only for *a* and *b* both positive integers

a && *b*: Boolean logical "and" of *a* and *b*

a || *b*: Boolean logical "or" of *a* and *b*

The following mathematical functions are defined as follows

$$\text{clip3}(a, b, c) = a \text{ if } c<a,\ b \text{ if } c>b, \text{ otherwise } c \tag{5-1}$$

$$\text{clip1}(c) = \text{clip3}(0, 255, c) \tag{5-2}$$

6 Source Coder

6.1 Picture Formats

The image width and height of the decoded luma memory arrays are multiples of 16. Decoder output picture sizes that are not a multiple of 16 in width or height can be specified using a cropping rectangle. At the moment only colour sequences using 4:2:0 chroma sub-sampling are supported.

This specification describes coding of video that contains either progressive or interlaced frames, which may be mixed together in the same sequence. The vertical and horizontal locations of luma and chroma samples in progressive frames are shown in Figure 6-1.

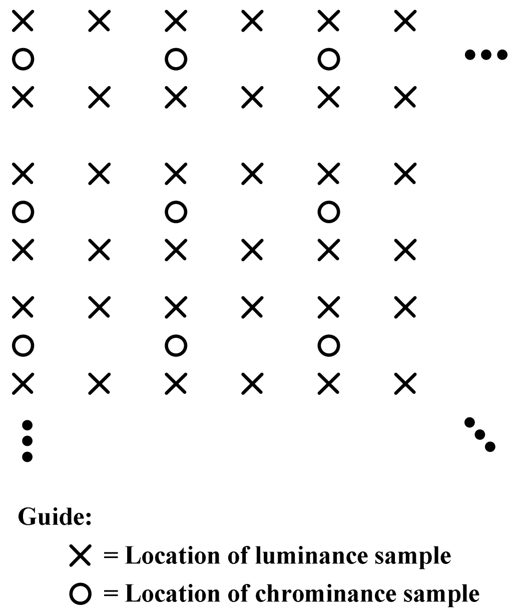

Guide:
= Location of luminance sample
= Location of chrominance sample

Figure 6-1 – Vertical and horizontal locations of 4:2:0 luma and chroma samples in progressive scan pictures

A frame of video contains two fields, the top field and the bottom field, which are interleaved. The first (i.e., top), third, fifth, etc. lines of a frame are the top field lines. The second, fourth, sixth, etc. lines of a frame are the bottom field lines. A top field picture consists of only the top field lines of a frame. A bottom field picture consists of only the bottom field lines of a frame.

The two fields of an interlaced frame are separated in time by a field period (which is half the time of a frame period). They may be coded separately as two field pictures or together as a frame picture. 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) so that other field pictures may reference the sub-fields of the frame.

The vertical and temporal sampling positions of samples in interlaced frames are shown in Figure 6-2. The vertical sampling positions 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 position relative to the full-frame sampling grid. The vertical sampling positions 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 position relative to the full-frame sampling grid. The horizontal sampling positions of the chroma samples are specified as unaffected by the application of interlaced field coding.

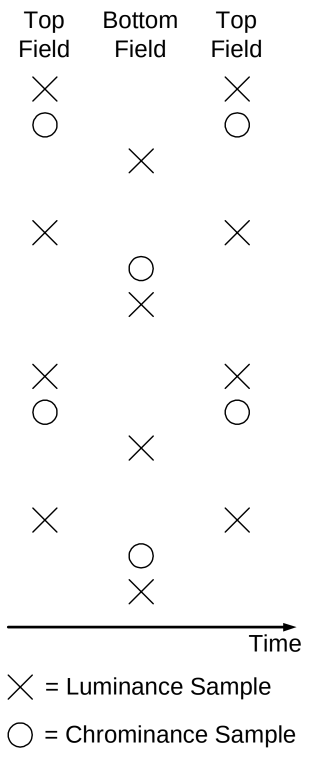


Figure 6-2 – Vertical and temporal sampling positions of samples in 4:2:0 interlaced frames

6.2 Subdivision of a picture into slices and macroblocks

Pictures (both frame and field) are divided into macroblocks of 16x16 luma samples each, with two associated 8x8 chroma samples. For instance, a QCIF picture is divided into 99 macroblocks as indicated in Figure 6-3.

Each Macroblock in a picture belongs to exactly one slice. The minimum number of slices for a picture is 1, and the maximum number of slices of a picture is the number of macroblocks in the picture (for example 99 in QCIF picture).

When Data Partitioning is not used, coded slices start with a slice header and are followed by the entropy coded symbols of the macroblock data for all the macroblocks of the slice. When Data Partitioning is used, the macroblock data of a Slice is partitioned in up to three partitions, containing header information, intra cbp and coeffiecients, and inter CPB and coefficients, respectively.

The order of the macroblocks in the bitstream depends on the Macroblock Allocation Map and is not necessarily raster scan order.

6.2.1 The Macroblock Allocation Map (mb_allocation_map)

The mb_allocation_map contains the slice_group_id of every macroblock in a picture. It is part of the Parameter Set that is referenced by the parameter_set_id in the slice header. All uncoded macroblocks that share the same slice_group_id are called a slice group. Hence, an mb_allocation_map allocates every macroblock to exactly one slice group. Each slice group is represented by one or more slices in the bitstream. Within each slice group, macroblocks are ordered in raster scan order. Figure 6-3 exemplifies macroblock assignments to a slice.

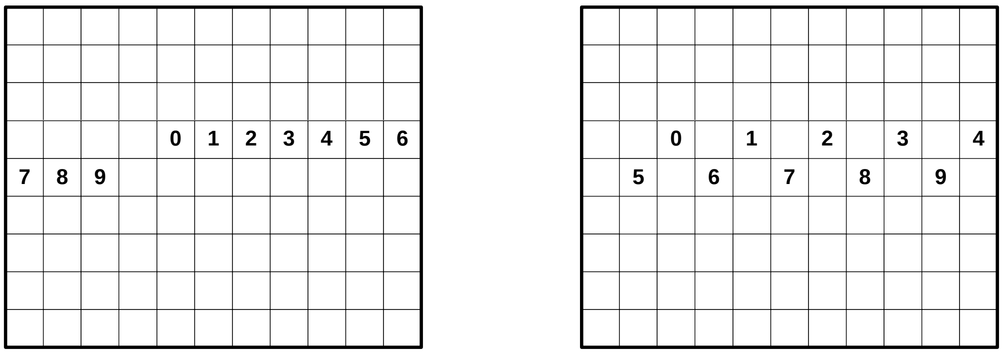


Figure 6-3 – Two examples for macroblock assignment to a slice and their numbering

6.3 Order of the bitstream within a macroblock

Figures 6-4 and 6-5 indicate how a macroblock or 8x8 sub-block is divided and the order of the different syntax elements resulting from coding a macroblock.

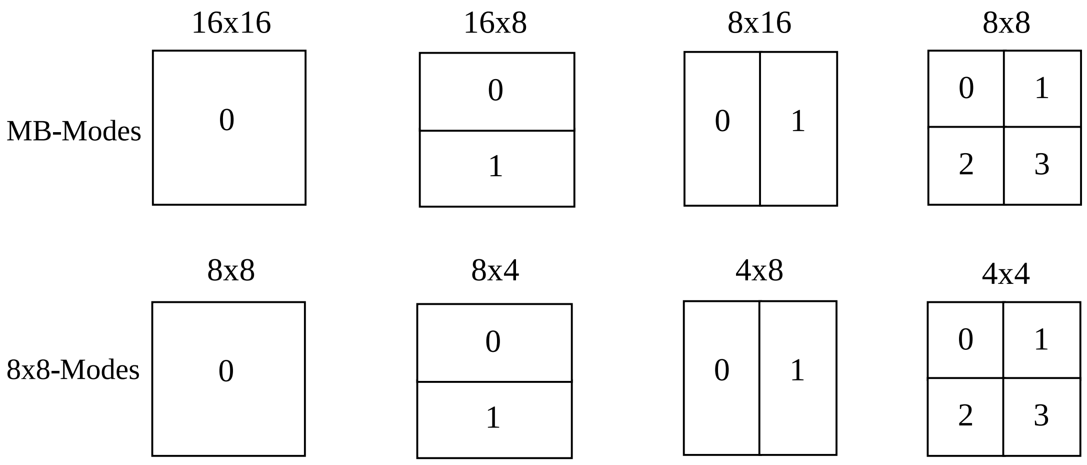


Figure 6-4 – Numbering of the vectors for the different blocks depending on the inter mode. For each block the horizontal component comes first followed by the vertical component (raster scan order)

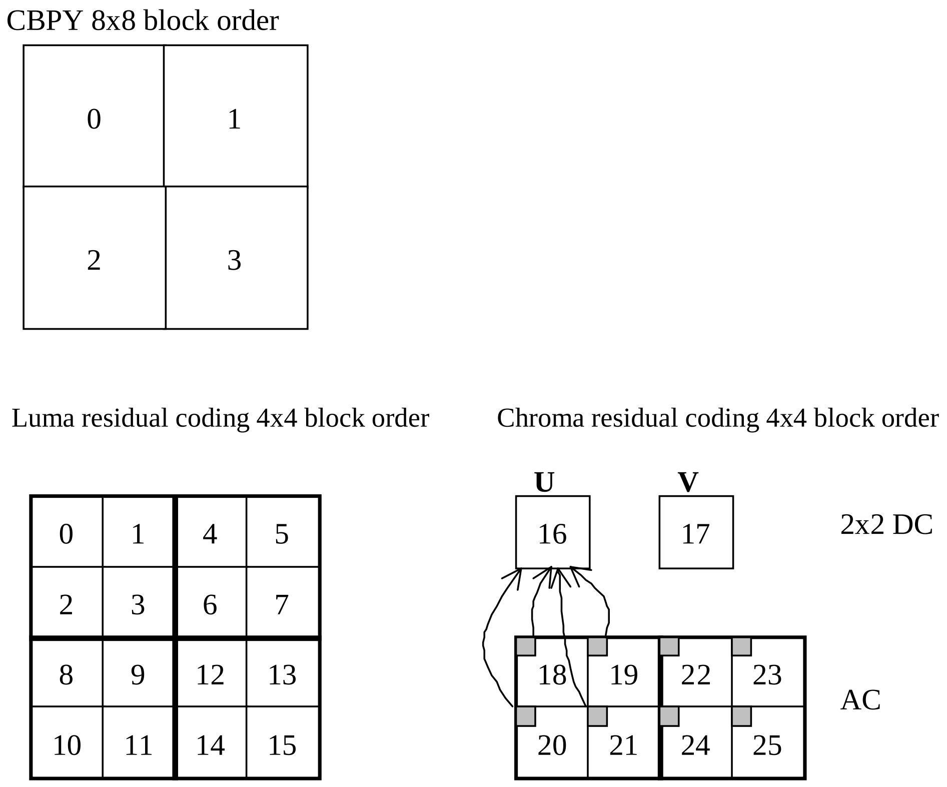


Figure 6-5 – Ordering of blocks for cbpY and residual coding of 4x4 blocks

7 Syntax

7.1 Method of describing the syntax in tabular form

The syntax is described in a similar manner as the syntax in ITU-T Recommendation H.262 | ISO/IEC 13818-2. It closely follows the C-language syntactic constructs. Data items in the bitstream are represented in **bold** type. Each data item is described by its name, its length in bits and an indication of type and order of transmission. A decoder behaves according to the value of the decoded data element in the bitstream and on the values of previously decoded data elements.

The syntax tables describe a superset of the syntax of all correct and error-free input bitstreams. An actual decoder must implement correct means for identifying the beginning of the bitstream for proper decoding and to identify and handle errors in the bitstream. 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 usually represented by '1', but any other value different than zero is understood as TRUE.

The following table lists examples of pseudo code used to describe the syntax. When data_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.

	Category	**Descriptor**
/* A statement can be a data element with an associated syntax category and descriptor or can be an expression used to specify conditions for the type and quantity of data elements, as in the following two examples */		
data_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 to better understand the behaviour of a compliant decoder. 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.

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

next_bits()

- Provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next token in the bitstream.

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).

coding_type()

- Returns the coding type of slice or picture.

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

- **b(8):** byte having any value (8 bits).
- **e(v)**: 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 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.

multiple_fwd_reference_pictures()

- Returns TRUE if picture has possibility of prediction from more than one previous decoded picture. Otherwise it returns FALSE.

two_picture_fwd_prediction()

- Returns TRUE if picture has possibility of prediction from less than or equal to two previous decoded pictures. Otherwise it returns FALSE.

multiple_bwd_reference_pictures()

- Returns TRUE if picture has possibility of prediction from more than one subsequent decoded picture. Otherwise it returns FALSE.

two_picture_bwd_prediction()

- Returns TRUE if picture has possibility of prediction from less than or equal to two subsequent decoded pictures. Otherwise it returns FALSE.

When CABAC entropy coding is applied, these descriptors do not apply to some categories of syntax elements.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

nal_unit (EBSPsize) {	**Category**	**Descriptor**
error_flag		u(1)
nal_unit_type		u(5)
picture_header_flag		u(1)
disposable_flag		u(1)
for(i=0; i<EBSPsize; i++)		
ebsp[i]		b(8)
}		

7.3.2 RBSP extraction syntax

rbsp_extraction(NumBytesInEBSP) { /* external framing */	**Category**	**Descriptor**
NumBytesInRBSP = 0		
for(i=0; i<NumBytesInEBSP; i++) {		
if(next_bits() == 0x0003) {		
rbsp[i++]		b(8)
emulation_prevention_byte /* = = 0x03 */		f(8)
} else		
rbsp[i]		b(8)
NumBytesInRBSP++		
}		
}		

7.3.3 Parameter set RBSP syntax

parameter_set_rbsp( ) {	**Category**	**Descriptor**
parameter_set_id	0	e(v)
log2_max_frame_num_minus_4	0	e(v)
num_of_reference_pictures	0	e(v)
required_frame_num_update_behaviour	0	u(1)
frame_width_in_MBs_minus1	0	e(v)
frame_height_in_MBs_minus1	0	e(v)
frame_cropping_rect_left_offset	0	e(v)
frame_cropping_rect_right_offset	0	e(v)
frame_cropping_rect_top_offset	0	e(v)
frame_cropping_rect_bottom_offset	0	e(v)
aspect_ratio_info	0	b(8)
if (aspect_ratio_info == “extended_PAR”) {		
par_width	0	u(8)
par_height	0	u(8)
}		
video_signal_type	0	u(1)
if (video_signal_type) {		
video_format	0	u(3)
video_range	0	u(1)
colour_description	0	u(1)
if (colour_description) {		
colour_primaries	0	b(8)
transfer_characteristics	0	b(8)
matrix_coefficients	0	b(8)
}		
}		
entropy_coding_mode	0	e(v)
motion_resolution	0	e(v)
constrained_intra_prediction_flag	0	u(1)
num_units_in_tick	0	u(32)
time_scale	0	u(32)
num_slice_groups	0	u(3)
if (num_slice_groups > 0) { /* use of Flexible MB Ordering */		
mb_allocation_map_type	0	e(v)
if (mb_allocation_map_type = = 0) {		
for (loop_count = 0; loop_count <= max_slice_group_id; loop_count++)		
run_length	0	e(v)
}		
else if (mb_allocation_map_type = = 2) {		
for (loop_count = 0; loop_count < num_mbs_in_picture; loop_count++)		
slice_group_id	0	u(3)
}		
else		
reserved		variable
}		
rbsp_trailing_bits()	All	

}		

7.3.4 Supplemental enhancement information syntax

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

7.3.5 Supplemental enhancement information message syntax

sei_message( ) {	**Category**	**Descriptor**
payloadType = 0		
while (next_bits() == 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() == 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	b(8)
}		

7.3.6 Picture layer RBSP syntax

picture_layer_rbsp( ) {	**Category**	**Descriptor**
picture_structure	3	e(v)
frame_num	3	u(v)
rps_layer()		
if(coding_type() = = B) {		
direct_mv_scale_fwd	3	e(v)
direct_mv_scale_bwd	3	e(v)
direct_mv_scale_divisor	3	e(v)
explicit_B_prediction_block_weight_indication		e(v)
if (explicit_B_prediction_block_weight_indication > 1)		
adaptive_B_prediction_coeff_table()		
}		
rbsp_trailing_bits()		
}		

7.3.7 Adaptive bi-prediction coefficient table syntax

adaptive_B_prediction_coeff_table () {	Category	Mnemonic
number_of_abp_coeff_minus1		e(v)
for (i=0; i<= number_of_luma_abp_coeff_minus1; i++) {		
luma_first_weight_factor_magnitude[i]		e(v)
luma_first_weight_factor_sign[i]		u(1)
luma_second_weight_factor_magnitude[i]		e(v)
luma_second_weight_factor_sign[i]		u(1)
luma_constant_factor_magniture[i]		e(v)
luma_constant_factor_sign[i]		u(1)
luma_logarithmic_weight_denominator[i]		e(v)
for (iCbCr=0; iCbCr<2; iCbCr++) {		
chroma_first_weight_factor_magnitude[iCbCr][i]		e(v)
chroma_first_weight_factor_sign[iCbCr][i]		u(1)
chroma_second_weight_factor_magnitude[iCbCr][i]		e(v)
chroma_second_weight_factor_sign[iCbCr][i]		u(1)
chroma_constant_factor_magnitude[iCbCr][i]		e(v)
chroma_constant_factor_sign[iCbCr][i]		u(1)
chroma_logarithmic_weight_denominator[iCbCr][i]		e(v)
}		
}		
}		

7.3.8 Slice layer RBSP syntax

slice_layer_rbsp( ) {	**Category**	**Descriptor**
slice_header()	4	
slice_data()		
rbsp_trailing_bits()		
}		

7.3.9 Data Partition A RBSP syntax

dpa_layer( ) {	**Category**	**Mnemonic**
slice_header()	4	
slice_id		e(v)
slice_data()	4	
}		

7.3.10 Data Partition B RBSP syntax

dpb_layer( ) {	**Category**	**Mnemonic**
slice_id		e(v)
slice_data()	5	
}		

7.3.10 Data Partition C RBSP syntax

dpc_layer( ) {	**Category**	**Mnemonic**
slice_id		e(v)
slice_data()	6	
}		

7.3.11 RBSP end data syntax

rsbp_trailing_bits( ) {	**Category**	**Descriptor**
rbsp_stop_bit /* equal to 1 */	4	f(1)
while(!byte_aligned())		
rbsp_alignment_bit	4	f(1)
}		

7.3.12 Slice header syntax

slice_header( ) {	**Category**	**Descriptor**
parameter_set_id	4	e(v)
first_mb_in_slice	4	e(v)
if (coding_type() == P || coding_type() == B) {		
num_ref_pic_active_fwd_minus1	4	e(v)
if(coding_type() == B)		
num_ref_pic_active_bwd_minus1	4	e(v)
}		
rps_layer()		
slice_delta_qp /* relative to 26 */	4	e(v)
if(coding_type() == SP || coding_type() == SI) {		
slice_delta_qp_sp /* relative to 26 */	4	e(v)
}		
if(entropy_coding_mode = = 1)		
num_mbs_in_slice	4	e(v)
}		

7.3.13 Reference Picture Selection Layer

rps_layer( ) {	**Category**	**Descriptor**
if(coding_type != I) {		
reference_reordering_indicator_fwd	3 | 4	
if(reference_reordering_indicator_fwd) {		
do {		
remapping_of_frame_nums_indicator	3 | 4	u(1)
if(remapping of frame_nums_indicator = = 0 | | remapping_of_frame_nums_indicator = = 1)		
abs_diff_pic_numbers	3 | 4	e(v)
else if(remapping_of_frame_nums_indicator = = 2)		
long_term_pic_index	3 | 4	e(v)
} while(remapping_of_frame_nums_indicator != 3)		
}		
}		
if(coding_type = = B) {		
reference_reordering_indicator_bwd	3 | 4	
if(reference_reordering_indicator_bwd) {		
do {		
remapping_of_frame_nums_indicator	3 | 4	u(1)
if(remapping of frame_nums_indicator = = 0 | | remapping_of_frame_nums_indicator = = 1)		
abs_diff_pic_numbers	3 | 4	e(v)
else if(remapping_of_frame_nums_indicator = = 2)		
long_term_pic_index	3 | 4	e(v)
} while(remapping_of_frame_nums_indicator != 3)		
}		
}		
ref_pic_buffering_type	3 | 4	u(1)
if (ref_pic_buffering_type = = Adaptive Memory Control) {		
do {		
memory_management_control_operation	3 | 4	e(v)
if(memory_management_control_operation = = 1 | | memory_management_control_operation = = 3)		
difference_of_frame_nums	3 | 4	e(v)
else if (memory_management_control_operation = = 2 | | memory_management_control_operation = = 3)		
long_term_picture_index	3 | 4	e(v)
else if (memory_management_control_operation = = 4)		
max_long_term_index_plus1	3 | 4	e(v)
} while(memory_management_control_operation != 0 | | memory_management_control_operation != 5)		
}		
}		

7.3.14 Slice data syntax

slice_data( ) {	**Category**	**Descriptor**
do {		
if (coding_type() != I)		
mb_skip_run	4	e(v)
if (more_rbsp_data ())		
coded_macroblock_layer()		
} while (more_rbsp_data ())		
}		

7.3.15 Coded macroblock layer syntax

coded_macroblock_layer( ) {	**Category**	**Descriptor**
mb_type	4	e(v)
if (mb_type = = MB8x8refX | | mb_type = = MB8x8ref0)		
for(i=0; i<4; i++)		
prediction8x8(mb_type)		
else		
prediction16x16(mb_type)		
if (mb_type != MBintra16x16)		
cbp	4	e(v)
if (cbp || (mb_type = = MBintra16x16)) {		
delta_qp	4	e(v)
residual( )		
}		
}		

7.3.16 Prediction data for 8x8 modes

prediction8x8(mb_type) {	**Category**	**Mnemonic**
for(i=0; i<4; i++)		
region8x8_type[i]	4	e(v)
for(i=0; i<4; i++)		
if (region8x8_type[i] = = BlockIntra4x4) {		
for (i4x4=0; i4x4<4; i4x4++)		
intra_pred_mode	4	e(v)
}		
for(i=0; i<4; i++)		
if (num_ref_pic_active_fwd_minus1 > 0 && mb_type ! = MB8x8ref0 && region8x8_type[i] !=BlockIntra4x4 && region8x8_type[i] !=Direct && region8x8_prediction_type[i] !=Backward)		
ref_idx_fwd	4	e(v)
for(i=0; i<4; i++)		
if (num_ref_pic_active_bwd_minus1 > 0 && (region8x8_type[i] !=Intra &&| | region8x8_type[i] != Direct && region8x8_prediction_type[i] != Forward)		
ref_idx_bwd	4	e(v)
for(i=0; i<4; i++)		
if (region8x8_type[i] = = Bipred. && explicit_B_prediction_block_weight_indication > 1 && number_of_abp_coeff_minus_1 > 0)		
abp_coeff_idx		e(v)
for (k=0; k<4; k++)		
if (region8x8_type[k] != Intra && region8x8_type[k] != Direct && region8x8_prediction_type[k] != Backward)		
for (i=0; i<number_region8x8_mvectors[region8x8_type[k]]; i++) {		
for (j=0; j<2; j++)		
mvd_fwd[k][i][j]	4	e(v)
}		
for (k=0; k<4; k++)		
if (region8x8_type[k]!=Intra && region8x8_type[k]!=Direct && region8x8_prediction_type[k]!=Forw.)		
for (i=0; i<number_region8x8_mvectors[region8x8_type[k]]; i++) {		
for (j=0; j<2; j++)		
mvd_bwd[k][i][j]	4	e(v)
}		
}		

7.3.17 Prediction data for 16x16 modes

prediction16x16(mb_type) {	**Category**	**Mnemonic**
if (mb_type = = MBintra4x4)		
for (i4x4=0; i4x4<16; i4x4++)		
intra_pred_mode	4	ecselbf
else if (mb_type != MBintra16x16 && mb_type != MBDirect) {		
for (i=0; i<number_sub_block[mb_type]; i++) {		
if (num_ref_pic_active_fwd_minus1 > 0 && (mb_type = = MBfwd | | mb_type = = MBbidir)) {		
ref_idx_fwd	4	e(v)
}		
for (i=0; i<number_sub_block[mb_type]; i++) {		
if (num_ref_pic_active_bwd_minus1 > 0 && (mb_type = = MBbwd | | mb_type = = MBbidir))		
ref_idx_bwd	4	e(v)
}		
if (mb_type == MBbidir && explicit_B_prediction_block_weight_indication > 1 && number_of_abp_coeff_minus_1 > 0)		
abp_coeff_idx		e(v)
for(i=0; i<number_mb_mvectors[mb_type]; i++) {		
if (mb_type = = MBfwd | | mb_type = = MBbidir)		
for(j=0; j<2; j++)		
mvd_fwd[j]	4	ecelbf
}		
for(i=0; i<number_mb_mvectors[mb_type]; i++) {		
if (mb_type = = MBbwd | | mb_type = = MBbidir)		
for(j=0; j<2; j++)		
mvd_bwd[j]	4	ecelbf
}		
}		
}		

7.3.18 Residual data syntax

residual(mb_type) {	**Category**	**Mnemonic**
if (mb_type = = MBintra16x16)		
residual_4x4block_cvlc(intra16x16DC, 16)		
for (i8x8=0; i8x8<4; i8x8++) /* each luma 8x8 region */		
for (i4x4=0; i4x4<4; i4x4++) /* each 4x4 sub-block of region */		
if (cbp & (1<< i8x8)		
if (mb_type == MBintra16x16)		
residual_4x4block_cvlc(intra16x16AC, 16)		
else		
residual_4x4block_cvlc(luma, 16)		
if (cbp & 0x30) { /* chroma DC residual coded */		
for (iCbCr=0; iCbCr<2; iCbCr++) {		
residual_4x4block_cvlc(luma, 16)		
if (cbp & 0x20) { /* chroma AC residual coded */		
for (iCbCr=0; iCbCr<2; iCbCr++)		
for (i4x4=0; i4x4<4; i4x4++)		
residual_4x4block_cvlc(luma, 16)		
}		
}		

residual_4x4block_cvlc(block_type, max_numcoeff) {		
select VLC for numcoeff_trailingones		
numcoeff_trailingones	5 | 6	ecselbf
if (trailingones)		
for(i=trailingones-1; i>=0;i--)		
trailingones_sign[i]	5 | 6	uimsbf(1)
if (numcoeff) {		
select VLC for level		
for(i=numcoeff-1; i>=0;i--) {		
luma_level[i]	5 | 6	ecselbf
Update level VLC		
}		
if (numcoeff_trailingones < max_numcoeff) {		
select VLC for totalzeros		
totalzeros	5 | 6	ecselbf
i = numcoeff_trailingones - 1		
zerosleft = totalzeros		
if (zerosleft > 0) {		
Do {		
select VLC for runbefore		
runbefore[i]	5 | 6	ecselbf
zerosleft -= runbefore[i]		
i--		
while (zerosleft != 0 && i != 0)		
runbefore[0] = zerosleft		
}		
}		
}		
}		

8 Semantics

8.1 Organization of syntax elements (NAL concept)

The Video Coding Layer (VCL) is defined to efficiently represent the content of the video data, and the Network Abstraction Layer (NAL) is defined to format that data and provide header information in a manner appropriate for conveyance by the transport layers or storage media. The data is organized into NAL units, each of which contains an integer number of bytes. A NAL unit defines 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 for the fact that each NAL unit can be preceded by a start code in a bitstream-oriented transport layer.

EBSPsize defines the size of the EBSP in bytes. This element of the NAL interface does not necessarily appear as a syntax element, but rather denotes the quantity of EBSP data to be carried by the system for the NAL unit.

error_flag signals whether a bit error exists in the NAL unit (including the nal_unit_type, the error_flag itself, the disposable_flag, the picture_header flag, and the EBSP). The handling of an error-marked unit is non-normative and is not defined by this specification.

nal_unit_type indicates the type of element contained in the NAL unit according to the types specified in Table 15.1.

picture_header_flag indicates that the EBSP contains a picture header that precedes any other data in the EBSP. If the picture_header_flag is 1, the nal_unit_type shall not indicate a picture header.

disposable_flag signals whether the content of the EBSP belongs to a picture that is not stored in the multi-picture buffer. disposable_flag shall not be signalled for supplemental enhancement information and parameter set information. The value of disposable_flag shall be the same for all the picture header, slice, and data partition EBSPs of a particular picture.

ebsp[i**]** (Encapsulated Byte Sequence Payload) contains the NAL unit data in the encapsulated byte sequence payload (EBSP) format. Any sequence of bits can be formatted into a sequence of bytes in a manner defined as a Raw Byte Sequence Payload (RBSP), and any RBSP can be encapsulated in a manner that prevents emulation of byte stream start code prefixes in a manner defined as an encapsulated byte sequence payload (EBSP).

Within the ebsp data, an ebsp[i] byte having the value zero (0x00) shall not be immediately followed by a byte ebsp[i+1] having any of the following three values:

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

If an ebsp[i] byte having the value zero (0x00) is immediately followed by an ebsp[i+1] byte having the value 0x03, these bytes shall not be immediately followed within the EBSP by a byte ebsp[i+2] having any value other than the following four values:

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

When an ebsp[i] byte having the value zero (0x00) is immediately followed within the EBSP by the special ebsp[i+1] byte value of three (0x03), the ebsp[i+1] value 0x03 shall be removed from the bitstream by the decoder and discarded. This special value is used to allow an RBSP payload to contain any sequence of byte values without allowing the EBSP to contain the three special two-byte patterns described above.

Table 8-1. NAL Unit Type (NUT) Codes

Code	NAL Unit Type (nal_unit_type)	Category	Priority of nal_unit_type (PNUT)	coding_type
0x01	Intra Picture Header	3	2	na
0x02	Intra IDER Picture Header	3	1	na
0x03	Intra Slice of Intra Picture	3, 4, 5	3	I
0x04	Intra Slice of Intra IDER Picture	3, 4, 5	3	I
0x05	Intra Slice of Mixed Picture	3, 4, 5	3	I
0x06	Inter Picture Header – All Inter	3, 4, 5, 6	2	P
0x07	Inter Picture Header – Mixed	3, 4, 5, 6	2	P
0x08	Inter Slice of Inter Picture	3, 4, 5, 6	3	P
0x09	Inter Slice DPA of All-Inter Picture	3, 4	3	P
0x0A	Inter Slice DPA of Mixed Picture	3, 4	3	P
0x0B	B Picture Header – All B	3	2	B
0x0C	B Picture Header – Mixed	3	2	B
0x0D	B Slice of All-B Picture	3, 4, 5, 6	3	B
0x0E	B Slice of Mixed Picture	3, 4, 5, 6	3	B
0x0F	B DPA Slice of All-B Picture	3, 4	3	B
0x10	B DPA Slice of Mixed Picture	3, 4	3	B
0x11	SI Picture Header	3	2	SI
0x12	SI IDER Picture Header	3	1	SI
0x13	SI Slice of All-SI Picture	3, 4, 5	3	SI
0x14	SI Slice of All-SI IDER Picture	3, 4, 5	3	SI
0x15	SI Slice of Mixed Picture	3, 4, 5	3	SI
0x16	SP Picture Header – All SP	3	2	SP
0x17	SP Picture Header – Mixed	3	2	SP
0x18	SP Slice of All-SP Picture	3, 4, 5, 6	3	SP
0x19	SP Slice of Mixed Picture	3, 4, 5, 6	3	SP
0x1A	SP Slice DPA of All-SP Picture	3, 4	3	SP
0x1B	SP Slice DPA of Mixed Picture	3, 4	3	SP
0x1C	DPB	5	4	na
0x1D	DPC	6	4	na
0x1E	Supplemental Enhancement Information	7	5	na
0x1F	Parameter Set Information	0	0	na

An instantaneous decoder refresh picture (IDER picture) implies that all pictures in the multi-picture buffer are marked as “unused” except the current picture. Moreover, the maximum long-term index is reset to zero. An IDER picture contains only I or SI slices, and IDER slice type shall be used for all slices of an IDER picture. The picture starts an independent group of pictures (independent GOP) that lasts until the next IDER picture.

8.2 Raw and encapsulated byte sequence payloads

8.2.1 Raw encapsulated sequence payload (RBSP)

rbsp[i**]**: a raw byte sequence payload (RBSP) is defined as an ordered sequence of bytes that contains a string of data bits (SODB). A SODB is defined as an ordered sequence 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). 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').

Note that the last byte of a RBSP can never have the 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 section using an "_rbsp" suffix.

8.2.2 Encapsulated byte sequence payload (EBSP)

ebsp[i**]** is an encapsulated byte sequence payload (EBSP) is an ordered sequence of bytes that contains an RBSP and may also contain some emulation_prevention_byte syntax elements in order to prevent the presence of certain specific sequences of data bytes within the EBSP.

emulation_prevention_byte is a byte equal to 0x03.

8.2.3 Parameter set RBSP

The parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. A parameter set is a collection of all the parameter values needed to correctly decode the VCL data. Each slice references the parameter set containing the proper set of values needed to decode that slice.

It is recommended to convey parameter sets out-of-band using a reliable transport mechanism. However, in applications where the bitstream has to be self-contained, in-band parameter set information units may be used. In error-prone transmission environments, in-band parameter set information units should be protected in a way that assures their successful reception before they are referenced. Synchronization between in-band and out-of-band transmission of the parameter set information is outside of the scope of this specification.

All the slices of an independent GOP shall refer to the same parameter set. A received parameter set information unit shall come into effect just before the next IDER picture is decoded.

parameter_set_id identifies the parameter set to be addressed.

log2_max_frame_num_minus_4 specifies the MAX_PN constant used in frame number related arithmetic. MAX_PN = 2 log2_max_frame_num_minus_4 + 4 – 1.

number_of_reference_pictures defines the total number of short- and long-term picture buffers in the multi-picture buffer.

If **required_frame_num_update_behaviour** is 1, a specific decoder behaviour in case of missing frame numbers is mandated (see section 9.1.1.2).

frame_width_in_MBs_minus1 and **frame_height_in_MBs_minus1** define the size of the luma frame array internal to the decoder in units of macroblocks. The frame width and height in units of macroblocks is computed by adding 1 to the decoded values of each of these parameters.

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 defined as the area within the rectangle containing luma samples with horizontal coordinates from cropping_rect_left to 16*(picture_width_in_MBs_minus1+1)-(cropping_rect_right+1) and with vertical coordinates from cropping_rect_top to 16*(picture_height_in_MBs_minus1+1)-(cropping_rect_bottom+1), inclusive.

video_signal_type: A flag which if set to '1' indicates the presence of video_signal_type information. If set to '0', the video signal type is undefined or specified externally.

video_format: This is a three bit integer indicating the representation of the pictures before being coded in accordance with this Specification. Its meaning is defined in Table 8-2. If the video_signal_type() is not present in the bitstream then the video format may be assumed to be "Unspecified video format".

Table 8-1 – 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: This one-bit flag indicates the black level and range of the luminance and chrominance signals.

colour_description: A 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 defined in Table 8-3.

Table 8-2 – 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

In the case that video_signal_type() is not present in the bitstream 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 defined in Table 8-4.

Table 8-3 – 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

In the case that video_signal_type() is not present in the bitstream 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, and is defined in Table 8-5.

In this table:

E'_Y is analogue with values between 0 and 1

E'_{PB} and E'_{PR} are analogue between the values -0,5 and 0,5

E'_R, E'_G and E'_B are analogue with values between 0 and 1

White is defined as E'_y=1, E'_{PB}=0, E'_{PR}=0; E'_R =E'_G =E'_B=1.

Y, Cb and Cr are related to E'_Y, E'_{PB} and E'_{PR} by the following formulae:

if video_range=0:

Y = round(219 * E'_Y + 16)

Cb = round(224 * E'_{PB} + 128)

Cr = round(224 * E'_{PR} + 128)

if video_range=1:

Y = round(255 * E'_Y)

Cb = round(255 * E'_{PB}) + 128)

Cr = round(255 * E'_{PR} + 128)

Table 8-4 – Matrix Coefficients

Value	**Matrix**
0	reserved
1	ITU-R Recommendation BT.709 E'_Y = 0,7152 E'_G + 0,0722 E'_B + 0,2126 E'_R E'_{PB} = -0,386 E'_G + 0,500 E'_B -0,115 E'_R E'_{PR} = -0,454 E'_G - 0,046 E'_B + 0,500 E'_R
2	Unspecified Video Image characteristics are unknown.
3	reserved

4	FCC $E'_Y = 0{,}59\ E'_G + 0{,}11\ E'_B + 0{,}30\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}421\ E'_G - 0{,}079\ E'_B + 0{,}500\ E'_R$
5	ITU-R Recommendation BT.470-2 System B, G $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
6	SMPTE 170M $E'_Y = 0{,}587\ E'_G + 0{,}114\ E'_B + 0{,}299\ E'_R$ $E'_{PB} = -0{,}331\ E'_G + 0{,}500\ E'_B - 0{,}169\ E'_R$ $E'_{PR} = -0{,}419\ E'_G - 0{,}081\ E'_B + 0{,}500\ E'_R$
7	SMPTE 240M (1987) $E'_Y = 0{,}701\ E'_G + 0{,}087\ E'_B + 0{,}212\ E'_R$ $E'_{PB} = -0{,}384\ E'_G + 0{,}500\ E'_B - 0{,}116\ E'_R$ $E'_{PR} = -0{,}445\ E'_G - 0{,}055\ E'_B + 0{,}500\ E'_R$
8-255	reserved

In the case that video_signal_type() is not present in the bitstream or colour_description is zero the matrix coefficients are assumed to be undefined or specified externally.

In the case that video_signal_type() is not present in the bitstream, video_range is assumed to have the value 0 (a nominal range of Y from 16 to 235).

aspect_ratio_info: This is an eight-bit integer which defines the value of pixel aspect ratio. Table 8-6 shows the meaning of the code. If aspect_ratio_info indicates extended PAR, pixel_aspect_ratio is represented by par_width and par_height. The par_width and par_height shall be relatively prime. If aspect_ratio_info is zero or if either par_width or par_height are zero, the pixel aspect ratio shall be considered unspecified or specified externally.

Table 8-5 – Meaning of pixel aspect ratio

aspect_ratio_info	pixel aspect ratios
'0000 0000'	undefined
'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'	60:33 (2/3-wide 16:9 for 525)
'0000 1110'	48:33 (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'-'1111 1110'	reserved
'1111 1111'	extended PAR

par_width: This is an 8-bit unsigned integer which indicates the horizontal size of pixel aspect ratio. A zero value is forbidden.

par_height: This is an 8-bit unsigned integer which indicates the vertical size of pixel aspect ratio. A zero value is forbidden.

entropy_coding_mode equal to zero stands for the non-arithmetic variable-length coding, whereas value one stands for the arithmetic variable-length coding (see clause 10).

motion_resolution equal to zero stands for ¼-sample motion resolution, and equal to one stands for 1/8-sample motion resolution.

constrained_intra_prediction_flag equal to zero stands for normal intra prediction, whereas one stands for the constrained intra prediction. In the constrained intra prediction mode, no intra prediction is done from inter macroblocks.

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 defined externally or in Annex C. 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 should be 30 000 and num_units_in_tick should be 1001.

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.

num_slice_groups: The number of slice groups is equal to num_slice_groups + 1. If num_slice_groups is zero, all slices of the picture belong to the same slice group, and no flexible macroblock ordering is applied. If num_slice_groups is greater than zero, flexible macroblock ordering is in use. Values greater than 7 are reserved for future use.

mb_allocation_map_type: The macroblock allocation map type is present only if num_slice_groups is greater than 0. This parameter indicates how the macroblock allocation map is coded. Values ranging from 0 to 2 are valid.

run_length codeword follows for each slice group. It signals the number of consecutive macroblocks that are assigned to the slice group in raster scan order

mb_allocation_map_type 0 is used to interleave slices. 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. For example, to signal interleaved slices in a QCIF picture, the number of slice groups is two and run_length of 11 for both slice groups.

mb_allocation_map_type 1 is used to define a dispersed macroblock allocation. 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 the equation

$$S = \begin{cases} [(x\%n)+1]\%p & for\ (x/n)\ even \\ [(x\%n)+1+p/2]\%p & for\ (x/n)\ odd \end{cases}$$

where “%” and “/” represent the modulo operation and division with truncation respectively.

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

slice_group_id identifies a slice group of a macroblock ranging from 0 to 7.

8.2.4 Supplemental Enhancement Information RBSP

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

An SEI element contains one or more SEI messages. Each SEI message consists of a 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. Valid SEI payload types are listed in Annex C.

The SEI payload may have a SEI payload header. For example, a payload header may indicate to which picture the particular data belongs. The payload header shall be defined for each payload type separately. Definitions of SEI payloads are specified in Annex C.

The transmission of SEI units is synchronous relative to other NAL units. An SEI message may concern a slice, a part of a picture, a picture, any group of pictures, or a sequence in the past, currently decoded, or in the future. An SEI message may also concern one or more NAL units previous or next in transmission order.

byte_ff is a byte equal to xff 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.

8.2.5 Picture layer RBSP

picture_struct identifies the picture structure.

Code_number =0:	Progressive frame picture.
Code_number =1:	Top field picture.
Code_number =2:	Bottom field picture.
Code_number =3:	Interlaced frame picture, whose top field precedes its bottom field in time.
Code_number =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.

frame_num labels the frame. frame_num shall be incremented by 1 for each coded picture in coding order, in modulo MAX_frame_num operation, relative to the frame_num of the previous stored frame in coding order. For non-stored frames, frame_num shall be incremented from the value in the most temporally recent stored frame which precedes the non-stored frame in coding order. The frame_num serves as a unique ID for each frame stored in the multi-frame 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. The encoder shall ensure that no frame_num equals to each other among the short-term frames in the multi-frame 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 multi-frame buffer should treat this condition as an error.

direct_mv_scale_fwd is a scaling factor for the direct motion vector computation pointing into the forward reference set.

direct_mv_scale_bwd is a scaling factor for the direct motion vector computation pointing into the backward reference set.

direct_mv_scale_divisor scales direct_mv_scale_fwd and direct_mv_scale_bwd.

Reference picture selection layer see subclause 8.4.

8.2.6 Adaptive Bi-prediction Picture Coefficient

8.2.6.1 Explicit_Bi_Prediction_Block_Weight_Indication

If explicit_B_prediction_block_weight_indication is 0, the prediction block of a B-block shall be generated by averaging the pixel values of the prediction blocks. If explicit_B_prediction_indication is 1, the implicit B prediction block weighting is in use. Otherwise, the explicit B prediction block weighting is in use.

8.2.6.2 Number of Adaptive Bi-prediction Coefficients (number_of_abp_coeff_minus_1)

The number of adaptive bi-prediction coefficients is equal to (number_of_abp_coeff_minus_1 represents + 1).

8.2.6.3 Adaptive Bi-prediction Coefficients

luma_first_weight_factor_magnitude and luma_first_weight_factor_sign are magnitude and sign of the first weighting factor to generate the luminance ABP prediction signal, respectively. luma_second_weight_factor_magnitude and luma_second_weight_factor_sign are magnitude and sign of the second weighting factor to generate the luminance ABP prediction signal, respectively. luma_constant_factor_magnitude and luma_constant_factor_sign are magnitude and sign of the constant factor to generate the luminance ABP prediction signal, respectively. luma_logarithmic_weight_denominator is the binary logarithm of the denominator of the weighting factor.

The same applies for chrominance ABP coefficients using chroma_first_weight_factor_magnitude, chroma_first_weight_factor_sign, chroma_second_weight_factor_magnitude, chroma_second_weight_factor_sign, chroma_constant_factor_magniture, chroma_constant_factor_sign, and chroma_logarithmic_weight_denominator.

These are summarized in Table 8-x.

Table 8-6 – ABP coefficients

	factor	luminance/chrominance	magnitude/sign
luma_first_weight_factor_magnitude	first weight factor	luminance	magnitude
luma_first_weight_factor_sign	first weight factor	luminance	sign
luma_second_weight_factor_magnitude	second weight factor	luminance	magnitude
luma_second_weight_factor_sign	second weight factor	luminance	sign
luma_constant_factor_magnitude	constant factor	luminance	magnitude
luma_ constant_factor_factor_sign	constant factor	luminance	sign
luma_logarithmic_weight_denominator	precision(bits)	luminance	-
chroma_first_weight_factor_magnitude	first weight factor	chrominance	magnitude
chroma_first_weight_factor_sign	first weight factor	chrominance	sign
chroma_second_weight_factor_magnitude	second weight factor	chrominance	magnitude
chroma_second_weight_factor_sign	second weight factor	chrominance	sign
chroma_constant_factor_magnitude	constant factor	chrominance	magnitude
chroma_constant_factor_sign	constant factor	chrominance	sign
chroma_logarithmic_weight_denominator	precision(bits)	chrominance	-

8.2.7 Slice layer RBSP

The slice layer RBSP consists of a slice header defined in subclause 8.3 and the following slice data.

8.2.8 Data partition RBSP

When data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains the header symbols of all coded MBs, partition B the intra Coded Block Patterns (cbps) and coefficients, and partition C the inter cbps 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. Note that the coding order of slices may not be identical to the normal scan order.

8.3 Slice header

8.3.1 Parameter set identifier (parameter_set_id)

The parameter set identifier indicates the parameter set in use.

8.3.2 First MB in slice (first_mb_in_sliceX)

Horizontal macroblock position of the first macroblock contained in this slice.

8.3.2 First MB in slice (first_mb_in_sliceY)

Vertical macroblock position of the first macroblock contained in this slice.

8.3.3 Number of active reference frames (num_ref_frame_pic_active_fwd_minus1)

Number of reference pictures in the forward reference set used to decode the picture minus 1.

8.3.3 Number of active reference frames (num_ref_frame_pic_active_bwd_minus1)

Number of reference pictures in the backward reference set used to decode the picture minus 1.

8.3.4 Slice-level QP (slice_delta_qp)

In the slice layer, a slice_delta_qp parameter is sent. The entropy-coded representation of this parameter is defined in the same manner as for the value of delta_qp in the macroblock layer. The decoded value of this parameter shall be in the range of -26 to +25, inclusive. From this value, the initial QP_Y parameter for the slice is computed as:

$$QP_Y = 26 + \text{slice\_delta\_qp} \quad (8\text{-}1)$$

The value of QP_Y is initialized 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.

8.3. SP and SI slice QP (slice_delta_qp_s)

For SP and SI frames the SP slice slice_delta_qp_sp is sent in the slice layer. The entropy-coded representation of this parameter is defined in the same manner as for the value of delta_qp in the macroblock layer. The decoded value of this parameter shall be in the range of -26 to +25, inclusive. The $QPSP_Y$ parameter for the slice is computed from slice_delta_qp_sp as:

$$QPSP_Y = 26 + \text{slice\_delta\_qp\_sp} \qquad (8\text{-}2)$$

This value of $QPSP_Y$ is used for the decoding of all macroblocks in the slice.

8.3.6 Number of macroblocks in slice (num_mbs_in_slice)

For CABAC entropy coding the number of macroblocks in the slice is transmitted.

8.4 Reference picture selection layer (rps_layer)

8.4.2 Reference picture reordering indicator (reference_picture_reordering_indicator_fwd/bwd)

reference_picture_reordering_indicator_fwd/bwd indicates the presence of syntax elements to indicate a reordering of the active reference . reference_picture_reordering_indicator_fwd/bwd shall be one of the following two values: “0”: remapping_of_frame_nums_indicator, abs_diff_pic_numbers, and long_term_pict_index are not present. In this case the default buffer indexing order presented in the next subsection shall be applied. RPS layer information sent at the slice layer does not affect the decoding process of any other slice. “1”: remapping_of_frame_nums_indicator, abs_diff_pic_numbers, and long_term_pict_index are present. In this case, the buffer indexing used to decode the current slice and to manage the contents of the picture buffer is sent using the following code words.

8.4.3 Re-Mapping of Frame numbers Indicator (remapping_of_frame_nums_indicator)

remapping_of_frame_nums_indicator is present in the RPS layer if the picture is a P, B, or SP picture. It is not present if the picture is an I picture. It is present at least once in P and SP pictures and at least twice in B pictures. remapping_of_frame_nums_indicator indicates whether any default picture indices are to be re-mapped for forward or backward motion compensation of the current slice – and how the re-mapping of the relative indices into the multi-picture buffer is to be specified if indicated. The interpretation of remapping_of_frame_nums_indicator is shown in Figure 8-2. If remapping_of_frame_nums_indicator indicates the presence of an abs_diff_pic_numbers or long_term_pict_index field, an additional remapping_of_frame_nums_indicator field immediately follows the abs_diff_pic_numbers or long_term_pict_index field.

A picture reference index parameter (ref_idx_fwd or ref_idx_bwd) is a relative index into an ordered set of reference pictures. For the coding of a P picture, there is one such set of reference pictures, called the forward reference set. For the coding of a B picture, there are two such sets of reference pictures, called the forward and backward reference sets.

The remapping_of_frame_nums_indicator, abs_diff_pic_numbers, and long_term_pict_index fields allow the order of that relative indexing into the multi-picture buffer to be temporarily altered from the default index order for the decoding of a particular slice. An remapping_of_frame_nums_indicator "end loop" indication indicates the end of a list of re-ordering commands for the forward or backward reference set.

8.4.3.1 Default index order for P pictures

The default index order for forward prediction of P frames 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 most recently-transmitted reference frame and proceeding through to the reference frame in the short-term buffer that was transmitted first (i.e., in decreasing order of frame number in the absence of wrapping of the ten-bit frame number field). 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 long-term index equal to the most recent value of max_long_term_index_plus1-1.

For example, if the buffer contains three short-term frames with short-term frame numbers 300, 302, and 303 (which were transmitted in increasing frame-number order) 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 number 303,

default relative index 1 refers to the short-term frame with frame number 302, default relative index 2 refers to the short-term frame with frame number 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.

For the purposes of default reference picture index calculation, the transmitted order of reference frames is calculated as the order in which frames occurred in the bitstream, regardless of whether the transmitted frame had been coded as a pair of two separate field-structured pictures, or a single frame-structured picture.

In the case that the current picture is field-structured, each field of the stored reference frames is identified as a separate reference picture with a unique index. Thus field structured pictures effectively have twice the number of pictures available for referencing. The calculated transmission 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 transmitted nor stored, the transmission order calculation shall ignore the missing field and instead index the next available stored reference field of the respective parity in transmission order. The decoder shall treat the missing field as “unknown” data, while encoder shall not generate bitstreams that have data dependences on the missing field.

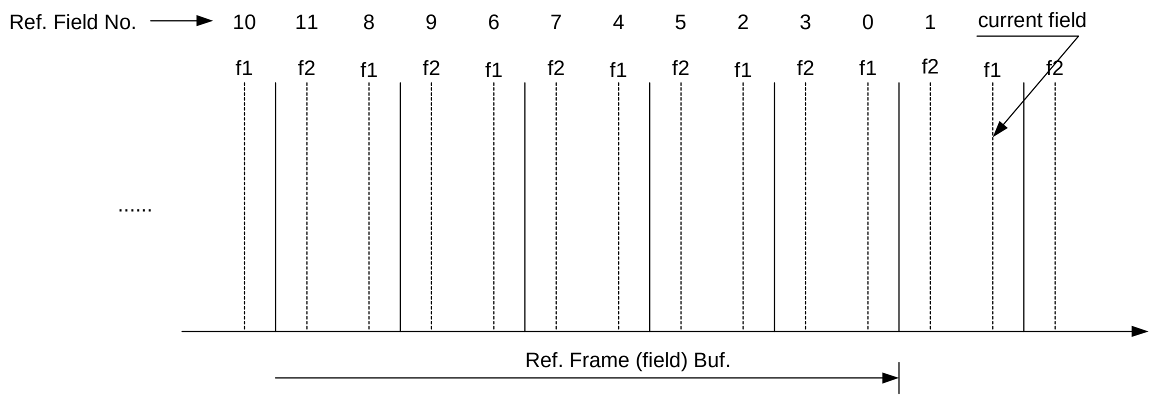


Figure 8-1 – Default reference field number assignment when the current picture is the first field coded in a frame

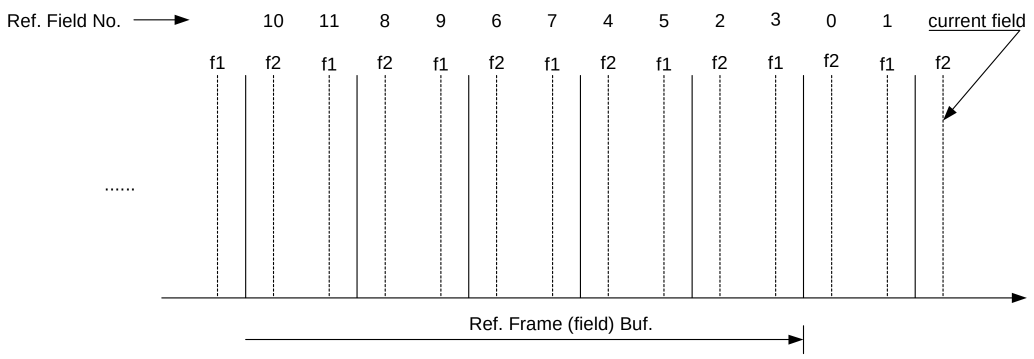


Figure 8-2 - Default reference field number assignment when the current picture is the second field coded in a frame

8.4.3.2 Default index order for B pictures

The default index order for B frames is defined such that short-term frames that temporally precede the B frame are distinguished from short-term frames that temporally follow the B frame, based on the display order for each reference frame. The default order for forward prediction (subclause 8.2.13.2.1) is specified in a manner that gives a lower index order to short-term frames that temporally precede the current frame, and the default order for backward prediction (subclause 8.2.13.2.1) is specified in a manner that ordinarily gives a lower index order to short-term frames that temporally follow the current frame.

The default index order for B field pictures is split between even indices starting at 0 for fields of the same parity (top or bottom) as the current field and odd indices starting at 1 for fields of the opposite parity as the current field. This split ordering is analogous to the ordering defined in P pictures above except that a B field picture never references the opposite-parity field with which it shares the current frame.

8.4.3.2.1 Forward prediction in B pictures

Within the set of short-term pictures, the default order for B-frame forward prediction shall be for the frames to be ordered starting with the most recently-transmitted temporally-preceding reference frame and proceeding through to the temporally-preceding reference frame in the short-term buffer that was transmitted first. These temporally-preceding frames shall then be followed by the temporally-following reference frames, starting with the most recently-transmitted temporally-following reference frame in the short-term buffer and proceeding through to the temporally-following reference frame in the short-term buffer that was transmitted first. These frames shall then be followed by the long-term frames, starting with the frame with the smallest long-term index and proceeding up to the frame with long-term index equal to the most recent value of max_long_term_index_plus1-1. The default ordering for B field pictures follows in like manner except that it is split between even indices for same-parity fields and odd indices for opposite-parity fields.

8.4.3.2.2 Backward prediction in B pictures

The default order for B-picture backward prediction is defined in a similar manner as for forward prediction, but giving preference in the order for pictures that temporally follow the B picture and swapping the order of the first two pictures if this would result in an identical forward and backward default indexing order.

Within the set of short-term pictures, the ordinary default order for B-picture backward prediction shall be for the pictures to be ordered starting with the most recently-transmitted temporally-following reference picture and proceeding through to the temporally-following reference picture that has been in the short-term buffer the longest. These temporally-following pictures shall then be followed by the temporally-preceding reference pictures, starting with the most recently-transmitted temporally-preceding reference picture and proceeding through to the temporally-preceding reference picture that has been in the short-term buffer the longest. These pictures shall then be followed by the long-term pictures, starting with the picture with the smallest long-term index and proceeding up to the picture with long-term index equal to the most recent value of max_long_term_index_plus1-1.

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

8.4.3.3 Reordering of forward and backward reference sets

The first abs_diff_pic_numbers or long_term_pict_index field that is received (if any) moves a specified picture out of the default order to the relative index of zero. The second such field moves a specified picture to the relative index of one, etc. The set of remaining pictures not moved to the front of the relative indexing order in this manner shall retain their default order amongst themselves and shall follow the pictures that have been moved to the front of the buffer in relative indexing order. Note that if the current picture is a frame, then abs_diff_pic_numbers or long_term_pic_index refer to the relative frame indices defined above; if the current picture is a field, then abs_diff_pic_numbers or long_term_pic_index refer to the relative field indices defined above.

If there is not more than one forward reference picture used, no more than one abs_diff_pic_numbers or long_term_pict_index field shall be present in the same RPS layer unless the current picture is a B picture. If the current picture is a B picture and there is not more than one backward reference picture used, no more than two abs_diff_pic_numbers or long_term_pict_index fields shall be present in the same RPS layer.

Any re-mapping of frame numbers specified for some slice shall not affect the decoding process for any other slice.

In a P picture an remapping_of_frame_nums_indicator “end loop” indication is followed by ref_pic_buffering_type. In a B picture, the first remapping_of_frame_nums_indicator "end loop" indication, which concludes the remapping of the forward reference set, is followed by an additional remapping_of_frame_nums_indicator indicator that begins the remapping operations (if any) for the backward reference set.

Within one RPS layer, remapping_of_frame_nums_indicator shall not specify the placement of any individual reference picture into more than one re-mapped position in relative index order.

Table 8-7 – remapping_of_frame_nums_indicator operations for re-mapping of reference pictures

Code Number	Re-mapping Specified
0	abs_diff_pic_numbers field is present and corresponds to a negative difference to add to a frame number prediction value
1	abs_diff_pic_numbers field is present and corresponds to a positive difference to add to a frame number prediction value
2	long_term_pict_index field 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

8.4.3.4 Absolute Difference of Frame numbers (abs_diff_pic_numbers)

abs_diff_pic_numbers is present only if indicated by remapping_of_frame_nums_indicator. abs_diff_pic_numbers follows remapping_of_frame_nums_indicator when present. The code number of the UVLC corresponds to abs_diff_pic_numbers – 1. abs_diff_pic_numbers represents the absolute difference between the frame number of the currently re-mapped picture and the prediction value for that frame number. If no previous abs_diff_pic_numbers fields have been sent within the current RPS layer, the prediction value shall be the frame number of the current picture. If some previous abs_diff_pic_numbers field has been sent, the prediction value shall be the frame number of the last picture that was re-mapped using abs_diff_pic_numbers.

If the frame number prediction is denoted PNP, and the frame number in question is denoted PNQ, and MAX_PNF equals MAX_PN if the current picture is a frame, and MAX_PNF equals 2*MAX_PN if the current picture is a field, the decoder shall determine PNQ from PNP and abs_diff_pic_numbers in a manner mathematically equivalent to the following:

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

The encoder shall control remapping_of_frame_nums_indicator and abs_diff_pic_numbers such that the decoded value of abs_diff_pic_numbers 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_numbers and remapping_of_frame_nums_indicator to specify a re-mapped frame number in question, PNQ:

```
DELTA = PNQ – PNP;
if (DELTA < 0) {
  if (DELTA < –MAX_PNF/2-1)
    MDELTA = DELTA + MAX_PNF;
  else
    MDELTA = DELTA;
}else{
  if (DELTA > MAX_PNF/2)
    MDELTA = DELTA – MAX_PNF;
  else
    MDELTA = DELTA;
```

```
        }
        abs_diff_pic_numbers = abs(MDELTA);
```

where abs() indicates an absolute value operation. Note that the code number of the UVLC corresponds to the value of abs_diff_pic_numbers – 1, rather than the value of abs_diff_pic_numbers itself.

remapping_of_frame_nums_indicator would then be determined by the sign of MDELTA.

8.4.3.5 Long-term Picture Index for Re-Mapping (long_term_pict_index)

long_term_pict_index is present only if indicated by remapping_of_frame_nums_indicator. long_term_pict_index follows remapping_of_frame_nums_indicator when present. long_term_pict_index is transmitted using UVLC codewords. It represents the long-term picture index to be re-mapped. The prediction value used by any subsequent abs_diff_pic_numbers re-mappings is not affected by long_term_pict_index.

8.4.4 Reference Picture Buffering Type (ref_pic_buffering_type)

ref_pic_buffering_type specifies the buffering type of the currently decoded picture picture and thus decides how the multi-picture buffer will be modified after the current picture is decoded. It follows an remapping_of_frame_nums_indicator “end loop” indication when the picture is not an I picture. It is the first element of the RPS layer if the picture is an I picture. The values for ref_pic_buffering_type are defined as follows:

ref_pic_buffering_type = = “0”:	Sliding Window,
ref_pic_buffering_type = = “1”:	Adaptive Memory Control.

In the “Sliding Window” buffering type, the current decoded picture shall be added to the buffer with default relative index 0, and any marking of pictures as “unused” in the buffer is performed automatically in a first-in-first-out fashion among the set of short-term pictures. In this case, if the buffer has sufficient “unused” capacity to store the current picture, no additional pictures shall be marked as “unused” in the buffer. If the buffer does not have sufficient “unused” capacity to store the current picture, the picture with the largest default relative index among the short-term pictures in the buffer shall be marked as “unused”. If in this case the current picture is the first field of a frame, then only the field of the same parity (top or bottom) will be marked as unused in the buffer, so that the second field of the current frame may still reference the other field of the largest relative index. In the sliding window buffering type, no additional information is transmitted to control the buffer contents.

In the "Adaptive Memory Control" buffering type, the encoder explicitly specifies any addition to the buffer or marking of data as “unused” in the buffer, and may also assign long-term indices to short-term frames. The current frame and other frames may be explicitly marked as “unused” in the buffer, as specified by the encoder. This buffering type requires further information that is controlled by memory management control operation (memory_management_control_operation) parameters.

8.4.4.1 Memory Management Control Operation (memory_management_control_operation)

memory_management_control_operation is present only when ref_pic_buffering_type indicates “Adaptive Memory Control”, and may occur multiple times if present. It specifies a control operation to be applied to manage the multi-frame buffer memory. The memory_management_control_operation parameter is followed by data necessary for the operation specified by the value of memory_management_control_operation, and then an additional memory_management_control_operation parameter follows – until the memory_management_control_operation value indicates the end of the list of such operations. memory_management_control_operation commands do not affect the buffer contents or the decoding process for the decoding of the current frame – rather, they specify the necessary buffer status for the decoding of subsequent frames in the bitstream. The values and control operations associated with memory_management_control_operation are defined in Table 8-3.

Note that memory_management_control_operation commands always indicate the same treatment for both fields of a frame, and the frame numbers referred to in memory_management_control_operation commands are always frame numbers even if the current picture is a field picture.

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

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

A “stored frame” does not contain an memory_management_control_operation command in its RPS layer which marks that (entire) frame as “unused”. If the current frame is not a stored frame, its RPS layer 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 frame (other than the current frame) as “unused” that has not also been marked as “unused” in the RPS layer of a prior stored frame, or

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

Table 8-8 – Memory management control operation (memory_management_control_operation) values

Code Number	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 Frame as “Unused”	difference_of_frame_nums
2	Mark a Long-Term Frame as “Unused”	long_term_frame_index
3	Assign a Long-Term Index to a Frame	difference_of_frame_nums and long_term_frame_index
4	Specify the Maximum Long-Term Frame Index	max_long_term_index_plus1
5	Reset	None

8.4.4.2 Difference of Frame numbers (difference_of_frame_nums)

difference_of_frame_nums is present when indicated by memory_management_control_operation. difference_of_frame_nums follows memory_management_control_operation if present. difference_of_frame_nums is transmitted using UVLC codewords and is used to calculate the PN of a frame for a memory control operation. It is used in order to assign a long-term index to a frame, mark a short-term frame as “unused”. If the current decoded frame number is PNC and the decoded UVLC code number is difference_of_frame_nums, an operation mathematically equivalent to the following equations shall be used for calculation of PNQ, the specified frame number in question:

```
if (PNC – difference_of_frame_nums < 0)
  PNQ = PNC – difference_of_frame_nums + MAX_PN;
else
  PNQ = PNC – difference_of_frame_nums;
```

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

```
if (PNC – PNQ < 0)
  difference_of_frame_nums = PNC – PNQ + MAX_PN;
else
  difference_of_frame_nums = PNC – PNQ;
```

For example, if the decoded value of difference_of_frame_nums is zero and memory_management_control_operation indicates marking a short-term frame as “unused”, the current decoded frame shall be marked as “unused” (thus indicating that the current frame is not a stored frame).

8.4.4.3 Long-term Frame Index (long_term_frame_index)

long_term_frame_index is present when indicated by memory_management_control_operation. long_term_frame_index specifies the long-term frame index of a frame. It follows difference_of_frame_nums if the operation is to assign a long-term index to a frame. It follows memory_management_control_operation if the operation is to mark a long-term frame as “unused”.

8.4.4.4 Maximum Long-Term Frame Index Plus 1 (max_long_term_index_plus1)

max_long_term_index_plus1 is present if indicated by memory_management_control_operation. max_long_term_index_plus1 follows memory_management_control_operation if present. If present, max_long_term_index_plus1 is used to determine the maximum index allowed for long-term reference frames (until receipt of another value of max_long_term_index_plus1). The decoder shall initially assume max_long_term_index_plus1 is "0" until some other value has been received. Upon receiving an max_long_term_index_plus1 parameter, the decoder shall consider all long-term frames having indices greater than the decoded value of max_long_term_index_plus1 – 1 as “unused” for referencing by the decoding process for subsequent frames. For all other frames in the multi-frame buffer, no change of status shall be indicated by max_long_term_index_plus1.

8.5 Macroblock layer

Following the syntax diagram for the macroblock elements, the various elements are described.

8.5.1 Number of Skipped Macroblocks (mb_skip_run)

Number of Skipped Macroblocks indicates the number of consecutive macroblocks coded in Skip mode. Skip mode macroblocks are motion-compensated using the last decoded picture as reference. The motion vector shall be obtained as described in subclause 8.4.6.1.3. Motion Vector Data and prediction error coding information are omitted for all the Skip mode macroblocks.

For a B macroblock skip means direct mode without coefficients. mb_skip_run indicates the number of skipped macroblocks in an inter- or B-picture before a coded macroblock. If a picture or slice ends with one or more skipped macroblocks, they are represented by an additional mb_skip_run which counts the number of skipped macroblocks.

For the explicit B prediction block weighting, the ABP coefficient corresponding to abp_coeff_idx=0 is used.

8.5.2 Macro block type (mb_type)

Refer to Table 9-7. There are different MB-Type tables for Intra and Inter frames.

8.5.2.1 Intra Macroblock modes

Intra 4x4 Intra coding as defined in sections QQ to QQ.

Imode, nc, AC See definition in section QQ. These modes refer to 16x16 intra coding.

8.5.2.2 Inter Macroblock Modes

Skip No further information about the macroblock is transmitted. The motion vector for a Skip mode macroblock shall be obtained as described in subclause 8.4.6.1.3. If picture_struct indicates a frame, then the decoded frame with ref_pic_idx_fwd = = 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 picture_struct indicates a field, then the decoded field of the same parity (top or bottom) with ref_pic_idx_fwd = = 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.

NxM (eg. 16x8) The macroblock is predicted from a past picture with block size NxM. For the macroblock modes 16x16, 16x8, and 8x16, a motion vector is provided for each NxM block. If N=M=8, for each 8x8 sub-partition an additional codeword is transmitted which indicates in which mode the corresponding sub-partition is coded (see subclause 8.4.3). Depending on N,M and the 8x8 sub-partition modes there may be 1 to 16 sets of motion vector data for a macroblock.

Intra 4x4 4x4 intra coding.

Code numbers from 6 and upwards represent 16x16 intra coding.

8.5.3 8x8 sub-partition modes

NxM (eg. 8x4) The corresponding 8x8 sub-partition is predicted from a past picture with block size NxM. A motion vector is transmitted for each NxM block. Depending on N and M, up to 4 motion vectors are coded for an 8x8 sub-partition, and thus up to 16 motion vectors are transmitted for a macroblock.

Intra The 8x8 sub-partition is coded in intra 4x4 mode.

8.5.4 Intra Coding

In intra mode, prediction is always used for each sub block in a macroblock. A 4x4 block is to be coded (samples labeled a to p below). The samples A to Q from neighbouring blocks may already be decoded and may be used for prediction. When samples E-H are not available, whether because they have not yet been decoded, are outside the picture or outside the current slice, 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.

8.5.4.1 Coding of Intra 4x4 prediction modes

Since each of the 4x4 luma blocks is assigned a prediction mode, this will require a considerable number of bits if coded independently. The chosen prediction of a block is highly correlated with the prediction modes of adjacent blocks. This is illustrated in Figure 8-4. When the prediction modes of A and B are known (including the case that A or B or both are outside the slice) an ordering of the most probable, next most probable etc. of C is given. When an adjacent block is coded by 16x16 intra mode, prediction mode is “mode 0: DC_prediction”; when it is coded in inter mode, prediction mode is “mode 0: DC_prediction” in the usual case and “outside” in the case of constrained intra update. This ordering is listed in Table 8-3.

For each prediction mode of A and B a list of 9 numbers is given in Table 8-3. Example: Prediction mode for A and B is 2. The string 2 8 7 1 0 6 4 3 5 indicates that mode 2 is also the most probable mode for block C. Mode 8 is the next most probable one etc. In the bitstream there will for instance be information that Prob0 = 1 (see Table 9-5) indicating that the next most probable mode shall be used for block C. In our example this means Intra prediction mode 8. Use of '–' in the table indicates that this instance cannot occur because A or B or both are outside the slice.

For more efficient coding, information on intra prediction of two 4x4 luma blocks are coded in one codeword (Prob0 and Prob1 in Table 9-5). The order of the resulting eight codewords is indicated in Table 8-4.

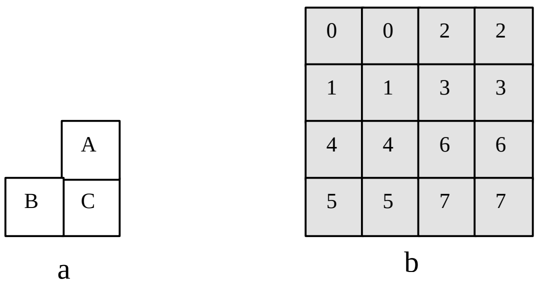


Figure 8-3 – a) Prediction mode of block C to be established, where A and B are adjacent blocks. b) order of intra prediction information in the bitstream

Table 8-9 – Prediction mode as a function of ordering indicated in the bitstream

Index	B/A	Outside	0	1	2	3
0a	Outside	0--------	01-------	10-------	---------	---------
1a	0	02-------	021648573	125630487	021876543	021358647
2a	1	---------	102654387	162530487	120657483	102536487
3a	2	20-------	280174365	217683504	287106435	281035764
4a	3	---------	201385476	125368470	208137546	325814670
5a	4	---------	201467835	162045873	204178635	420615837
6a	5	---------	015263847	152638407	201584673	531286407
7a	6	---------	016247583	160245738	206147853	160245837
8a	7	---------	270148635	217608543	278105463	270154863
9a	8	---------	280173456	127834560	287104365	283510764

Table 8-9 (concluded)

Index	B/A	4	5	6	7	8
0b	Outside	---------	---------	---------	---------	---------
1b	0	206147583	512368047	162054378	204761853	208134657
2b	1	162045378	156320487	165423078	612047583	120685734

3b	2	287640153	215368740	216748530	278016435	287103654
4b	3	421068357	531268470	216584307	240831765	832510476
5b	4	426015783	162458037	641205783	427061853	204851763
6b	5	125063478	513620847	165230487	210856743	210853647
7b	6	640127538	165204378	614027538	264170583	216084573
8b	7	274601853	271650834	274615083	274086153	278406153
9b	8	287461350	251368407	216847350	287410365	283074165

8.5.4.2 Coding of mode information for Intra-16x16 mode

See Table 7. Three parameters have to be indicated. They are all included in MB-type.

Imode: 0,1,2,3

AC: 0 means there are no ac coefficients in the 16x16 block. 1 means that there is at least one ac coefficient and all 16 blocks are scanned.

nc: cbp for chroma (see qq)

8.5.5 Reference picture (ref_idx_fwd/bwd)

If PTYPE indicates possibility of prediction from more than one previously-decoded picture, the exact picture to be used must be indicated. This is done according to the following tables. If picture_struct indicates that the current picture is a frame picture, then the reference picture is a previous frame in the forward reference buffer 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_number	Reference frame
0	The first frame in the forward reference set
1	The second frame in the forward reference set
2	The third frame in the forward reference set
..	..

If num_ref_pic_active_fwd_minus1 is equal to 0, ref_idx_fwd is not present. If num_ref_pic_active_fwd_minus1 is equal to 1, only a single encoded bit is used to represent ref_idx_fwd. If num_ref_pic_active_fwd_minus1 is greater than 1, the value of ref_idx_fwd is represented by a decoded index.

If num_ref_pic_active_bwd_minus1 is equal to 0, ref_idx_bwd is not present. If num_ref_pic_active_bwd_minus1 is equal to 1, only a single encoded bit is used to represent ref_idx_bwd. If num_ref_pic_active_bwd_minus1 is greater than 1, the value of ref_idx_bwd is represented by a decoded index.

The reference index parameter is transmitted for each 16x16, 16x8, or 8x16 motion compensation block. If the macroblock is coded in 8x8 mode, the reference frame parameter is coded once for each 8x8 sub-partition unless the 8x8 sub-partition is transmitted in intra mode. If the UVLC is used for entropy coding and the macroblock type is indicated by codeword 4 (8x8, ref=0), no reference frame parameters are transmitted for the whole macroblock.

8.5.6 Motion Vector Data (mvd_fwd)

If so indicated by MB_type, vector data for 1-16 blocks are transmitted. For every block a prediction is formed for the horizontal and vertical components of the motion vector. mvd_fwd signals the difference between the vector component to be used and this prediction. The order in which vector data is sent is indicated in Figure 2. Motion vectors are allowed to point to samples outside the reference frame. If a sample outside the reference frame is referred to in the prediction process, the nearest sample belonging to the frame (an edge or corner sample) shall be used. All fractional sample positions shall be interpolated as described in subclause 9.2. If a sample referred in the interpolation process (necessarily integer accuracy) is outside of the reference frame it shall be replaced by the nearest sample belonging to the frame (an edge or corner sample). Reconstructed motion vectors shall be clipped to ±19 integer samples outside of the frame.

8.5.7 Coded Block Pattern (cbp)

The cbp contains information of which 8x8 blocks - luma and chroma - contain transform coefficients. Notice that an 8x8 block contains 4 4x4 blocks meaning that the statement '8x8 block contains coefficients' means that 'one or more of the 4 4x4 blocks contain coefficients'. The 4 least significant bits of cbp contain information on which of 4 8x8 luma blocks in a macroblock contains nonzero coefficients. Let us call these 4 bits cbpY. The ordering of 8x8 blocks is indicated in Figure 3. A 0 in position n of cbp (binary representation) means that the corresponding 8x8 block has no coefficients whereas a 1 means that the 8x8 block has one or more non-zero coefficients.

For chroma we define 3 possibilities:

nc=0: no chroma coefficients at all.

nc=1 There are nonzero 2x2 transform coefficients. All chroma AC coefficients = 0. Therefore we do not send any EOB for chroma AC coefficients.

nc=2 There may be 2x2 nonzero coefficients and there is at least one nonzero chroma AC coefficient present. In this case we need to send 10 EOBs (2 for DC coefficients and 2x4=8 for the 8 4x4 blocks) for chroma in a macroblock.

The total cbp for a macroblock is: cbp = cbpY + 16xnc

The cbp is indicated with a different codeword for Inter macroblocks and Intra macroblocks since the statistics of cbp values are different in the two cases.

8.5.8 Change of Quantiser Value (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 defined by:

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

The decoded value of delta_qp shall be in the range from -26 to +25, inclusive, which enables the value of QP_Y to be changed to any value in the range [0..51], as specified by

$$QP_Y = (QP_Y + \text{delta\_qp} + 52) \,\%\, 52 \tag{8-5}$$

9 Decoding Process

9.1 Slice Decoding

A slice is decoded as follows:

If the frame number of the slice to be decoded is different from the frame number of the previously decoded slice, decoding of a new picture is started. Otherwise, decoding of the current picture is continued.

The macroblock address of the first macroblock of the slice is used as an index to the macroblock allocation map to determine which slice group the slice belongs to. Using the calculated macroblock address, the macroblock can be reconstructed directly. Since no other macroblock belonging to this slice has been decoded (as it is the first macroblock after the slice header), all in-picture prediction mechanisms are reset before decoding and reconstruction.

When mb_allocation_map_type equals to zero, the macroblock address of the next macroblock is implicitly the current macroblock address plus 1. When mb_allocation_map_type is not zero, the next macroblock address is determined by searching the macroblock allocation map for the next macroblock in raster-scan order belonging to the slice group being decoded.

9.1.1 Multi-Picture Decoder Process

The decoder stores the reference pictures for inter-picture decoding in a multi-picture buffer. The decoder replicates the multi-picture buffer of the encoder according to the reference picture buffering type and any memory management control operations specified in the bitstream. The buffering scheme may also be operated when partially erroneous pictures are decoded.

Each coded and stored picture is assigned a Frame number (PN) which is stored with the picture in the multi-picture buffer. PN represents a sequential picture counting identifier for stored pictures. PN is constrained, using modulo MAX_PN arithmetic operation. For the first coded picture of an independent GOP, PN shall be "0". For each and every other coded and stored picture, PN shall be increased by 1.

Besides the PN, each picture stored in the multi-picture buffer has an associated index, called the default relative index. When a picture is first added to the multi-picture buffer it is given default relative index 0 – unless it is assigned to a long-term index. The default relative indices of pictures in the multi-picture buffer are modified when pictures are added to or removed from the multi-picture buffer, or when short-term pictures are assigned to long-term indices.

The pictures stored in the multi-picture buffers can also be divided into two categories: long-term pictures and short-term pictures. A long-term picture can stay in the multi-picture buffer for a long time (more than MAX_PN-1 coded and stored picture intervals). The current picture is initially considered a short-term picture. Any short-term picture can be changed to a long-term picture by assigning it a long-term index according to information in the bitstream. The PN is the unique ID for all short-term pictures in the multi-picture buffer. When a short-term picture is changed to a long-term picture, it is also assigned a long-term picture index (long_term_picture_index). A long-term picture index is assigned to a picture by associating its PN to an long_term_picture_index. Once a long-term picture index has been assigned to a picture, the only potential subsequent use of the long-term picture's PN within the bitstream shall be in a repetition of the long-term index assignment. long_term_picture_index becomes the unique ID for the life of a long-term picture.

PN (for a short-term picture) or long_term_picture_index (for a long-term picture) can be used to re-map the pictures into re-mapped relative indices for efficient reference picture addressing.

9.1.1.1 Decoder Process for Short/Long-term Picture Management

The decoder may have both long-term pictures and short-term pictures in its multi-picture buffer. The max_long_term_index_plus1 field is used to indicate the maximum long-term picture index allowed in the buffer. If no prior value of max_long_term_index_plus1 has been sent, no long-term pictures shall be in use, i.e. max_long_term_index_plus1 shall initially have an implied value of "0". Upon receiving an max_long_term_index_plus1 parameter, a new max_long_term_index_plus1 shall take effect until another value of max_long_term_index_plus1 is received. Upon receiving a new max_long_term_index_plus1 parameter in the bitstream, all long-term pictures with associated long-term indices greater than or equal to max_long_term_index_plus1 shall be considered marked "unused". The frequency of transmitting max_long_term_index_plus1 is out of the scope of this Recommendation. However, the encoder should send an max_long_term_index_plus1 parameter upon receiving an error message, such as an Intra request message.

A short-term picture can be changed to a long-term picture by using an memory_management_control_operation command with an associated difference_of_frame_nums and long_term_picture_index. The short-term frame number is derived from difference_of_frame_nums and the long-term picture index is long_term_picture_index. Upon receiving such an memory_management_control_operation command, the decoder shall change the short-term picture with PN indicated by difference_of_frame_nums to a long-term picture and shall assign it to the long-term index indicated by long_term_picture_index. If a long-term picture with the same long-term index already exists in the buffer, the previously-existing long-term picture shall be marked "unused". An encoder shall not assign a long-term index greater than max_long_term_index_plus1–1 to any picture. If long_term_picture_index is greater than max_long_term_index_plus1–1, this condition should be treated by the decoder as an error. For error resilience, the encoder may send the same long-term index assignment operation or max_long_term_index_plus1 specification message repeatedly. If the picture specified in a long-term assignment operation is already associated with the required long_term_picture_index, no action shall be taken by the decoder. An encoder shall not assign the same picture to more than one long term index value. If the picture specified in a long-term index assignment operation is already associated with a different long-term index, this condition should be treated as an error. An encoder shall only change a short-term picture to a long-term picture if its PN has not been used in any subsequent coded picture. An encoder 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. An encoder shall not assign a long-term index to a frame number that has not been sent.

9.1.1.2 Decoder Process for Reference Picture Buffer Mapping

The decoder employs indices when referencing a picture for motion compensation on the macroblock layer. In pictures other than B pictures, these indices are the default relative indices of pictures in the multi-picture buffer when the fields abs_diff_pic_numbers and long_term_pict_index are not present in the current slice layer as applicable, and are re-mapped relative indices when these fields are present. In B pictures, the first one or two pictures (depending on BTPSM) in relative index order are used for backward prediction, and the forward picture reference parameters specify a relative index into the remaining pictures for use in forward prediction. (needs to be changed)

The indices of pictures in the multi-picture buffer can be re-mapped onto newly specified indices by transmitting the remapping_of_frame_nums_indicator, abs_diff_pic_numbers, and long_term_pict_index fields. remapping_of_frame_nums_indicator indicates whether abs_diff_pic_numbers or long_term_pict_index is present. If abs_diff_pic_numbers is present, remapping_of_frame_nums_indicator specifies the sign of the difference to be added to a frame number prediction value. The abs_diff_pic_numbers value corresponds to the absolute difference between the PN of the picture to be re-mapped and a prediction of that PN value. The first transmitted abs_diff_pic_numbers is computed as the absolute difference between the PN of the current picture and the PN of the picture to be re-mapped.

The next transmitted abs_diff_pic_numbers field represents the difference between the PN of the previous picture that was re-mapped using abs_diff_pic_numbers and that of another picture to be re-mapped. The process continues until all necessary re-mapping is complete. The presence of re-mappings specified using long_term_pict_index does not affect the prediction value for subsequent re-mappings using abs_diff_pic_numbers. If remapping_of_frame_nums_indicator indicates the presence of an long_term_pict_index field, the re-mapped picture corresponds to a long-term picture with a long-term index of long_term_pict_index. If any pictures are not re-mapped to a specific order by remapping_of_frame_nums_indicator, these remaining pictures shall follow after any pictures having a re-mapped order in the indexing scheme, following the default order amongst these non-re-mapped pictures.

If the indicated parameter set in the latest received slice or data partition signals the required frame number update behavior, the decoder shall operate as follows. The default picture index order shall be updated as if pictures corresponding to missing frame numbers were inserted to the multi-picture buffer using the “Sliding Window” buffering type. An index corresponding to a missing frame number 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 the indicated parameter set in the latest received slice or data partition does not signal the required frame number update behavior, the decoder should infer an unintentional picture loss if one or several frame numbers are missing or if a picture not stored in the multi-picture buffer is indicated in a transmitted abs_diff_pic_numbers or long_term_pict_index.

In case of an unintentional picture loss, the decoder may invoke some concealment process. If the required frame number update behavior was indicated, the decoder may replace the picture corresponding to an “invalid” index with an error-concealed one and remove the “invalid” indication. If the required frame number update behavior was not indicated, the decoder may insert an error-concealed picture into the multi-picture buffer assuming the “Sliding Window” buffering type. Concealment may be conducted by copying the closest temporally preceding picture that is available in the multi-picture buffer into the position of the missing picture. The temporal order of the short-term pictures in the multi-picture buffer can be inferred from their default relative index order and PN fields. In addition or instead, the decoder may send a forced intra update signal to the encoder by external means (for example, Recommendation H.245), 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.

9.1.1.3 Decoder Process for Multi-Picture Motion Compensation

Multi-picture motion compensation is applied if the use of more than one reference picture is indicated. For multi-picture motion compensation, the decoder chooses a reference picture as indicated using the reference frame fields on the macroblock layer. Once, the reference picture is specified, the decoding process for motion compensation proceeds.

9.1.1.4 Decoder Process for Reference Picture Buffering

The buffering of the currently decoded picture can be specified using the reference picture buffering type (ref_pic_buffering_type). The buffering may follow a first-in, first-out ("Sliding Window") mode. Alternatively, the buffering may follow a customized adaptive buffering ("Adaptive Memory Control") operation that is specified by the encoder in the forward channel.

The "Sliding Window" buffering type operates as follows. First, the decoder determines whether the picture can be stored into “unused” buffer capacity. If there is insufficient “unused” buffer capacity, the short-term picture with the largest default relative index (i.e. the oldest short-term picture in the buffer) shall be marked as “unused”. The current picture is stored in the buffer and assigned a default relative index of zero. The default relative index of all other short-term pictures is incremented by one. The default relative indices of all long-term pictures are incremented by one minus the number of short-term pictures removed.

In the "Adaptive Memory Control" buffering type, specified pictures may be removed from the multi-picture buffer explicitly. The currently decoded picture, which is initially considered a short-term picture, may be inserted into the buffer with default relative index 0, may be assigned to a long-term index, or may be marked as “unused” by the encoder. Other short-term pictures may also be assigned to long-term indices. The buffering process shall operate in a manner functionally equivalent to the following: First, the current picture is added to the multi-picture buffer with default relative index 0, and the default relative indices of all other pictures are incremented by one. Then, the memory_management_control_operation commands are processed:

If memory_management_control_operation indicates a reset of the buffer contents or if the current picture is the first one in an independent GOP, all pictures in the buffer are marked as “unused” except the current picture (which will be the picture with default relative index 0). Moreover, the maximum long-term index shall be reset to zero.

If memory_management_control_operation indicates a maximum long-term index using max_long_term_index_plus1, all long-term pictures having long-term indices greater than or equal to max_long_term_index_plus1 are marked as “unused” and the default relative index order of the remaining pictures are not affected.

If memory_management_control_operation indicates that a picture is to be marked as “unused” in the multi-picture buffer and if that picture has not already been marked as “unused”, the specified picture is marked as “unused” in the multi-picture buffer and the default relative indices of all subsequent pictures in default order are decremented by one.

If memory_management_control_operation indicates the assignment of a long-term index to a specified short-term picture and if the specified long-term index has not already been assigned to the specified short-term picture, the specified short-term picture is marked in the buffer as a long-term picture with the specified long-term index. If another picture is already present in the buffer with the same long-term index as the specified long-term index, the other picture is marked as “unused”. All short-term pictures that were subsequent to the specified short-term picture in default relative index order and all long-term pictures having a long-term index less than the specified long-term index have their associated default relative indices decremented by one. The specified picture is assigned to a default relative index of one plus the highest of the incremented default relative indices, or zero if there are no such incremented indices.

9.2 Motion compensation

The motion compensation process generates predictions for picture blocks using previously decoded reference pictures. The selected reference picture and motion vectors to be used are described in subclauses 7.4.5 and 7.4.6, respectively. If picture_struct indicates a field picture, only the reference field indicated by the ref_idx_fwd or ref_idx_bwd is used in the motion compensation.

9.2.1 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 8.4.6.1.1) is used. In case the macroblock may be classified to have directional segmentation the prediction is defined in subclause 8.4.6.1.2. Motion vector for a Skip mode macroblock shall be obtained as described in subclause 8.4.6.1.3.

9.2.1.1 Median prediction

In the Figure below the vector component E of the indicated block shall be predicted. The prediction is normally formed as the median of A, B and C. However, the prediction may be modified as described below. Notice that it is still referred to as "median prediction"

A	The component applying to the sample to the left of the upper left sample in E
B	The component applying to the sample just above the upper left sample in E
C	The component applying to the sample above and to the right of the upper right sample in E
D	The component applying to the sample above and to the left of the upper left sample in E

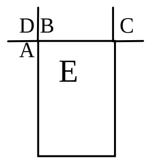


Figure 9-1 -Median prediction of motion vectors

A, B, C, D and E may represent motion vectors from different reference pictures. As an example we may be seeking prediction for a motion vector for E from the last decoded picture. A, B, C and D may represent vectors from 2, 3, 4 and 5 pictures back. The following substitutions may be made prior to median filtering.

- If A and D are outside the picture, their values are assumed to be zero and they are considered to have "different reference picture than E".
- If D, B, C are outside the picture, the prediction is equal to A (equivalent to replacing B and C with A before median filtering).

– If C is outside the picture or still not available due to the order of vector data (see Figure 2), C is replaced by D.

If any of the blocks A, B, C, D are intra coded they count as having "different reference picture". If one and only one of the vector components used in the median calculation (A, B, C) refer to the same reference picture as the vector component E, this one vector component is used to predict E.

9.2.1.2 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 below 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.

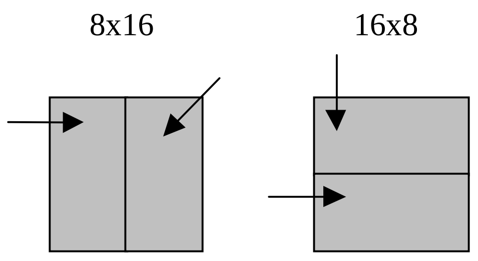


Figure 9-2 – Directional segmentation prediction

9.2.1.3 Motion vector for a Skip mode macroblock

Motion vector for a Skip mode macroblock shall be obtained identically to the prediction motion vector for the 16x16 macroblock mode. 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 8.4.6.1.1) uses the last decoded picture as reference and has zero magnitude.

9.2.1.4 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.

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. 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.

9.2.2 Fractional sample accuracy

Fractional sample accuracy is indicated by motion_resolution. Depending on the value of themotion_resolution**,** either quarter sample interpolation with a 6-tap filter or one eighth sample interpolation with an 8-tap filter is performed for all the luma samples in the block. The prediction process for chroma samples in both cases is described in subclause 9.2.1.3.

9.2.2.1 Quarter sample luma interpolation

The positions labelled 'A' in Figure 8-1 below represent reference picture samples in integer positions. Other symbols represent interpolated values at fractional sample positions.

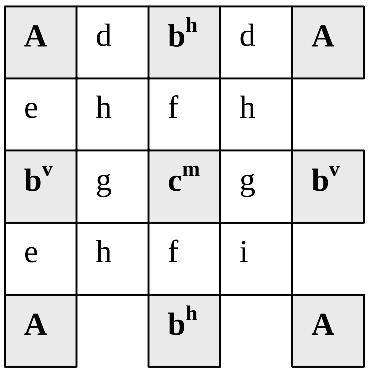


Figure 9-3 – Integer samples ('A') and fractional sample positions for quarter sample luma interpolation

The prediction values at integer positions shall be obtained by using the samples of the reference picture without alteration. The 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 prediction values at quarter sample positions shall be generated by averaging samples at integer and half sample positions. The process for each position is described below.

– The samples at half sample positions labelled as 'b^h' shall be obtained by first calculating intermediate value *b* applying the 6-tap filter to the nearest samples 'A' at integer positions in horizontal direction. The final value shall be calculated using b^h = clip1(((*b*+16)>>5). The samples at half sample positions labelled as 'b^v' shall be obtained equivalently with the filter applied in vertical direction.

– The samples at half sample positions labelled as 'c^m' shall be obtained by applying the 6-tap filter to intermediate values *b* of the closest half sample positions in either vertical or horizontal direction to form an intermediate result *c*. The final value shall be calculated using c^m = clip1(((*c*+512)>>10).

– The samples at quarter sample positions labelled as 'd', 'g', 'e' and 'f' shall be obtained by averaging with truncation the two nearest samples at integer or half sample position using d=(A+b^h)>>1, g=(b^v+c)>>1, e=(A+b^v)>>1, f=(b^h+c^m)>>1.

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

– The samples at quarter sample positions labelled as 'i' shall be computed using the four nearest samples in integer positions using i = (A_1+A_2+A_3+A_4+2)>>2.

9.2.2.2 One eighth sample luma interpolation

The positions labelled 'A' in the Figure 9-2 represent reference picture samples in integer positions. Other symbols represent interpolated values at fractional sample positions.

A	d	$\mathbf{b^h}$	d	$\mathbf{b^h}$	d	$\mathbf{b^h}$	d	**A**
d	e	d	f^h	d	f^h	d	e	
$\mathbf{b^v}$	d	$\mathbf{c^q}$	d	$\mathbf{c^q}$	d	$\mathbf{c^q}$	d	$\mathbf{b^v}$
d	f^v	d	g	d	g	d	f^v	
$\mathbf{b^v}$	d	$\mathbf{c^q}$	d	$\mathbf{c^m}$	d	$\mathbf{c^q}$	d	$\mathbf{b^v}$
d	f^v	d	g	d	g	d	f^v	
$\mathbf{b^v}$	d	$\mathbf{c^q}$	d	$\mathbf{c^q}$	d	$\mathbf{c^q}$	d	$\mathbf{b^v}$
d	e	d	f^h	d	f^h	d	e	
A		$\mathbf{b^h}$		$\mathbf{b^h}$		$\mathbf{b^h}$		**A**

Figure 9-4 – 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:

– 1/4 position: (-3, 12, -37, 229, 71, -21, 6, -1)

– 2/4 position: (-3, 12, -39, 158, 158, -39, 12, -3)

– 3/4 position: (-1, 6, -21, 71, 229, -37, 12, -3)

The samples at one eighth sample positions are defined 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 labelled as 'b^h' shall be obtained by first calculating an intermediate value *b* applying 8-tap filtering to the nearest samples 'A' at integer positions in horizontal direction. 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.

– 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 a value *c*, and then the final result shall be obtained using c^m = clip1(((c+32768)>>16) or c^q = clip1(((c+32768)>>16).

– The samples at one eighth sample positions labelled as 'd' shall be obtained by averaging with truncation the two closest 'A', 'b' or 'c' samples using d = (A+b^h)>>1, d = (b^h_0+b^h_1)>>1, d = (A+b^v)>>1, d = (b^h+c^q)>>1, d = (b^v+c^q)>>1, d = (c^q_0+c^q_1)>>1, d = (b^v_0+b^v_1)>>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^c' samples using g = (A+3c^c+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.

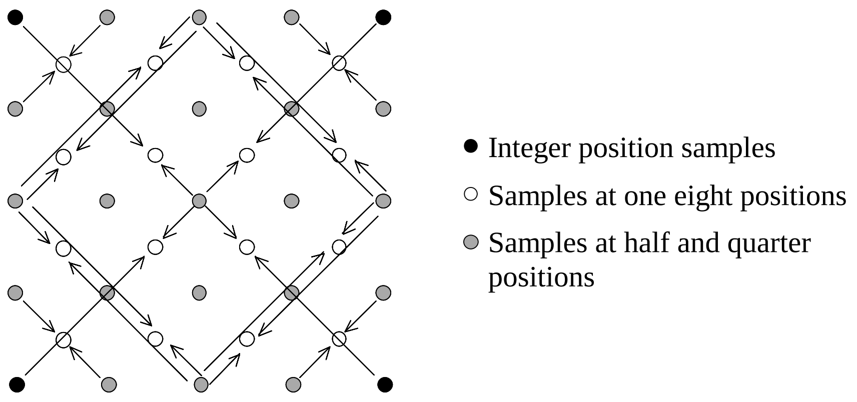

● Integer position samples
○ Samples at one eight positions
● Samples at half and quarter positions

Figure 9-5 – Diagonal interpolation for one eighth sample luma interpolation

9.2.2.3 Chroma interpolation

Motion compensated prediction fractional chroma samples shall be obtained using Equation 9-1.

$$v = ((s - d^x)(s - d^y)A + d^x(s - d^y)B + (s - d^x)d^yC + d^xd^yD + s^2/2)/s^2 \tag{9-1}$$

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 is 16 for one eighth sample interpolation. The relationships between the variables in Equation 9-1 and reference picture positions are illustrated in Figure 9-4.

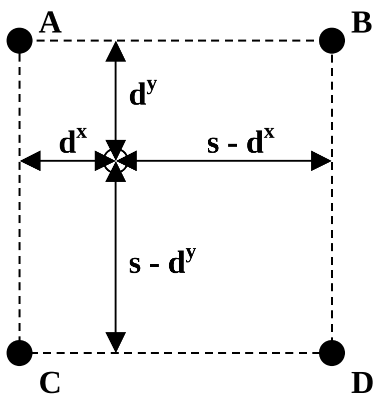


Figure 9-6 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D.

9.3 Intra Prediction

Two Intra coding modes for macroblocks are described below.

9.3.1 Intra Prediction for 4x4 mode for luma

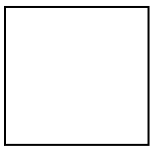


Figure 9-7 – Identification of samples used for intra spatial prediction

For the luma signal, there are nine intra prediction modes labeled 0 to 8. Mode 0 is 'DC-prediction' (see below). The other modes represent directions of predictions as indicated below.

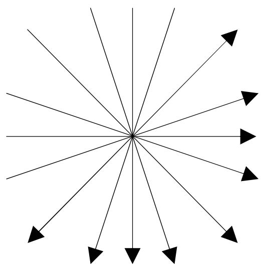


Figure 9-8 – Intra prediction directions

9.3.1.1 Mode 0: DC prediction

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

9.3.1.2 Mode 1: Vertical Prediction

If A, B, C, D are inside the slice, then

– 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.

9.3.1.3 Mode 2: Horizontal prediction

If I, J, K, L are inside the slice, then

– 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.

9.3.1.4 Mode 3: Diagonal Down/Right prediction

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

– 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

9.3.1.5 Mode 4: Diagonal Down/Left prediction

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

– 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) >> 3

9.3.1.6 Mode 5: Vertical-Left prediction

This mode is 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

9.3.1.7 Mode 6: Vertical-Right prediction

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

– 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

9.3.1.8 Mode 7: Horizontal-Up prediction

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

– 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	(D + 2E + F + 2J + 2K + 4) >> 3
– d, f are predicted by	(E + 2F + G + J + 2K + L + 4) >> 3
– g, i are predicted by	(F + 2G + H + 2K + 2L + 4) >> 3
– h, j are predicted by	(G + 3H + K + 3L + 4) >> 3
– l, n are predicted by	(L + 2M + N + 2) >> 3
– k, m are predicted by	(G + H + L + M + 2) >> 2
– o is predicted by	(M + N + 1) >> 1
– p is predicted by	(M + 2N + O + 2) >> 2

9.3.1.9 Mode 8: Horizontal-Down prediction

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

– 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

9.3.2 Intra prediction for 16x16 mode for luma

Assume that the block to be predicted has sample locations 0 to 15 horizontally and 0 to 15 vertically. We use the notation P(i,j) where i,j = 0..15. P(i,-1), i=0..15 are the neighbouring samples above the block and P(-1,j), j=0..15 are the neighbouring samples to the left of the block. Pred(i,j) i,j = 0..15 is the prediction for the whole luma macroblock. We have 4 different prediction modes:

9.3.2.1 Mode 0 (vertical)

Pred(i, j) = P(i, -1), i, j=0..15

9.3.2.2 Mode 1 (horizontal)

Pred(i, j) = P(-1, j), i, j=0..15

9.3.2.3 Mode 2 (DC prediction)

Pred(i, j) = $((\sum_{i=0}^{15}(P(-1,i)+P(i,-1)))+16) >> 5$ i, j=0..15,

where only the average of 16 samples are used when the other 16 samples are outside the slice. If all 32 samples are outside the slice, the prediction for all samples in the block is 128.

9.3.2.4 Mode 3 (Plane prediction)

Pred(i,j) = clip1((a + b·(i-7) + c·(j-7) +16) >> 5),

where:

- a = 16·(P(-1,15) + P(15,-1))
- b = (5*H+32)>>6
- c = (5*V+32)>>6

and H and V are defined as:

$$H = \sum_{i=1}^{8} i \cdot (P(7+i,-1) - P(7-i,-1)) \qquad (8\text{-}1)$$

$$V = \sum_{j=1}^{8} j \cdot (P(-1,7+j) - P(-1,7-j)) \qquad (8\text{-}2)$$

Residual coding

The residual coding is based on 4x4 transform. But similar to coding of chroma coefficients, another 4x4 transform to the 16 DC coefficients in the macroblock are added. In that way we end up with an overall DC for the whole MB which works well in flat areas.

Since we use the same integer transform to DC coefficients, we have to perform additional normalization to those coefficients, which implies a division by 676. To avoid the division we performed normalization by 49/215 on the encoder side and 48/215 on the decoder side, which gives sufficient accuracy.

Only single scan is used for 16x16 intra coding.

To produce the bitstream, we first scan through the 16 'DC transform' coefficients. There is no 'cbp' information to indicate no coefficients on this level. If AC = 1 (see below) ac coefficients of the 16 4x4 blocks are scanned. There are 15 coefficients in each block since the DC coefficients are included in the level above.

9.3.3 Prediction in intra coding of chroma blocks

For chroma prediction there is only one mode. No information is therefore needed to be transmitted. The prediction is indicated in Figure 9-9. The 8x8 chroma block consists of 4 4x4 blocks A,B,C,D. S0,1,2,3 are the sums of 4 neighbouring samples.

If S0, S1, S2, S3 are all inside the frame:

– A = (S0 + S2 + 4)>>3

– B = (S1 + 2)>>2

– C = (S3 + 2)>>2

– D = (S1 + S3 + 4)>>3

If only S0 and S1 are inside the frame:

– A = (S0 + 2)>>2

– B = (S1 + 2)>>2

– C = (S0 + 2)>>2

– D = (S1 + 2)>>2

If only S2 and S3 are inside the frame:

– A = (S2 + 2)>>2

– B = (S2 + 2)>>2

– C = (S3 + 2)>>2

– D = (S3 + 2)>>2

If S0, S1, S2, S3 are all outside the frame: A = B = C = D = 128

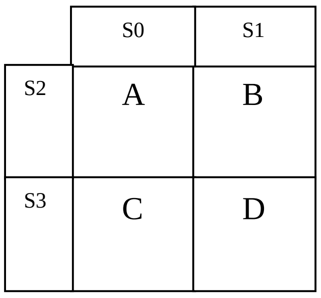


Figure 9-9 – Prediction of chroma blocks

9.4 Transform Coefficient Decoding

This subclause defines aspects related to transform coefficient decoding.

9.4.1 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:

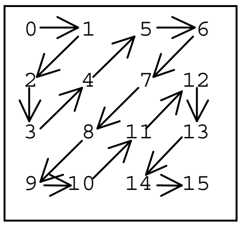


Figure 9-10 – Zig-zag scan

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.

9.4.3 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 defined 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 9-1:

Table 9-1 – 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(m,i,j), used in the formulas below, are defined in pseudo code by:

R[m][i][j] = $S_{m,0}$ for (i,j) = {(0,0),(0,2),(2,0),(2,2)},

R[m][i][j] = $S_{m,1}$ for (i,j) = {(1,1),(1,3),(3,1),(3,3)},

R[m][i][j] = $S_{m,2}$ otherwise;

where the first and second subscripts of *S* are row and column indices, respectively, of the matrix defined as:

$$S = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix} \qquad (9\text{-}2)$$

9.4.3.1 Luma DC coefficients in Intra 16x16 macroblock

After decoding the scaled coefficient indices for a 4x4 block of luma DC coefficients coded in 16x16 intra mode, the 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 defined by:

$$X_{QD} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} y_{QD00} & y_{QD01} & y_{QD02} & y_{QD03} \\ y_{QD10} & y_{QD11} & y_{QD12} & y_{QD13} \\ y_{QD20} & y_{QD21} & y_{QD22} & y_{QD23} \\ y_{QD30} & y_{QD31} & y_{QD32} & y_{QD33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \qquad (9\text{-}3)$$

A bitstream conforming to this Specification shall not contain indicated data that results in a value of $X_{QD}(i,\mathrm{j})$ 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 zero, then the scaled result shall be calculated as

$$X_D(i,j) = [X_{QD}(i,j) \cdot R(QP\%6,0,0)] << (QP/6-2), \quad i,j = 0,...,3 \qquad (9\text{-}4)$$

b) If QP is less than zero, then the scaled results shall be calculated as

$$X_D(i,j) = [X_{QD}(i,j) \cdot R(QP\%6,0,0) + 2^{1-QP/6}] >> (2-QP/6). \qquad (9\text{-}5)$$

A bitstream conforming to this Specification shall not contain data that results in a value of $X_D(i,j)$ that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

9.4.3.2 Chroma DC coefficients

After decoding the scaled coefficient indices for a 2x2 block of chroma DC coefficients, the transform process is applied before the scaling process.

Definition of transform:

$$X_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} i_{00} & i_{01} \\ i_{10} & i_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (9\text{-}6)$$

A bitstream conforming to this Specification shall not contain indicated data that results in a value of $X_{QD}(i,j)$ 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

$$X_D(i,j) = [X_{QD}(i,j) \cdot R(QP\%6,0,0)] << (QP/6-1), \quad i,j = 0,...,3 \tag{9-7}$$

b) If QP is less than 6, then the scaling results shall be calculated by

$$X_D(i,j) = X_{QD}(i,j) \cdot R(QP\%6,0,0) >> 1, \quad i,j = 0,...,3 \tag{9-8}$$

A bitstream conforming to this Specification shall not contain indicated data that results in a value of $X_D(i,j)$ that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

9.4.3.3 Residual 4x4 blocks

Scaling of coefficients other than those as specified in subclauses 9.4.2.1 and 9.4.2.2 is performed according to the following equation:

$$Y'(i,j) = Y_Q(i,j) \cdot R(QP\%6,i,j) << (QP/6), \quad i,j = 0,...,3 \tag{9-9}$$

where the R(m,i,j) are the scaling coefficients listed below.

A bitstream conforming to this Specification shall not contain indicated data that results in a value of $Y'(i,j)$ that exceeds the range of integer values from -2^{15} to $2^{15}-1$, inclusive.

After constructing the entire 4x4 block of reconstructed transform coefficients expressed in matrix form as

$$Y' = \begin{bmatrix} y'_{00} & y'_{01} & y'_{02} & y'_{03} \\ y'_{10} & y'_{11} & y'_{12} & y'_{13} \\ y'_{20} & y'_{21} & y'_{22} & y'_{23} \\ y'_{30} & y'_{31} & y'_{32} & y'_{33} \end{bmatrix}, \tag{9-10}$$

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:

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 defined as follows for four input samples y_0, y_1, y_2, y_3.

a) First, a set of intermediate values is computed:

$$z_0 = y_0 + y_2 \tag{9-11}$$

$$z_1 = y_0 - y_2 \tag{9-12}$$

$$z_2 = (y_1 >> 1) - y_3 \tag{9-13}$$

$$z_3 = y_1 + (y_3 >> 1) \tag{9-14}$$

b) Then the transformed result is computed from these intermediate values

$$x_0 = z_0 + z_3 \tag{9-15}$$

$$x_1 = z_1 + z_2 \tag{9-16}$$

$$x_2 = z_1 - z_2 \tag{9-17}$$

$$x_3 = z_0 - z_3 \tag{9-18}$$

A bitstream conforming to this Specification 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 Specification 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,

$$X' = \begin{bmatrix} x'_{00} & x'_{01} & x'_{02} & x'_{03} \\ x'_{10} & x'_{11} & x'_{12} & x'_{13} \\ x'_{20} & x'_{21} & x'_{22} & x'_{23} \\ x'_{30} & x'_{31} & x'_{32} & x'_{33} \end{bmatrix} \tag{9-19}$$

the final reconstructed sample residual values shall be obtained as

$$X''(i,j) = [X'(i,j) + 2^5] >> 6 \tag{9-20}$$

Finally, the reconstructed sample residual values $X''(i,j)$ are added to the prediction values $P(i,j)$ from motion compensated prediction or spatial prediction and clipped to the range of 0 to 255 to form the final decoded sample result:

$$S'(i,j) = \text{clip1}(\ P(i,j) + X''(i,j)\ ) \tag{9-21}$$

9.5 Deblocking Filter

A conditional filtering shall be applied to all reconstructed macroblocks of a picture. This filtering is done on a macroblock basis. As the first step, the 16 pels of each of the 3 vertical edges internal to the macroblock of the 4x4 raster shall be filtered as shown on the left side of Figure 9-11 (horizontal filtering). Filtering of the 3 horizontal edges internal to the macroblock follows (vertical filtering). Next the left edge of the macroblock is filtered, followed by the top edge after the corresponding macroblocks to the left and top of the current macroblock are reconstructed. Picture edges are not filtered.

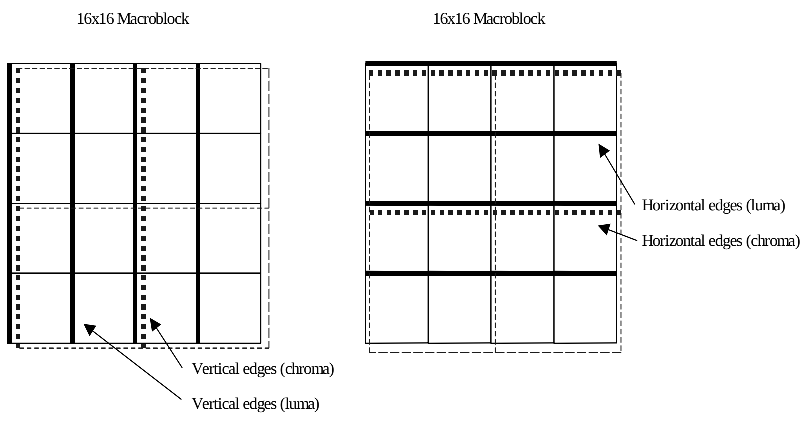


Figure 9-11 – Boundaries in a macroblock to be filtered (luma boundaries shown with solid lines and chroma boundaries shown with dotted lines)

Note 1: The intra prediction of a macroblock takes place based on 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.

Note 2: When picture_struct indicates a field picture, then all calculations for the deblocking filter are based solely on within the current field.

9.5.1 Content dependent boundary filtering strength

For each boundary between neighbouring 4x4 luma blocks, a "**Boundary Strength" Bs** is assigned as shown in Figure 9-8 that influences the strength of filtering for this particular piece of boundary. As indicated in Figure 9-7, every block boundary of a chroma block corresponds to a specific boundary of a luma block. Bs values for chroma are not calculated, but simply copied from the corresponding luma Bs.

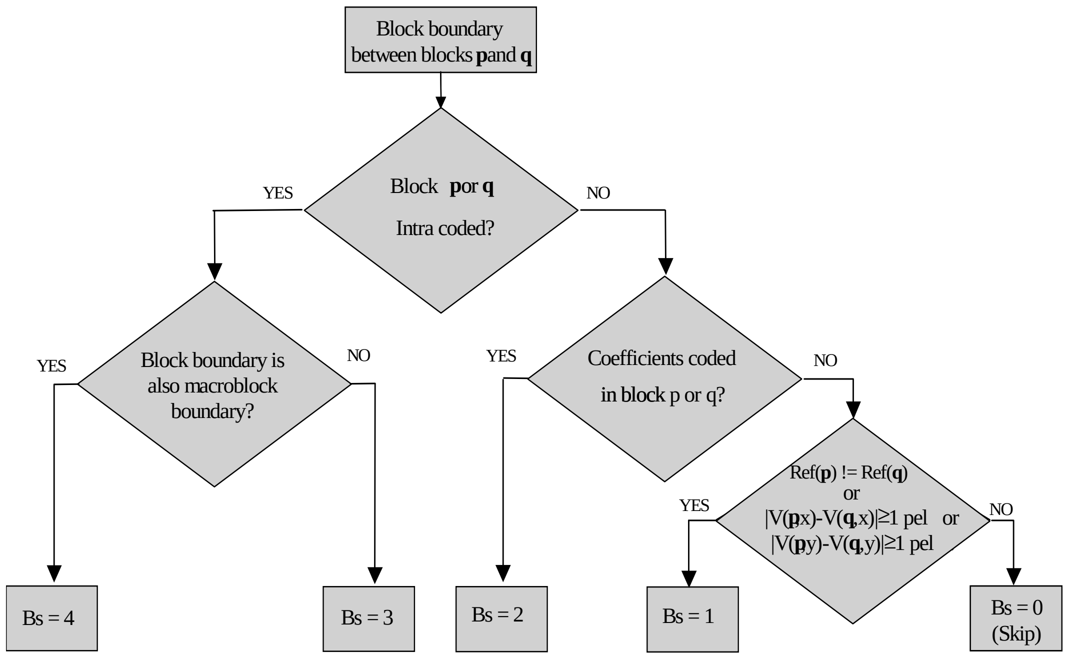

Block boundary between blocks p and q
Block p or q Intra coded?
YES
NO
Block boundary is also macroblock boundary?
YES
NO
Coefficients coded in block p or q?
YES
NO
Ref(p) != Ref(q) or |V(p,x)-V(q,x)|≥1 pel or |V(p,y)-V(q,y)|≥1 pel
YES
NO
Bs = 4
Bs = 3
Bs = 2
Bs = 1
Bs = 0 (Skip)

Figure 9-12 – Flow chart for determining the boundary strength (Bs), for the block boundary between two neighbouring blocks p and q, where Ref(p) is the reference frame or field of block p and V(p) is the motion vector of block p

9.5.2 Thresholds for each block boundary

p_3	p_2	p_1	p_0	q_0	q_1	q_2	q_3

Figure 9-13 – Convention for describing samples across 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 9-9 with the actual boundary lying between p_0 and q_0. In the default mode, up to two samples on both sides of the boundary can be updated as a result of the filtering process (that is at most p_1, p_0, q_0, q_1). Filtering across a certain 4x4 block boundary is skipped altogether if the corresponding Bs is equal to zero. Sets of samples across this edge are only filtered if the condition

$$Bs \neq 0 \;\; \&\& \;\; |p_0 - q_0| < \alpha \;\; \&\& \;\; |p_1 - p_0| < \beta \;\; \&\& \;\; |q_1 - q_0| < \beta \qquad (9\text{-}26)$$

is true. The values of the thresholds α and β are dependent on the average value of QP for the two macroblocks, as determined by $QP_{av} = (QP_p+QP_q)>>1$ is used to determine α and β. The values for the thresholds are shown in Table 9-1.

Table 9-1 – QP_{av} dependent threshold parameters α and β

	QP_{av}																									
	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	0	0	0	0	4	4	5	6	7	9
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	3	4	4	4

Table 9-1 (concluded)

	QP$_{av}$																									
	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	10	12	14	17	20	24	28	33	39	46	55	65	76	90	106	126	148	175	207	245	255	255	255	255	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

9.5.3 Filtering of edges with Bs < 4

Two types of filtering are defined. In the default case the equations below are used to filter p_0 and q_0. Here, uppercase letters indicate filtered 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.

$$\Delta = \text{clip3}(-C, C, (((q_0 - p_0) << 2 + (p_1 - q_1) + 4) >> 3)) \quad (9\text{-}27)$$

$$P_0 = \text{clip1}(p_0 + \Delta) \quad (9\text{-}28)$$

$$Q_0 = \text{clip1}(q_0 - \Delta) \quad (929)$$

where C is determined as specified below.

Two intermediate threshold variables

$$a_p = |p_2 - p_0| \quad (9\text{-}30)$$

$$a_q = |q_2 - q_0| \quad (9\text{-}31)$$

shall be used to determine whether luma samples p_1 and q_1 are filtered. These samples are only processed for luma.

If $a_p < \beta$ for a luma edge, a filtered sample P_1 shall be produced as specified by

$$P_1 = p_1 + \text{clip3}(-C0, C0, (p_2 + (p_0 + q_0)>>1 - 2*p_1) >> 1) \quad (9\text{-}32)$$

If $a_q < \beta$ for a luma edge, a filtered sample Q_1 shall be produced as specified by

$$Q_1 = q_1 + \text{clip3}(-C0, C0, (q_2 + (p_0 + q_0)>>1 - 2*q_1) >> 1) \quad (9\text{-}33)$$

where C0 is specified in Table 9-2.

C is determined by setting C equal to C0 and then incrementing C by one if $a_p<\beta$, and again by one if $a_q<\beta$.

Table 9-2 – Value of filter clipping parameter C as a function of QP_{av} and Bs

	QP_{av}																									
	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 or 4	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 9-2 (concluded)

	QP_{av}																									
	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 or 4	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

9.5.4 Filtering of edges with Bs = 4

When Bs is equal to 4, if the following condition holds:

$$a_p < \beta \qquad (9\text{-}34)$$

filtering of the left/upper side of the block edge is defined by the equations:

$$P_0 = (\, p_2 + 2{*}p_1 + 2{*}p_0 + 2{*}q_0 + q_1 + ditherP[pos]) >> 3 \qquad (9\text{-}35)$$

$$P_1 = (\, p_3 + 2{*}p_2 + 2{*}p_1 + 2{*}p_0 + q_0 + ditherP[pos]) >> 3 \qquad (9\text{-}36)$$

and in the case of luma filtering only:

$$P_2 = (\, 2{*}p_3 + 3{*}p_2 + p_1 + p_0 + q_0 + ditherP[pos]) >> 3 \qquad (9\text{-}38)$$

Otherwise, if the condition of 9-34 does not hold, the following filter is applied:

$$P_0 = (\, 2{*}p_1 + p_0 + q_1 + 2) >> 2 \qquad (9\text{-}39)$$

Similarly, for filtering of the right/lower side of the edge, if the following condition holds:

$$a_q < \beta \qquad (9\text{-}40)$$

filtering is defined by the equations:

$$Q_0 = (\, p_1 + 2{*}p_0 + 2{*}q_0 + 2{*}q_1 + q_2 + ditherQ[pos]) >> 3 \qquad (9\text{-}41)$$

$$Q_1 = (\, p_0 + 2{*}q_0 + 2{*}q_1 + 2{*}q_2 + q_3 + ditherQ[pos]) >> 3 \qquad (9\text{-}42)$$

and, in the case of luma filtering only:

$$Q_2 = (\, 2{*}q_3 + 3{*}q_2 + q_1 + q_0 + p_0 + ditherQ[pos]) >> 3 \qquad (9\text{-}43)$$

Otherwise, if the condition of 9-39 does not hold, the following filter is applied:

$$Q_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \tag{9-44}$$

Where *ditherP* and *ditherQ* are taken from Table 9-3 and *pos* is the distance from the top line of the macroblock when filtering vertical edges (horizontal filtering) and the distance from the left column of the macroblock when filtering horizontal edges (vertical filtering).

Table 9-3 – Rounding Value for Strong Filter

pos	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ditherP[pos]	**4**	**5**	**3**	**6**	**2**	**7**	**1**	**5**	**3**	**1**	**7**	**2**	**6**	**3**	**5**	**4**
ditherQ[pos]	**4**	**3**	**5**	**2**	**6**	**1**	**7**	**3**	**5**	**7**	**1**	**6**	**2**	**5**	**3**	**4**

NOTE – Q_1(block$_n$) might be used and be overwritten as P_2 during the calculation of block$_{n+1}$. The same is true for Q_2(block$_n$) which might become P_1(block$_{n+1}$).

9.6 Entropy Coding

9.6.1 Default Entropy Coding

In the default entropy coding mode, one VLC is used to code the syntax elements. The table of codewords are 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. We will sometimes refer to a codeword with 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_number = 2^L/2 + INFO -1 (L/2 use division with truncation. INFO = 0 when L = 1) Some of the first code numbers and codewords are written explicitly in Table 9-3. As an example, for the code number 5, L = 5 and INFO = 10 (binary) = 2 (decimal).

Table 9-3 – UVLC code number and codewords in explicit form

Code_number	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`
10	`0 0 0 1 0 1 1`
......	`. . . . . . .`

When L (L = 2N-1) and INFO is known, the regular structure of the table makes it easy to create a codeword. Similarly, a decoder may easily decode a codeword by reading in N bit prefix followed by N-1 INFO. L and INFO is then readily available. For each parameter to be coded, there is a conversion rule from the parameter value to the code number (or L and INFO). Table 9-4 lists the connection between code number and most of the parameters used in the present coding method.

Table 9-4 – Connection between codeword number and parameter values

Code_ number	RUN	MB_Type		8x8 mode	mvd_fwd delta_qp	cbp		Tcoeff_chroma_DC[1]		Tcoeff_chroma_AC[1] Tcoeff_luma[1] Simple scan		Tcoeff_luma[1] Double scan	
		Intra	Inter			Intra	Inter	Level	Run	Level	Run	Level	Run
0	0	Intra4x4	16x16	8x8	0	47	0	EOB	-	EOB	-	EOB	-
1	1	0,0,0[2]	16x8	8x4	1	31	16	1	0	1	0	1	0
2	2	1,0,0	8x16	4x8	-1	15	1	-1	0	-1	0	-1	0
3	3	2,0,0	8x8	4x4	2	0	2	2	0	1	1	1	1
4	4	3,0,0	8x8 (ref=0)	Intra	-2	23	4	-2	0	-1	1	-1	1
5	5	0,1,0	Intra4x4		3	27	8	1	1	1	2	2	0
6	6	1,1,0	0,0,0[2]		-3	29	32	-1	1	-1	2	-2	0
7	7	2,1,0	1,0,0		4	30	3	3	0	2	0	1	2
8	8	3,1,0	2,0,0		-4	7	5	-3	0	-2	0	-1	2
9	9	0,2,0	3,0,0		5	11	10	2	1	1	3	3	0
10	10	1,2,0	0,1,0		-5	13	12	-2	1	-1	3	-3	0
11	11	2,2,0	1,1,0		6	14	15	1	2	1	4	4	0
12	12	3,2,0	2,1,0		-6	39	47	-1	2	-1	4	-4	0
13	13	0,0,1	3,1,0		7	43	7	1	3	1	5	5	0
14	14	1,0,1	0,2,0		-7	45	11	-1	3	-1	5	-5	0
15	15	2,0,1	1,2,0		8	46	13	4	0	3	0	1	3
16	16	3,0,1	2,2,0		-8	16	14	-4	0	-3	0	-1	3
17	17	0,1,1	3,2,0		9	3	6	3	1	2	1	1	4
18	18	1,1,1	0,0,1		-9	5	9	-3	1	-2	1	-1	4
19	19	2,1,1	1,0,1		10	10	31	2	2	2	2	2	1
20	20	3,1,1	2,0,1		-10	12	35	-2	2	-2	2	-2	1
21	21	0,2,1	3,0,1		11	19	37	2	3	1	6	3	1
22	22	1,2,1	0,1,1		-11	21	42	-2	3	-1	6	-3	1

23	23	2,2,1	1,1,1		12	26	44	5	0	1	7	6	0
24	24	3,2,1	2,1,1		-12	28	33	-5	0	-1	7	-6	0
25	25		3,1,1		13	35	34	4	1	1	8	7	0
26	26		0,2,1		-13	37	36	-4	1	-1	8	-7	0
27	27		1,2,1		14	42	40	3	2	1	9	8	0
28	28		2,2,1		-14	44	39	-3	2	-1	9	-8	0
29	29		3,2,1		15	1	43	3	3	4	0	9	0
30	30				-15	2	45	-3	3	-4	0	-9	0
31	31				16	4	46	6	0	5	0	10	0
32	32				-16	8	17	-6	0	-5	0	-10	0
33	33				17	17	18	5	1	3	1	4	1
34	34				-17	18	20	-5	1	-3	1	-4	1
35	35				18	20	24	4	2	3	2	2	2
36	36				-18	24	19	-4	2	-3	2	-2	2
37	37				19	6	21	4	3	2	3	2	3
38	38				-19	9	26	-4	3	-2	3	-2	3
39	39				20	22	28	7	0	2	4	2	4
40	40				-20	25	23	-7	0	-2	4	-2	4
41	41				21	32	27	6	1	2	5	2	5
42	42				-21	33	29	-6	1	-2	5	-2	5
43	43				22	34	30	5	2	2	6	2	6
44	44				-22	36	22	-5	2	-2	6	-2	6
45	45				23	40	25	5	3	2	7	2	7
46	46				-23	38	38	-5	3	-2	7	-2	7
47	47				24	41	41	8	0	2	8	11	0
	..				..			..	..	..	..	..	..

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).

For 16x16 based intra mode. The 3 numbers refer to values for (Imode,AC,nc) - see qq.

Table 9-5 – Connection between codeword number and Intra Prediction Mode Probability

Code_ number	Prob0, Prob1[3]	Code_ number	Prob0, Prob1[3]	Code_ number	Prob0, Prob1[3]	Code_ number	Prob0, Prob1[3]
0	0,0	21	2,3	41	2,6	61	6,5
1	0,1	22	3,2	42	6,2	62	4,7
2	1,0	23	1,5	43	3,5	63	7,4

3	1,1	24	5,1	44	5,3	64	3,8
4	0,2	25	2,4	45	1,8	65	8,3
5	2,0	26	4,2	46	8,1	66	4,8
6	0,3	27	3,3	47	2,7	67	8,4
7	3,0	28	0,7	48	7,2	68	5,7
8	1,2	29	7,0	49	4,5	69	7,5
9	2,1	30	1,6	50	5,4	70	6,6
10	0,4	31	6,1	51	3,6	71	6,7
11	4,0	32	2,5	52	6,3	72	6,7
12	3,1	33	5,2	53	2,8	73	5,8
13	1,3	34	3,4	54	8,2	74	8,5
14	0,5	35	4,3	55	4,6	75	6,8
15	5,0	36	0,8	56	6,4	76	8,6
16	2,2	37	8,0	57	5,5	77	7,7
17	1,4	38	1,7	58	3,7	78	7,8
18	4,1	39	7,1	59	7,3	79	8,7
19	0,6	40	4,4	60	5,6	80	8,8
20	6,0						

Prob0 and Prob1 defines the Intra prediction modes of two blocks relative to the prediction of prediction modes (see details in the section for Intra coding).

9.6.2 Entropy Coding of Transform Coefficients

CAVLC (Contex-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. A common parameter Num-Trail is used that contains the number of coefficients as well as the number of "Trailing 1s" (from now referred to as T1s). For T1s only the sign has to be decoded.
2. For coefficients other than the T1s, Level information is decoded.
3. Lastly, the Run information is decoded. Since the number of coefficients is already known, this limits possible values for Run. Run is split into Total number of zeros before all coefficients and Run before each non-zero coefficient.

Zig-zag scanning as described in Section 9.4.1 is used, but in the decoding of coefficient data, both levels and runs, the scanning is done in reverse order. Therefore, in the Level information, the signs of T1s are decoded first (in reverse order), then the Level information of the last coefficient where this is needed, 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, and so on.

9.6.2.1 Num-Trail

Three luma and one chroma DC 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.

Table 9-6: Number of coefficients / Trailing Ones: Num-VLC0

NumCoef\T1s	0	1	2	3
0	1	-	-	-
1	000011	01	-	-
2	00000111	0001001	001	-
3	000001001	00000110	0001000	00011
4	000001000	000001011	000000101	000010
5	0000000111	000001010	000000100	0001011
6	00000000111	0000000110	0000001101	00010101
7	000000001001	00000000110	0000001100	00010100
8	000000001000	00000001001	000000001010	000000111
9	0000000000111	000000001011	000000000101	0000000101
10	0000000000110	0000000001101	0000000001111	00000001000
11	00000000000011	0000000001100	0000000001110	000000000100
12	00000000000010	00000000000100	00000000000110	0000000000101
13	00000000000101	00000000000111	000000000010001	00000000001001
14	000000000000011	000000000000010	000000000010000	0000000000000011
15	00000000000000001	00000000000000011	00000000000000010	00000000000000101
16	00000000000000000	000000000000001001	0000000000000010001	0000000000000010000

Table 9-7: Number of coefficients / Trailing Ones: Num-VLC1

NumCoeff\T1s	0	1	2	3
0	11	-	-	-
1	000011	011	-	-
2	000010	00011	010	-
3	001001	001000	001010	101
4	1000001	001011	100101	0011
5	00000111	1000000	1000010	00010
6	00000110	1000011	1001101	10001
7	000001001	10011101	10011100	100100
8	000001000	000001011	000000101	1001100
9	0000000111	000001010	000000100	10011111
10	0000000110	0000001101	0000001100	10011110
11	00000000101	00000000111	00000001001	000000111
12	00000000100	00000000110	00000001000	0000000101
13	000000000011	000000000010	000000000100	000000000111
14	0000000000011	000000000101	0000000000010	0000000001101
15	00000000000001	00000000000000	000000000000111	0000000001100
16	000000000000101	000000000000100	0000000000001101	0000000000001100

Table 9-8: Number of coefficients / Trailing Ones: Num-VLC2

NumCoeff\T1s	**0**	**1**	**2**	**3**
0	0011	-	-	-
1	0000011	0010	-	-
2	0000010	101110	1101	-
3	000011	101001	010110	1100
4	000010	101000	010001	1111
5	101101	101011	010000	1110
6	101100	101010	010011	1001
7	101111	010101	010010	1000
8	0110101	010100	011101	00011
9	0110100	010111	011100	00010
10	0110111	0110110	0110000	011111
11	01111001	0110001	01111010	0110011
12	01111000	01111011	01100101	01100100
13	000000011	000000010	000000100	000000111
14	0000000011	000000101	0000001101	0000001100
15	0000000010	00000000011	00000000010	00000000001
16	0000000000001	000000000001	00000000000001	00000000000000

Table 9-9: Number of coefficients / Trailing Ones: Num-VLC_Chroma_DC

NumCoeff\T1s	**0**	**1**	**2**	**3**
1	0001	1	-	-
2	00001	00111	01	-
3	00110	000001	001010	00100
4	0000001	00000000	00000001	001011

9.6.2.1.1 Table Selection

For all elements, except chroma DC, a choice between three tables and one FLC is made. Selection is done as follows: N is calculated based on the number of coefficients in 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

Table 9-10: 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

5 <= N < 8 : Num-VLC2

N >= 8 : 6 bit FLC xxxxyy, as follows:

NumCoeff–1 is transmitted in the first 4 bits (xxxx). The last 2 bits (yy) are used for T1. There is one exception: the codeword 000011 represents NumCoeff=0.

For chroma DC, Num-VLC_Chroma_DC is used.

9.6.2.2 Decoding of Level information

First, the sign of T1s are decoded from 1 bit each. For the remaining level information, four structured VLCs are used to decode levels. The structured level tables are explained in Table 9-11.

Table 9-11: Level tables

Lev-VLC0

Code no	Code	Level (±1, ±2..)	Level (±2, ±3..)
0	1	1	2
1	01	-1	-2
2	001	2	3
3	000`	-2	-3
..	..	..	..
13	00000000000001	-7	-8
14-29	000000000000001xxxx	±8 to ±15	±9 to ±16
30->	0000000000000001xxxxxxxxxxxx	±16 ->	±17 ->

Lev-VLC1

Code no	Code	Level (±1, ±2..)	Level (±2, ±3..)
0-1	1x	±1	±2
2-3	01x	±2	±3
..	..	..	..
26-27	00000000000001x	±14	±15
28-43	000000000000001xxxx	±15 to ±22	±16 to ±23
44 ->	0000000000000001xxxxxxxxxxxx	±23 ->	±24 ->

Lev-VLC2

Code no	Code	Level (±1, ±2..)
0-3	1xx	±1 to ±2
4-7	01xx	±3 to ±4
..	..	..
52-55	00000000000001xx	±27 to ±28
56-71	000000000000001xxxx	±29 to ±36
72 ->	0000000000000001xxxxxxxxxxxx	±37 ->

Lev-VLC3

Code no	Code	Level (±1, ±2..)
0-7	1xxx	±1 to ±4
8-16	01xxx	±5 to ±8

..	..	..
104-111	00000000000001xxx	±53 to ±56
112-127	000000000000001xxxx	±57 to ±64
128 ->	0000000000000001xxxxxxxxxxxx	±66 ->

Lev-VLC4

Code no	Code no	Code no
0-15	1xxxx	±1 to ±8
16-31	01xxxx	±9 to ±16
..	..	..
112-127	000000000000001xxxx	±57 to ±64
128 ->	0000000000000001xxxxxxxxxxxx	±66 ->

Normally levels to be coded take values ± 1, ± 2 etc. However, for the first coefficient to be decoded (after T1s) and if T1s < 3 or Number_of_coefficients = T1s, levels to be decoded take values ± 2, ± 3 etc (Level'). Since the first coefficient is always decoded with Lev-VLC0 or Lev-VLC1, these levels starting at ± 2 are only applicable to Lev-VLC0 and Lev-VLC1.

Levels are assigned according to ascending code numbers. Positive values receive the lowest code number and negative values receive the next code number.

The last two entries in each table are escape codes. The first escape code, with four "x"'s, is used to decode the 8 levels above the last regularly coded level. The next escape code, with 12 "x"'s, is used to decode all remaining levels.

9.6.2.2.1 Table Selection

The tables are changed along with the decoding process based on QP value, number of coefficients, and the size of the previously decoded level value.

After each Level is decoded, the VLC number is updated according to the following method, where Level is the absolute value of the previously decoded level.

Inter and intra with QP >= 21

First coefficient with VLC0. Next VLC1.

Increase VLC by one (up to 2) if |Level|> 3

Intra with QP < 21

```
if (number of coefficients > 10)
        First coefficient with VLC1.  Next VLC2.
else
        First coefficient with VLC0.  Next VLC1.
if (vlc == VLC1) change to VLC2 if |Level|> 3.
if (vlc >=VLC2) increase vlc by one (up to 4) if |Level| > 5
```

The same procedure is used for chroma AC and DC coefficient levels.

9.6.2.3 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) and Run (of zeros) before each coefficient.

9.6.2.3.1 TotalZeros

The parameter TotalZeros is the sum of all zeros located before the last non-zero coefficient in a forward 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 NumCoeff is already known, it determines the maximum possible value of TotalZeros. One out of 15 VLC tables is chosen based on NumCoeff.

If NumCoeff indicates that all coefficients are non-zero, TotalZeros is not decoded since it is known to be zero

Table 9-12: TotalZeros tables for all 4x4 blocks

NumCoeff TotalZeros	**1**	**2**	**3**	**4**	**5**	**6**	**7**	
1	1	111	0010	111101	01000	101100	111000	
2	011	101	1101	1110	01010	101101	111001	
3	010	011	000	0110	01011	1010	11101	
4	0011	001	010	1010	1110	001	1001	
5	0010	000	1011	000	011	010	1111	
6	00011	1000	1111	100	100	000	00	
7	00010	0101	011	110	1111	110	01	
8	000011	1001	100	1011	110	111	101	
9	000010	1100	0011	010	101	100	110	
10	0000011	01000	1110	001	001	011	100	
11	0000010	11011	1010	0111	000	10111	-	
12	00000001	11010	11000	1111	01001	-	-	
13	00000000	010010	110011	111100	-	-	-	
14	00000011	0100111	110010	-	-	-	-	
15	000000101	0100110	-	-	-	-	-	
NumCoeff TotalZeros	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	101000	111000	10000	11000	1000	100	00	0
2	101001	111001	10001	11001	1001	101	01	1
3	10101	11101	1001	1101	101	11	1	-
4	1011	1111	101	111	0	0	-	-
5	110	00	01	0	11	-	-	-
6	00	01	11	10		-	-	-
7	111	10	00	-	-	-	-	-
8	01	110	-	-	-	-	-	-
9	100	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-
11	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-
15	-	-	-	-	-	-	-	-

Table 9-13: TotalZeros table for chroma DC 2x2 blocks

NumCoeff	**1**	**2**	**3**

TotalZeros			
0	1	1	1
1	01	01	0
2	001	00	-
3	000	-	-

9.6.2.4 Run before each coefficient

At this stage it is known how many zeros are left to distribute (call this ZerosLeft). When decoding a non-zero coefficient for the first time, ZerosLeft begins at TotalZero, and decreases as more non-zero coefficients 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 backward scan. Then the value is known to be ZerosLeft. This also means that the maximum value to be coded is 14.

Table 9-14 Tables for Run before each coefficient

RunsLeft Run Before	1	2	3	4	5	6	>6
0	**1**	**1**	**01**	**01**	**01**	**01**	**000**
1	**0**	**01**	**00**	**00**	**00**	**00**	**010**
2	-	**00**	**11**	**11**	**11**	**101**	**101**
3	-	-	**10**	**101**	**101**	**100**	**100**
4	-	-	-	**100**	**1001**	**111**	**111**
5	-	-	-	-	**1000**	**1101**	**110**
6	-	-	-	-	-	**1100**	**0011**
7	-	-	-	-	-	-	**0010**
8		-	-	-	-	-	**00011**
9	-	-	-	-	-	-	**00010**
10	-	-	-	-	-	-	**00001**
11	-	-	-	-	-	-	**0000011**
12	-	-	-	-	-	-	**0000010**
13	-	-	-	-	-	-	**0000001**
14	-	-	-	-	-	-	**00000001**

10 Context-Based Adaptive Binary Arithmetic Coding

10.1 Overview

Next we give a short overview of the main coding elements of the CABAC entropy coding scheme. Suppose a symbol related to an arbitrary syntax element is to be coded. In a first step, a suitable set of prior transmitted symbols is chosen that should be useful in estimating the symbol to be coded. This process of constructing a model conditioned on neighbouring symbols is commonly referred to as *context modelling* and is the first step in the CABAC entropy coding scheme. The particular context models that are designed for each given symbol are described in detail in Section 10.4 - 10.5. Generally, the design of the context models is such that it involves at most two neighboring symbols to the left and on top of a given symbol, as is shown in Figure 10-1.

In a second step, if a given symbol is non-binary valued, it will be mapped onto a sequence of binary decisions called bins. The actual binarization of non-binary valued symbols is done according to specific binary trees, as specified in Section 10.4-10.5.

Next, for each bin a Context Variable is defined by an equation containing the prior transmitted symbols, or parts thereof, defined by the Context Modelling. The possible numerical values of the Context Variable are called Contexts. Typically, there are several possible values, i.e. several Contexts. However, in some cases the context variable may simply be a constant, in which case there is only one Context.

Associated with each Context is a probability distribution. Since only bins are encoded, each probability distribution is determined by just a single number *p,* which, for example, represents the probability of the bin value "1". In some cases several bins may share the same Context Variable and Probability Model. The initial values for the Probability Model may be supplied by the Context Modelling or the binarization.

During actual coding of a bin at a particular point, block, macroblock, etc. in the picture, first the context is calculated, then the bin is encoded with the adaptive binary arithmetic coding (AC) engine using the probability distribution corresponding to the calculated context. After encoding of each bin, the Probability Model will be updated using the value of the encoded bin. Hence, CABAC keeps track of the actual statistics during coding.

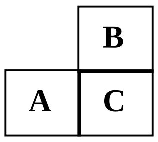


Figure 10-1 – Illustration of generic conditioning scheme using neighbouring symbols A and B for conditional coding of a current symbol C

10.2 Initialisation of Context Models

For a proper initialisation of the probability model associated with each context, initial counts c_0 and c_1 of the events "0" and "1", respectively, are provided. Regarding the initialisation, there are three categories of models: a) models with initial counts depending on the quantisation parameter (QP), b) models with fixed initial counts (independent of the QP) and c) models with a flat, i.e. uniform initialisation. In each of the three cases, the given initial counts have to be translated into the representation of the probability models as specified in subclause 10.qq.xx.yy. Next, a detailed description of the initialisation process for all three categories of context models is given.

10.2.1 Models with QP-independent Initial Counts

Given the initial counts c_0 and c_1, in a first step a conditional rescaling operation is performed in order to guarantee that the condition $2 \leq c_{total} < 17$ holds, where : $c_{total} = c_0 + c_1$:

```
Count_Rescaling:

while ($c_{total}$ >16)
{
    $c_{temp} \leftarrow c_1$
    $c_1 \leftarrow (c_1+1)>>1$
    $c_{total} \leftarrow c_1+((c_{total} - c_{temp}+ 1)>>1)$
}
```

In a second step, the *most probable symbol* (MPS) and the state index *State* corresponding to the underlying probability model is determined by using the appropriately rescaled counts of the first step:

```
Get_MPS:

MPS $\leftarrow$ 0
if (($c_{total}$ - $c_1$) < $c_1$)
```

```
{
    MPS←1
    State ← StateTab[$c_{total}$ - $c_1$ - 1][ $c_{total}$ - 2]
}
else
{
    State ← StateTab[$c_1$ - 1][ $c_{total}$ - 2]
}
```

Here the translation between counts and the corresponding probability states of the LPS is obtained by means of the table *StateTab* as shown in Table 10-1. In the following, this method of initialisation will be referred as *Ini_type* 1.

Table 10-1 – StateTab for translating between given counts and LPS related probability states

$c_{LPS}-1$	$c_{total}-2$														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	1	2	3	4	5	32	33	34	35	36	37	38	39	40
1	0	0	6	7	8	9	23	25	26	27	28	29	30	31	32
2	0	0	0	0	10	11	17	20	20	21	23	24	25	26	27
3	0	0	0	0	0	0	12	15	16	18	20	21	22	22	23
4	0	0	0	0	0	0	0	0	13	14	16	18	19	19	20
5	0	0	0	0	0	0	0	0	0	0	13	14	16	17	17
6	0	0	0	0	0	0	0	0	0	0	0	0	13	14	15
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13

10.2.2 Models with QP-dependent Initialisation

For QP-dependent initialisation, scaling factors r_0 and r_1 will be given in addition to the initial counts c_0 and c_1 such that scaled counts cs_0 and cs_1 are determined either as:

```
Scale_Counts Case1:
 {
    qp_factor  ←  min(33, max (0,QP-22))-12
    $cs_0$ ←  $c_0$ + ($r_0$*qp_factor)/8
    $cs_1$ ←  $c_1$ + ($r_1$*qp_factor)/8
 }
```

or

```
Scale_Counts Case2:
  {
    qp_factor  ←  min(36, max (0,40-QP))-12
    $cs_0$ ←  $c_0$ + ($r_0$*qp_factor)/16
    $cs_1$ ←  $c_1$ + ($r_1$*qp_factor)/16
}.
```

In both cases the scaled counts cs_0 and cs_1 are further processed in the same way as described in subclause 10.qq.1 in order to obtain the corresponding probability state index *State* and the value of MPS. This method of QP-dependent initialisation will be called *Ini_type 2, case 1* or *Ini_type 2, case 2* depending on whether Scale_Counts_Case1 or Scale_Counts_Case2 will be applied, respectively.

10.2.3 Models with Uniform Initialisation

In this case, no assumption of the initial probability distribution is made and the initial values of the counts are given by

$$c_0 = c_1 = 1, \qquad (10\text{-}1)$$

such that *MPS*=0 and *State*=0. This method of initialisation is called *Ini_type 3*.

10.3 Context Modelling and Binarization for Coding of Motion and Mode Information

In this section the context models and binarization schemes used for the syntax elements macro block type (MB_type), motion vector data (mvd_fwd, mvd_bwd) and reference frame parameter (ref_idx_fwd, ref_idx_bwd) are described.

10.3.1 Macroblock Type (MB_type)

Context models and binarization for MB_type depend on the slice type. In the following, a detailed description for the distinct cases of I-, P- and B-slices is given.

10.3.1.1 I Slices

For I slices, there are two possible mode decisions for each macroblock, namely Intra4x4 and Intra16x16 mode, such that signalling the mode information is reduced to transmitting a binary decision. Coding of the binary decision MB_type(C) for a given macroblock C is conditioned on the MB_type values of the neighbouring macroblocks A to the left and B on top of the current macroblock C as depicted in Figure (10-2). The related context *ctx_mb_type_I(C*) is determined by

$$ctx\_mb\_type\_I(C) = ((MB\_type(A)!{=}Intra4x4)\ ?\ 1 : 0) + ((MB\_type(B)!{=}Intra4x4)\ ?\ 1 : 0). \qquad (10\text{-}2)$$

This results in three different contexts (0,1,2) according to the 4 possible combinations of MB_type for A and B, which are initialised using Ini_type 1 with initial counts as given in Table 10-2. In the case that one or both of the neighbouring macroblocks A and B are not available, their corresponding values in the sum of the right hand side of (10-1) are assumed to be zero.

Additional information has to be encoded if MB_type(C)=Intra16x16. This will be described in subclause 10.qq.1.4.

Table 10-2 – Initial counts for context variable *ctx_mb_type_I*

Value of *ctx_mb_type_I*	Count c_0	Count c_1
0	8	1
1	2	1
2	2	1

Table 10-3 – Binarization for macroblock modes in P-slices

Code Number	Macroblock Mode	Binarization
0	SKIP	0
1	16x16	1 0 0 0
2	16x8	1 0 1 1
3	8x16	1 0 1 0
4	8x8 (split)	1 0 0 1
5 (VLC only)	8x8 (split, all ref=0)	
6	Intra4x4	1 1 0
7	Intra16x16	1 1 1
	bin	1 2 3 4

Table 10-4 – Binarization for 8x8 sub-partition modes in P-slices

Code Number	8x8 Partition Mode	Binarization
0	8x8	1
1	8x4	0 0 0
2	4x8	0 0 1 1
3	4x4	0 0 1 0
4	BlockIntra4x4	0 1
	bin	1 2 3 4

10.3.1.2 P Slices

Tables (10-3) and (10-4) show the macroblock modes and the modes of 8x8 sub-partitions for P-slices together with their binarization. The context modelling for the individual bins as shown in Tables (10-3) and (10-4) depends on their position and the values of the preceding bins in the corresponding binarization in the following way.

For the first bin of the binarization of the macroblock mode given in Table (10-3) three distinct context models are used depending on the macroblock mode of the neighbouring macroblocks. The context determination is as follows:

$$ctx\_mb\_type\_P(C) = ((MB\_type(A)!=SKIP)\ ?\ 1 : 0) + ((MB\_type(B)!=SKIP)\ ?\ 1 : 0), \qquad (10\text{-}3)$$

where A and B denote the neighbouring macroblocks to the left and on top of the current macroblock C. In the case that one or both of the neighbouring macroblocks A and B are not available, their corresponding values in the sum of the right side of Equation (10-2) are assumed to be zero.

Table 10-5 – Initial counts for context variable *ctx_mb_type_P*

Value of *ctx_mb_type_P*	c_0	c_1	r_0	r_1
0	7	2	5	0
1	1	2	0	0
2	1	2	0	0
3	30	1	0	0
4	2	1	0	0
5	5	9	9	-9
6	4	3	0	0

For coding of the second bin a fixed context model *ctx_mb_type_P(C)*=3 is used. Depending on the coding decision in the second bin, the decision in bin 3 is coded either in context model *ctx_mb_type_P(C)*=4 (second bin = 0) or *ctx_mb_type_P(C)*=6 (second bin = 1). The decision in bin 4 is finally coded using *ctx_mb_type_P(C)*=5, if the decision in third bin has the value 0; otherwise it is coded using *ctx_mb_type_P(C)*=6. The initialisation of the context model *ctx_mb_type_P* is performed using Ini_type 2, case 1, where the start values of Table 10-5 are used.

Table 10-6 – Initial counts for context variable *ctx_b8_type_P*

Value of *ctx_b8_type_P*	Count c_0	Count c_1
0	1	2
1	2	1

2	1	1
3	1	2

Coding of the bins corresponding to the modes for 8x8 sub-partitions as shown in Table 10-4 is performed as follows: The first bin is coded in fixed context model *ctx_b8_type_P(C)*=0 and for each binary decision in the next higher bin > 1 *ctx_b8_type_P(C)* is incremented by one. Thus, a total number of 3 additional context models (1,2,3) are used for the coding of the modes for 8x8 sub-partitions. The initial counts for context variable *ctx_b8_type*_P are given in Table 10-6, where the initialisation method Ini_type 1 is applied to this context variable.

If MB_type(C)=Intra16x16, the additional information is encoded as described in subclause 10.qq.1.4

10.3.1.3 B Slices

In Table 10-7 the binarization for the macroblock modes of B-slices are shown. Similar to the coding of the P slice modes, the context modelling for the individual bins as shown in Table 10-7 depends on their position and the values of the preceding bins in the corresponding binarization. In the following, a detailed specification of the context determination for each bin is given.

Table 10-7 – Binarization for macroblock modes in B-slices

Macroblock Mode	1. Block	2. Block	Binarization							
Direct (cbp=0)			0							
Direct			1	0						
16x16	Forw.		1	1	0	0				
16x16	Backw.		1	1	0	1				
16x16	Bidirect.		1	1	1	0	0	0	0	
16x8	Forw.	Forw.	1	1	1	0	0	0	1	
8x16	Forw.	Forw.	1	1	1	0	0	1	0	
16x8	Backw.	Backw.	1	1	1	0	0	1	1	
8x16	Backw.	Backw.	1	1	1	0	1	0	0	
16x8	Forw.	Backw.	1	1	1	0	1	0	1	
8x16	Forw.	Backw.	1	1	1	0	1	1	0	
16x8	Backw.	Forw.	1	1	1	0	1	1	1	
8x16	Backw.	Forw.	1	1	1	1	1	1	0	
16x8	Forw.	Bidirect.	1	1	1	1	0	0	0	0
8x16	Forw.	Bidirect.	1	1	1	1	0	0	0	1
16x8	Backw.	Bidirect.	1	1	1	1	0	0	1	0
8x16	Backw.	Bidirect.	1	1	1	1	0	0	1	1
16x8	Bidirect.	Forw.	1	1	1	1	0	1	0	0
8x16	Bidirect.	Forw.	1	1	1	1	0	1	0	1
16x8	Bidirect.	Backw.	1	1	1	1	0	1	1	0
8x16	Bidirect.	Backw.	1	1	1	1	0	1	1	1
16x8	Bidirect.	Bidirect.	1	1	1	1	1	0	0	0
8x16	Bidirect.	Bidirect.	1	1	1	1	1	0	0	1
8x8 (split)			1	1	1	1	1	1	1	

Intra4x4			1	1	1	1	1	0	1	0
Intra16x16			1	1	1	1	1	0	1	1
		bin	1	2	3	4	5	6	7	8

Table 10-8 – Binarization for 8x8 sub-partition modes in B-slices

Code Number	8x8 Sub-partition Mode	Binarization						
0	Direct	0						
1	8x8 Forward	1	0					
2	8x8 Backward	1	1	0	0			
3	8x8 Bidirect	1	1	0	1			
4	8x4 Forward	1	1	1	0	0	0	0
5	4x8 Forward	1	1	1	0	0	0	1
6	8x4 Backward	1	1	1	0	0	1	0
7	4x8 Backward	1	1	1	0	0	1	1
8	8x4 Bidirect	1	1	1	0	1	0	0
9	4x8 Bidirect	1	1	1	0	1	0	1
10	4x4 Forward	1	1	1	0	1	1	0
11	4x4 Backward	1	1	1	0	1	1	1
12	4x4 Bidirect	1	1	1	1	1	1	0
13	BlockIntra4x4	1	1	1	1	0	0	0
	bin	1	2	3	4	5	6	7

The first bin of the binarization of the macroblock mode given in Table 10-7 uses three context models depending on the macroblock mode of the neighboring macroblocks. For this bin the context determination is as follows:

$$ctx\_mb\_type\_B(C) = ((MB\_type(A)==SKIP)\ ?\ 1 : 0) + ((MB\_type(B)==SKIP)\ ?\ 1 : 0), \qquad (10\text{-}4)$$

where SKIP denotes the macroblock type Direct mode with cbp=0, and where A and B denote the neighbouring macroblocks to the left and on top of the current macroblock C. In the case that one or both of the neighbouring macroblocks A and B are not available, their corresponding values in the sum of the right hand side of Equation (10-3) are assumed to be zero.

This results in three different contexts (0,1,2) according to the 4 possible combinations of MB_type for A and B. For coding of the second bin another three context models (3,4,5) are determined in the following way:

$$ctx\_mb\_type\_B(C) = ((MB\_type(A)!=Direct)\ ?\ 1 : 0) + ((MB\_type(B)!=Direct)\ ?\ 1 : 0). \qquad (10\text{-}5)$$

Here, the choice of the context depends on whether the neighbouring macroblock A to the left and B on top of the current macroblock is coded in Direct mode regardless of the value of the corresponding cbp. As before, the values in the sum of the right hand side of Equation (10-4) are assumed to be zero, if one or both of the neighbouring macroblocks A and B are not available.

Table 10-9 – Initial counts for context variable *ctx_mb_type_B*

Value of *ctx_mb_type_B*	c_0	c_1	r_0	r_1
0	17	1	-18	0

1	2	3	0	-2
2	1	7	0	-5
3	2	5	0	0
4	1	6	0	0
5	1	10	0	0
6	1	5	0	-5
7	2	5	0	-5
8	2	3	0	0

For coding of the third bin a fixed context model *ctx_mb_type_B(C)*=6 is used. If the decision of the third bin has the value 1, the fourth bin is coded with context model *ctx_mb_type_B(C*)=7, otherwise the model *ctx_mb_type_B(C)*=8 is used. All remaining bins use the fixed model *ctx_mb_type_B(C)*=8. The initialisation of the context model *ctx_mb_type_B* is performed by using Ini_type 2, case 1. The corresponding initial values are given in Table 10-9.

The binarizations of the modes for 8x8 sub-partitions are shown in Table 10-8. The corresponding context models are chosen as follows: The first bin is coded with context model *ctx_b8_type_B(C)*=0, while for the second bin the context model *ctx_b8_type_B(C)*=1 is chosen. In the case the value of the second bin is equal to one, the decision of the third bin is coded using the model *ctx_b8_type_B(C)*=2; otherwise it is coded using the model *ctx_b8_type_B(C)*=3. All decisions related to bin > 3 are coded using the fixed model *ctx_b8_type_B(C)*=3. The initial counts for the context variable *ctx_b8_type_B* are given in Table 10-10; the initialisation method Ini_type 1 applies to this context variable.

If MB_type(C)=Intra16x16, the additional information has to be encoded. This will be described in subclause 10.qq.1.4.

Table 10-10 – Initial counts for context variable *ctx_b8_type_B*

Value of *ctx_b8_type_B*	Count c_0	Count c_1
0	1	3
1	3	1
2	3	2
3	1	1

Table 10-11 – Initial counts for context variable *ctx_mb_intra16x16*

Value of *ctx_mb_intra16x16*	I slice		P-,B-slice	
	c_0	c_1	c_0	c_1
0	1	1	7	2
1	1	1	2	1
2	2	1	3	2
3	2	1	-/-	-/-
4	1	1	-/-	-/-

10.3.1.4. Additional Information for Mode Intra16x16

For the macroblock mode Intra16x16, additional information has to be encoded to signal the occurrence of significant AC-coefficients (AC=1 or 2), the coded block pattern for the chroma coefficients (nc=0,1 or 2) and the chosen intra prediction mode (Imode= 0,1,2 or 3) of the related macroblock. This will be done as follows.

First, the one-bit symbol AC is coded using the fixed context model *ctx_mb_intra16x16*=0. For I-Slices, a model increment *ctx_incr*=1 is chosen, while for P- and B-Slices *ctx_incr*=0, such that in a second step nc is encoded as follows:

```
if (nc==0)
{
   Encode(0, (ctx_mb_intra16x16=1));
}
else
{
   Encode(1, (ctx_mb_intra16x16=1));
   if (nc==1)
   {
      Encode(0, (ctx_mb_intra16x16=1+ctx_incr));
   }
   else
   {
      Encode(1, (ctx_mb_intra16x16=1+ctx_incr));
   }
}
```

Finally, the two-bit symbol Imode is coded bit by bit using one fixed context model *ctx_mb_intra16x16*=2 in case of a P- or B-slice, or two fixed context models *ctx_mb_intra16x16*=3,4 in case of an I-slice:

```
Encode((Imode>>1), (ctx_mb_intra16x16=2+ctx_incr));
Encode((Imode&1), (ctx_mb_intra16x16=2+2*ctx_incr)).
```

The initial counts for the context variable *ctx_mb_intra16x16* are given in Table 10-11; the initialisation method Ini_type 1 applies to this context variable.

10.3.2 Context Models for Motion Vector Data

Motion vector data consists of residual vectors obtained by applying motion vector prediction. Thus, it is a reasonable approach to build a model conditioned on the local prediction error. The simple estimate of the local prediction error at a given block C, which is used in this Recommendation | International Standard, is given by evaluating the L1-norm

$$e_k(A,B) = |mvd_k(A)| + |mvd_k(B)| \qquad (10\text{-}6)$$

of two neighbouring motion vector prediction residues $mvd_k(A)$ and $mvd_k(B)$ for each component *k* of a motion vector residue $mvd_k(C)$ of a given block, where A and B are neighbouring blocks of block C, as shown in Figure 10-2. If one of the neighbouring blocks belongs to an adjacent macroblock, we take the residual vector component of the leftmost neighbouring block in the case of the upper block B, and in the case of the left neighbouring block A we use the topmost neighbouring block. If one of the neighbouring blocks is not available, because, for instance, the current block is at the slice boundary, the corresponding value of the right hand side of Equation (10-5) is set to zero. By using e_k as defined in Equation (10-5), a context variable *ctx_mvd(C,k)* for the residual motion vector component $mvd_k(C)$ is defined as follows:

$$ctx\_mvd(C,k) = \begin{cases} 0, e_k(C) < 3, \\ 2, e_k(C) > 32, \\ 1, otherwise. \end{cases} \qquad (10\text{-}7)$$

Note, that in the case of a B-slice motion vector prediction residue mvd_fwd or mvd_bwd the corresponding two neighbouring motion vector prediction residues $mvd_k(A)$ and $mvd_k(B)$, which are used in Equation (10-5) must both be chosen forward or backward, respectively. If one of these residual vectors is not available, the corresponding value of the right hand side of Equation (10-5) is set to zero.

For the actual coding process, $mvd_k(C)$ is separated into sign and modulus. The modulus is encoded using the binarization in Table 10-12. The construction of a codeword of this modified unary binarization table for a given index *v* is given by the following rules:

If *v<64 (the unary code cut-off value for mvd_fwd),*
    Use a unary code of *v* 1's terminated with a 0.
If *v>=64*,

1. Form an initial prefix of *63* 1's.
2. Extract the symbol γ representing the *most significant bits* (MSB) of *v-62, i.e.,* $\gamma = \lfloor \log_2(v-62) \rfloor$, and put it in a unary representation. This part is appended to the initial prefix to form the prefix of the codewords shown in the 'Unary Prefix' column of Table 10-12.
3. Append the γ *least significant bits* (LSB) in its binary representation to the prefix.
4. The corresponding suffix bits are shown in the LSB column of Table 10-12.

Given the binarization of the modulus of $mvd_k(C)$ according to Table 10-12, only the first bin is coded using the context variable *ctx_mvd(C,k)*. For the remaining bins of the modulus, 4 additional context models are used: *ctx_mvd(C,k)*=3 for the second bin, *ctx_mvd(C,k)*=4 for the third bin, *ctx_mvd(C,k)*=5 for the fourth bin, and the context *ctx_mvd(C,k)*=6 for all remaining bins. This results in a total sum of 7 context models for each vector component. Initialisation of *ctx_mvd* is performed by using Ini_type 1. The corresponding initial values are given in Table 10-13.

Finally, the sign of $mvd_k(C)$ is encoded/decoded by using the arithmetic coding bypass routine Encode_eq_prob/-Decode_eq_prob as specified in subclause 10.qq.xx.yy.

Table 10-12 – Binarization of the mvd_fwd modulus

Index	Unary Prefix	Exp-Golomb Suffix
0	0	
1	1 0	
2	1 1 0	
3	1 1 1 0	
63	1 … … 1 0	
64	1 … … 1 10	0
65	1 … … 1 10	1
66	1 … … 1 110	0 0
67	1 … … 1 110	0 1
68	1 … … 1 110	1 0
69	1 … … 1 110	1 1
70	1 … … 1 1110	0 0 0
71	1 … … 1 1110	0 0 1
72	1 … … 1 1110	0 1 0
73	1 … … 1 1110	0 1 1
74	1 … … 1 1110	1 0 0
75	1 … … 1 1110	1 0 1
…	…	…
bin	1.2.3.4…	…

Table 10-13 – Initial counts for context variable *ctx_mb_intra16x16*

Value of *ctx_mvd(·,k)*	k=0		k=1	
	c_0	c_1	c_0	c_1
0	9	5	13	5
1	1	1	6	5
2	4	5	1	1
3	1	2	4	5
4	1	4	1	4
5	1	2	2	5
6	1	10	1	10

10.3.3 Reference Frame Parameter

If the option of temporal prediction from more than one reference frame is enabled, the chosen reference frame parameters for each sub-block *C* of a given macroblock partition must be signalled. The context number for the current block *C* is determined by

$$ctx\_ref\_idx\ (C) = ((ref\_idx\_fwd(A)!=0)?1:0) + 2 \times ((ref\_idx\_fwd\ (B)!=0)?1:0), \qquad (10\text{-}8)$$

where A and B represent the corresponding blocks to the left and on the top of the regarded block C (see Figure 10-1). If a neighbouring block X (A or B) is not available, because it is positioned in a different slice, or because e.g. X is INTRA-coded, the corresponding ref_idx_fwd(X) value is replaced by the default value zero. In case of coding a reference parameter ref_idx_fwd or ref_idx_bwd from a B-slice, the corresponding reference parameters from neighbouring blocks as used in Equation (10-7) must be located in the same reference set (forward or backward, resp.). If none of those is available, the corresponding value in Equation (10-7) is set to zero.

Equation (10-7) results in 4 possible contexts (0 to 3) used for coding of the first bin of the binary equivalent of the reference frame parameter ref_idx_fwd(*C*). A single context *ctx_ref_idx(C)*=4 is used for the second bin. Another context *ctx_ref_idx(C)*=5 is used for all remaining bins of the reference frame parameter, which sums up to a total number of six different contexts (and thus six probability estimations) for the reference frame information. Table 10-14 shows the unary binarization, which is applied to the reference frame parameter. Initialisation of *ctx_ref_idx* is performed by using Ini_type 1, where the corresponding initial values are given in Table 10-15.

Table 10-14 – 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		
…	…	…	…	…	…	…	…
bin	1	2	3	4	5	6	…

Table 10-15 – Initial counts for context variable *ctx_ref_idx*

Value of *ctx_ref_idx*	Count c_0	Count c_1
0	10	1
1	2	1
2	1	1
3	1	3
4	2	1
5	1	1

10.4 Context Modelling and Binarization for Coding of Texture Information

This section provides detailed information about the context models used for the syntax elements of coded block pattern (cbp) and intra prediction mode (IPRED). Furthermore, the coding of transform coefficients is described.

10.4.1 Coded Block Pattern

Except for MB_type Intra16x16, the context modelling for the coded block pattern is treated as follows. There are 4 luma cbp bits belonging to four 8x8 blocks in a given macroblock. Let cbp_luma_bit(C) denote the Y-cbp bit for a sub-block C. Then, we define the corresponding context variable for the coding of cbp_luma_bit(C) by

$$ctx\_cbp\_luma(C) = cbp\_luma\_bit(A) + 2*cbp\_luma\_bit(B), \tag{10-9}$$

where cbp_luma_bit(A) and cbp_luma_bit(B) are Y-cbp bits of the neighbouring 8x8 blocks A and B, respectively, as depicted in Figure 10-2. In the case, that the corresponding cbp information is not available, default values of zero are substituted in the right hand side of Equation (10-8).

The remaining 2 bits of cbp are related to the chroma coefficients. These bits are translated into two dependent binary decisions, such that, in a first step, we send a bit cbp_chroma_sig, which signals whether there are significant chroma coefficients at all. Let C denote the corresponding macroblock, then the related context model is defined similar to that of the Y-cbp bits, i.e. the context variable is defined as

$$ctx\_cbp\_chroma\_sig(C) = cbp\_chroma\_sig(A)+ 2* cbp\_chroma\_sig(B), \tag{10-10}$$

where cbp_chroma_sig(A) and cbp_chroma_sig(B) are notations for the corresponding cbp_chroma_sig bits of neighbouring macroblocks A and B to the left and on top of C, respectively.

If cbp_chroma_sig(C) = 1 (indicating that non-zero chroma coefficients exist), then a second bit cbp_chroma_ac is signalled, which indicates the presence of non-zero AC chroma coefficients. This is done by using a context model conditioned on the cbp_chroma_ac decisions cbp_chroma_ac(A) and cbp_chroma_ac(B) of neighbouring macroblocks, i.e. the corresponding context variable is given by:

$$ctx\_cbp\_chroma\_ac(C) = cbp\_chroma\_ac(A) + 2* cbp\_chroma\_ac(B). \qquad (10\text{-}10)$$

Here and in Equation (10-9) the cbp information of non-available macroblocks is replaced by default values of zero. Note, that due to the different statistics there are different models for Intra and Inter macroblocks for each of the cbp bits, so that the total number of different contexts (and probability distributions) for cbp amounts to $2 \times 3 \times 4$=24. In Table 10-16 and 10-17 the initial values used for the cbp information is shown. Here the method of initialisation depends on the slice type. For I slices Ini_type 1 is used with corresponding values in Table 10-16, while for P- and B-slices method Ini_type 2, case 1 is applied using the values given in Table 10-17.

The context modelling and coding of cbp for MB_type Intra16x16 is given in subclause 10.qq.1.4.

Table 10-16 – Initial counts for context variable for cbp context variable (I-slices only)

Value of	ctx_cbp_luma	ctx_cbp_chroma_sig	ctx_cbp_chroma_ac

context variable	c_0	c_1	c_0	c_1	c_0	c_1
0	1	4	1	2	1	1
1	1	2	1	3	1	1
2	1	2	1	3	1	2
3	4	3	1	3	1	2

Table 10-17 – Initial counts for context variable for cbp context variable (P,B-slices only)

Value of context variable	ctx_cbp_luma				ctx_cbp_chroma_sig				ctx_cbp_chroma_ac			
	c_0	c_1	r_0	r_1	c_0	c_1	r_0	r_1	c_0	c_1	r_0	r_1
0	1	4	2	-2	3	1	5	0	5	2	2	0
1	1	1	5	0	6	5	0	-2	1	1	5	2
2	1	1	5	0	6	5	-5	-5	1	1	0	0
3	3	1	9	0	1	2	2	0	1	2	0	0

10.4.2 Intra Prediction Mode

In Intra4x4 mode, coding of the intra prediction mode of a given block C is conditioned on the intra prediction mode of the previous block A to the left of C (see Figure 10-1). In fact, it is not the prediction mode value itself, which is signalled and which is used for conditioning, but rather its predicted order ipred(C) as it is described in subclause qq.4.4.2. Note however, that unlike the case of VLC coding, information on intra prediction of two 4x4 luma blocks is not coded in one codeword.

There are nine different prediction modes (0,…,8) and for each predicted order ipred(A) of the neighbouring block A , two different contexts are supplied:

$$ctx\_ipred(C)= 2*ipred(A)$$

for the first bin of the binary equivalent of C and

$$ctx\_ipred(C)=2* ipred(A)+1$$

for all remaining bins. If ipred(A) of the neighbouring block A is not available, *ctx_ipred(C)*=0 is used for the first bin and *ctx_ipred(C)*=1 for the second bin. In total, there are 2*9=18 different contexts for coding of intra prediction modes.

Binarization of ipred(C) is performed using the unary binarization of Table 10-14. However, since the maximum number of ipred(C) is known in advance, the binarization can be restricted to a truncated code tree. That means, that for the binarization of the maximum value 8 of ipred(C) the terminating “1” can be omitted.

Note, that coding of the prediction modes for the macroblock type Intra16x16 is defined in subclause 10.qq.1.4.

10.4.3 Transform Coefficients

10.4.3.1 Overview

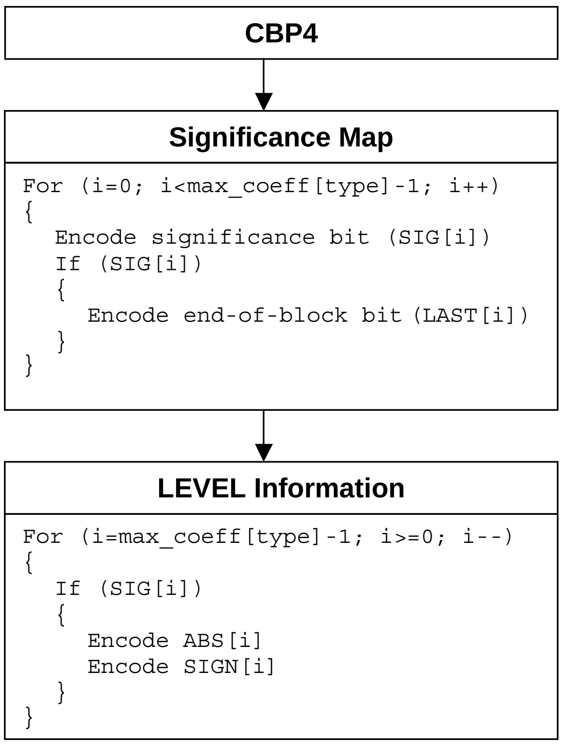
Figure 10-3 gives an overview of the coding scheme for transform coefficients. Firstly, a one-bit symbol called cbp4 is transmitted for each block of transform coefficients unless the cbp (on macroblock level) indicates that the regarded block has no non-zero coefficients. The cbp4 symbol is one, if there are any significant coefficients inside the block. If it is zero, no further information is transmitted for the block; otherwise a significance map specifying the positions of significant coefficients is encoded. Afterwards, the absolute value as well as the sign is encoded for each significant transform coefficient. These values are transmitted in reverse scanning order.

As shown in Table 10-7, there are 12 different types of transform coefficient blocks. In general, each of these transform coding units has a different statistics. However, for most sequences and coding conditions, the statistics of some of the

12 different types are very similar. Thus, in order to keep the number of contexts reasonably small, the given block types are classified into 5 categories as specified in the right column of Table 10-7. For each of these categories, a separate set of context models is used, as will be explained in more detail below.

Table 10-18 –Context categories for the different block types

Block type	max_coeff.	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 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	



```
For (i=0; i<max_coeff[type]-1; i++)
{
   Encode significance bit (SIG[i])
   If (SIG[i])
   {
      Encode end-of-block bit (LAST[i])
   }
}
```

```
For (i=max_coeff[type]-1; i>=0; i--)
{
   If (SIG[i])
   {
      Encode ABS[i]
      Encode SIGN[i]
   }
}
```

Figure 10-2 –Illustration of CABAC coding scheme for transform coefficients

10.4.3.2 cbp4

For each of the given transform block types, a one-bit symbol cbp4 is signaled first to indicate whether there are significant (non-zero) coefficients inside the regarded block of transform coefficients. If the cbp4 symbol is zero, no further information needs to be transmitted for the block.

For encoding the cbp4 bit, four different contexts are used for each of the five categories specified in Table 10-7. The context variable *ctx_cbp4 (C)* for the current block *C* is determined by

$$ctx\_cbp4\ (C) = cbp4\ (A) + 2 \times cbp4\ (B),$$

where A and B represent the corresponding blocks of the same type to the left and on the top of the regarded block C (see Figure 10-2). Only blocks of the same type (left column of Table 10-7) are used for context determination. If a neighboring block X (A or B) is positioned in a different slice, or if no neighboring block X of the same type exist (e.g. because the current block is INTRA-coded while the neighboring block X is INTER-coded), the corresponding cbp4(X) value is replaced by a default value. If the current block is coded in an INTRA-mode, a default value of 1 is used; otherwise, a default value of 0 is used. This results in a total sum of four contexts for each of the block types.

So, while all twelve block types (left column of Table 10-7) are distinguished for determining the context variable *ctx_cbp4 (C)*, only five different sets of contexts (each for one category specified in the right column of Table 10-7) are used for encoding the cbp4 symbol. This results in a total number of $5 \times 4=20$ contexts for the cbp4 bit.

10.4.3.3 Significance Map

If the cbp4 indicates that a block has significant coefficients, a significance map is encoded. For each coefficient in scanning order, a one-bit symbol (SIG) is transmitted. If the SIG symbol is one, that is, if a non-zero coefficient exists at this scanning position, a further one-bit symbol (LAST) is sent. This symbol indicates if the current significant coefficient is the last one inside the block or if further significant coefficients follow.

Coefficients	**14**	**0**	**-5**	**3**	**0**	**0**	**-1**	**0**	**1**
SIG	1	0	1	1	0	0	1	0	1
LAST	0		0	0			0		1

Coefficients	**18**	**-2**	**0**	**0**	**0**	**-5**	**1**	**-1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
SIG	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	(1)
LAST	0	0				0	0	0					0			(1)

Figure 10-19 – Two examples for encoding the significance map (the symbols in parenthesis are not transmitted)

Figure 10-4 shows two examples for the significance map encoding. Note, that the significance information (SIG, LAST) for the last scanning position of a block is never transmitted. If the last scanning position is reached and the significance map encoding was not already terminated by a LAST-symbol of one, it is obvious that the last coefficient has to be significant (see positions in parenthesis in Figure 10-4).

For encoding the significance map, up to 15 (*max_coeff*-1) contexts (depending on the block type category) are used for both the SIG and the LAST symbols. The context variables are determined by the corresponding scanning position, i.e. for a coefficient *coeff[i]*, which was scanned at the i-th position, the contexts *ctx_sig(coeff[i])* and *ctx_last(coeff[i])* are chosen as follows:

$$\mathit{ctx\_sig}(\mathit{coeff}[i]) = \mathit{ctx\_last}(\mathit{coeff}[i]) = i.$$

For each category of block types, we use *max_coeff*-1 different contexts. This gives a total number of 15+14+15+3+14=61 contexts for both the SIG and the LAST symbol.

10.4.3.4 Level Information

The encoded significance map determines the positions of all significant coefficients inside a block of quantized transform coefficients. The values of the significant coefficients (levels) are encoded by two coding symbols: ABS (representing the absolute value), and SIGN (representing the sign). While SIGN is a one-bit symbol (1 for negative coefficients), a modified unary binarization scheme (see Table 10-8) is used for encoding the absolute values of the coefficients. The levels are transmitted in reverse scanning order (beginning with the last significant coefficient of the block); this allows the usage of more reasonable and effective context models.

The absolute value of the significant coefficients is encoded using the modified uniform binarization as shown in Table 10-8. Here we use two different sets of contexts, one for the first bin given by ctx_abs_1bit, and another one (*ctx_abs_rbits*) for all remaining bins of the binarization (see Figure 10-5):

$$\mathit{ctx\_abs\_1bit} = (\#\ \mathit{transmitted}\ \mathit{ABS}(\mathit{coeff})>1\ ?\ 4 : \max(3, \#\ \mathit{transmitted}\ \mathit{ABS}(\mathit{coeff})=1)), \tag{10-11}$$

$$\mathit{ctx\_abs\_rbits} = \max(4, \#\ \mathit{transmitted}\ \mathit{ABS}(\mathit{coeff})>1). \tag{10-12}$$

The level information is transmitted in reverse scanning order. The context ctx_abs_1bit for the first bin of the absolute values is determined by the number of successive coefficients (in reverse scanning order) having an absolute value of 1. If more than three coefficients have an absolute value of 1, context *ctx_abs_1bit*=3 is always chosen for the first bin. When a coefficient with an absolute value greater 1 is encoded, context *ctx_abs_1bit*=4 is used for the first bin of all remaining coefficients of the regarded block.

All remaining bins of the absolute value are encoded using the context model *ctx_abs_rbits*. It is determined by the number of transmitted coefficients with an absolute value greater than 1 (in reverse scanning order); the maximum context number is restricted to 4. Figure 10-5 shows two examples of the context determination for encoding the absolute values of significant coefficients.

Coefficients	**14**	**0**	**-5**	**3**	**0**	**0**	**-1**	**0**	**1**
ctx_number_abs_1bit	4		4	2			1		0
ctx_number_abs_rbits	2		1	0					

Coefficients	**18**	**-2**	**-1**	**6**	**4**	**-5**	**1**	**-1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
ctx_number_abs_1bit	4	4	4	4	4	3	3	3		2			1			0
ctx_number_abs_rbits	4	3		2	1	0										

Figure 10-20 – Examples of context determination for encoding the absolute value of significant coefficients. The level information is transmitted in reverse scanning order.

The absolute values of significant coefficients are encoded using the binarization in Table 10-8. The construction of a codeword of this binarization table is similar to that used for the modulus of mvd_fwds, i.e. for a given index *v* it is given by the following rules:

If *v<16 (the unary code cut-off value for ABS(coeff)),*

Use a unary code of *v* 1's terminated with a 0.

If *v>=16,*

5. Form an initial prefix of *15* 1's.
6. Extract the symbol γ representing the *most significant bits* (MSB) of *v-14, i.e.,* $\gamma = \lfloor \log_2 (v - 14) \rfloor$, and put it in a unary representation. This part is appended to the initial prefix to form the prefix of the codewords shown in the 'Unary Prefix' column of Table 10-8.
7. Append the γ *least significant bits* (LSB) in its binary representation to the prefix.
8. The corresponding suffix bits are shown in the LSB column of Table 10-8.

For the sign of significant coefficients, the arithmetic bypass routine Encode_eq_prob() as specified in subclause 10.qq.xx.yy. is used

Thus the total number of contexts for encoding the level information is 5*(5+5)-1=49 (for the chroma DC blocks, there are only 4 different contexts for the "remaining" bins of the absolute value, since at maximum 4 coefficients are transmitted).

Table 10-21 – Coefficient level binarization

Index	Unary Prefix	Exp-Golomb Suffix
0	0	
1	1 0	
2	1 1 0	
3	1 1 1 0	
15	1 … … 1 0	
16	1 … … 1 10	0
17	1 … … 1 10	1
18	1 … … 1 110	0 0
19	1 … … 1 110	0 1

20	1 … … 1 110	1 0
21	1 … … 1 110	1 1
22	1 … … 1 1110	0 0 0
23	1 … … 1 1110	0 0 1
24	1 … … 1 1110	0 1 0
25	1 … … 1 1110	0 1 1
…	…	…
bin	1 2 3 4 5…	…

10.4.3.5 Scanning of Transform Coefficients

In CABAC entropy coding the simple single scan (zig-zag scan) is used for all block types including the luma blocks coded in INTRA-4x4 mode.

10.5 Context Modelling for Coding of Dquant

For a given macroblock C, a change of the quantiser value given by the value of Dquant(C) is first mapped to a positive value using the arithmetic wrap as described in subclause QQ. This wrapped value of Dquant(C) is then coded conditioned on the corresponding Dquant(A) of the left neighbouring macroblock A. The context variable ctx_dquant(C) is given by

$$ctx\_dquant(C)=((dquant(A)\ !=0)\ ?\ 1:0).$$

This results in twocontexts (0,1) for the first bin. Another context *ctx_dquant(C)* =2 is used for the second bin. Finally, all remaining bins of the binarized value C are coded using the last context *ctx_dquant(C)* = 3. Thus, a total of four contexts are used for Dquant. The unary binarization applied to the coding of Dquant is given in Table 10-6.

10.6 Table-based Arithmetic Coding

10.6.1 Overview and Principles

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 defined 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:

1) 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}.
2) The range R, which represents the state of the coding engine, is quantized to a small set $\{Q_1,\ldots,Q_4\}$ of pre-defined 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 $R\times P_k$.
3) A separate encoding and decoding path for syntax elements or parts thereof with an approximately uniform probability distribution.

10.6.2 Probability Estimation

The probability estimator is realized by a finite-state machine (FSM) consisting of a set of representative probabilities $\{P_k \mid 0\le k <64\}$ for the LPS together with some appropriately defined state transition rules. Table 10-9 shows 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 encoding of a *MPS* or *LPS* symbol, respectively.

States with indices *State*=0 to 11 correspond to the initial phase of the coding process and a *State*>11 is employed after initialisation, with decreasing LPS probability towards higher states. However, for I-slices it is of advantage to restrict the number of states to the first 36 state indices. Therefore, Table 10-9 contains a separate column containing the transition rule *Next_State_MPS_INTRA* that is used for coding the syntax elements of an I-slice only. Note, that *Next_State_MPS_INTRA* differs from *Next_State_MPS* only by one entry. To prevent the FSM from switching to states higher than *State*=35, we set *Next_State_MPS(35)= 35* for I-slices coding. For the clarity of presentation, a separate table entry for I-slice coding is shown in Table 10-9.

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 with a *State* index of 0,6,10 or 12 and a LPS symbol is observed, the sense of MPS and LPS has to be interchanged, which is indicated by the binary-valued *Switch_MPS* table entries. Thus, each time an LPS is observed, it has to be tested whether the condition *Switch_MPS(State)==1* holds, in which case the meaning of MPS and LPS has to be changed:

```
if (decision == LPS)
{
        if (Switch_MPS(State) == 1)
              MPS(State) ← MPS(State)^1
}
```

Table 10-22 – Probability transition and MPS/LPS Switch

State	Next_State_MPS_INTRA	Next_State_MPS	Next_State_LPS	Switch_MPS
0	1	1	1	1
1	2	2	6	0
2	3	3	7	0
3	4	4	8	0
4	5	5	9	0
5	32	32	23	0
6	7	7	7	1
7	8	8	10	0
8	9	9	11	0
9	23	23	17	0
10	11	11	11	1
11	17	17	12	0
12	13	13	12	1
13	14	14	12	0
14	15	15	13	0

15	16	16	14	0
16	17	17	14	0
17	18	18	15	0
18	19	19	16	0
19	20	20	17	0
20	21	21	18	0
21	22	22	19	0
22	23	23	20	0
23	24	24	20	0
24	25	25	22	0
25	26	26	22	0
26	27	27	22	0
27	28	28	23	0
28	29	29	24	0
29	30	30	25	0
30	31	31	26	0
31	32	32	26	0
32	33	33	26	0
33	34	34	26	0
34	35	35	27	0
35	35	36	28	0
36	-/-	37	29	0
37	-/-	38	30	0
38	-/-	39	31	0
39	-/-	40	31	0
40	-/-	41	32	0
41	-/-	42	32	0
42	-/-	43	33	0
43	-/-	44	33	0
44	-/-	45	34	0
45	-/-	46	34	0
46	-/-	47	35	0
47	-/-	48	35	0
48	-/-	49	36	0
49	-/-	50	36	0
50	-/-	51	37	0

51	-/-	52	37	0
52	-/-	53	38	0
53	-/-	54	38	0
54	-/-	55	39	0
55	-/-	56	39	0
56	-/-	57	40	0
57	-/-	58	40	0
58	-/-	59	41	0
59	-/-	60	41	0
60	-/-	61	42	0
61	-/-	62	42	0
62	-/-	63	42	0
63	-/-	63	43	0

Table 10-23 – RTAB[State][Q] table for interval subdivision

State	0	1	2	3
0	9216	11264	13312	15360
1	6144	7488	8896	10240
2	4608	5632	6656	7680
3	3712	4480	5312	6144
4	3072	3776	4416	5120
5	2624	3200	3776	4416
6	9216	11264	13312	15360
7	7360	9024	10624	12288
8	6144	7488	8896	10240
9	5248	6464	7616	8768
10	9216	11264	13312	15360
11	7872	9664	11392	13184
12	9216	11264	13312	15360
13	8832	10816	12800	14720
14	8512	10368	12288	14144
15	8128	9920	11712	13504
16	7680	9344	11072	12736
17	7168	8768	10368	11968
18	6912	8448	9984	11520

19	6336	7808	9216	10624
20	5888	7232	8512	9856
21	5440	6656	7872	9088
22	5120	6208	7360	8512
23	4608	5632	6656	7680
24	4224	5184	6144	7104
25	3968	4800	5696	6592
26	3712	4480	5312	6144
27	3456	4224	4992	5760
28	3072	3776	4416	5120
29	2816	3456	4096	4736
30	2624	3200	3776	4416
31	2432	3008	3520	4096
32	2304	2816	3328	3840
33	2048	2496	2944	3392
34	1856	2240	2688	3072
35	1664	2048	2432	2816
36	1536	1856	2240	2560
37	1408	1728	2048	2368
38	1344	1600	1920	2176
39	1216	1472	1792	2048
40	1152	1408	1664	1920
41	1088	1344	1536	1792
42	1024	1280	1472	1728
43	960	1216	1408	1600
44	896	1152	1344	1536
45	896	1088	1280	1472
46	832	1024	1216	1408
47	832	960	1152	1344
48	768	960	1088	1280
49	768	896	1088	1216
50	704	896	1024	1152
51	704	832	960	1152
52	640	832	960	1088
53	640	768	896	1088
54	640	768	896	1024

55	576	704	832	960
56	576	704	832	960
57	576	704	832	960
58	512	640	768	896
59	512	640	768	896
60	512	640	768	832
61	512	640	704	832
62	512	576	704	832
63	448	576	704	768

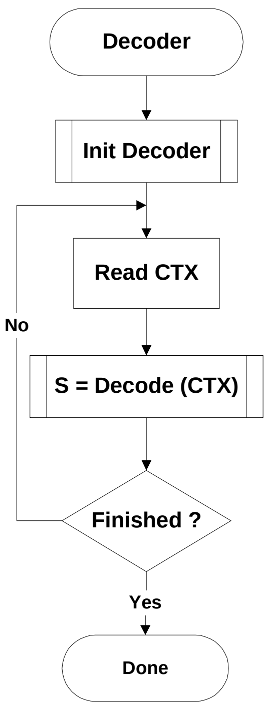


Figure 10-5: Overview of the Decoding Process

10.6.3 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 10-qq gives an illustration of the whole decoding process. Performing the InitDecoder procedure, which is further specified in subclause 10.qq.xx, appropriately initialises *V* and *R*. 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 subclause 10.qq-10.qq. Given the context model *CTX*, the decoding operation Decode(*CTX*) then delivers the decoded symbol *S* as is described in detail in subclause 10.qq.xx-10.qq.yy.

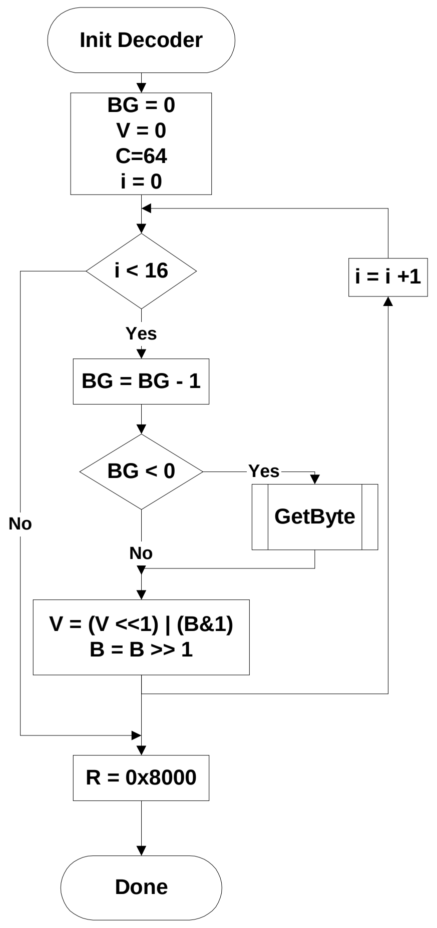


Figure 10-6– Flowchart of initialisation of the decoding engine

10.6.3.1 Initialisation of the Decoding Engine

In the initialisation procedure of the decoder, as illustrated in Figure 10-qq, *V* is first filled with two bytes of the compressed data using the GetByte routine as specified in subclause 10.qq.xx.yy. Then, the range *R* is set to 0x8000 and finally a counter *C* is initialized to -64.

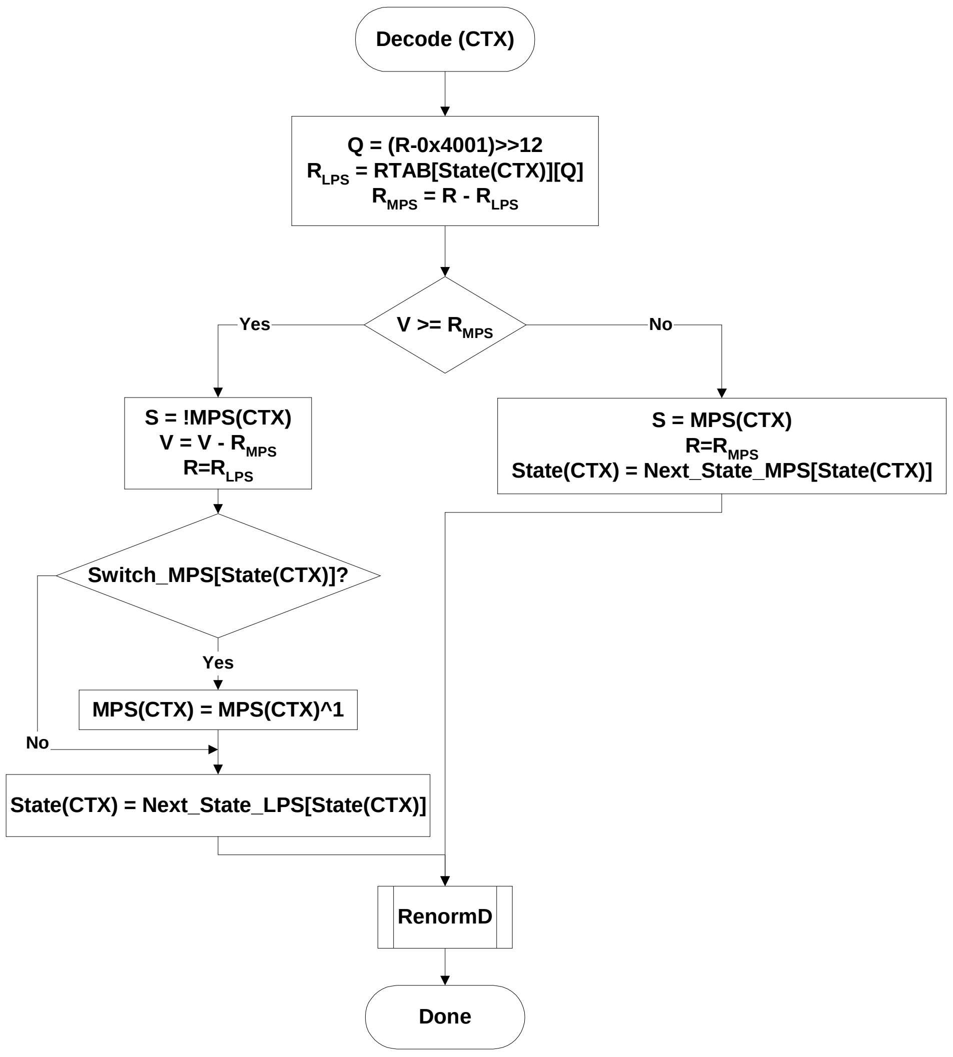


Figure 10-7– Flowchart for decoding a decision

10.6.3.2 Decoding a Decision

Figure 10-qq shows the flowchart for decoding a single decision. In a first step, the estimation of the sub-interval ranges R_{LPS} and R_{MPS} 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

$$Q=(R\text{-}0x4001)>>12, \qquad (10\text{-}qq)$$

such that the state index *State* and *Q* are used as an entry in the look-up table RTAB to determine R_{LPS}:

$$R_{LPS}=RTAB[State][Q]. \qquad (10\text{-}qq)$$

Table 10-qq contains 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_{MPS}. If V is greater than or equal to R_{MPS}.a MPS is decoded and the new range *R* is determined to be R_{MPS}; otherwise a LPS is decoded, V is decremented by R_{MPS}.and the new range *R* is set to R_{LPS}. Given the decoded decision, the probability update is performed accordingly as specified in subclause 10.qq.xx.yy. Depending on the current value of the new range *R*, renormalization will be performed as described in more detail in subclause 10.qq.xx.yy.

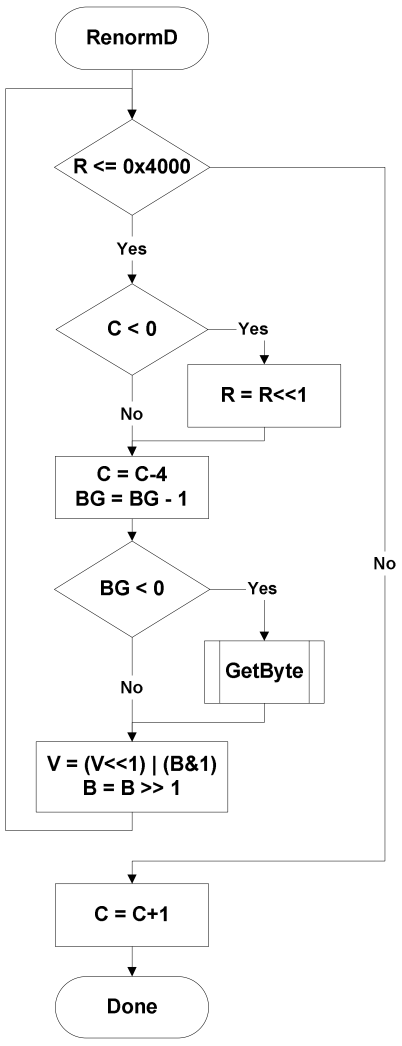


Figure 10-8– Flowchart of renormalization

10.6.3.3 Renormalization in the Decoding Engine (RenormD)

Renormalization is illustrated in Figure 10-qq. 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 by incrementing *C* by 1; otherwise the renormalization loop is entered. Within this loop, it is first tested whether the counter *C* has a negative value. If this condition is true, the range *R* is doubled, i.e. left-shifted by 1. In any case, the next step consists of decrementing the counter *C* by 4 and the bit-counter *BG* 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* and the counter *C* is incremented by 1.

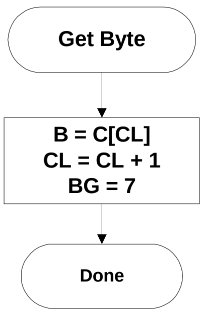


Figure 10-8– Flowchart for Input of Compressed Bytes

10.6.3.4 Input of Compressed Bytes (GetByte)

Figure 10-qq 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.

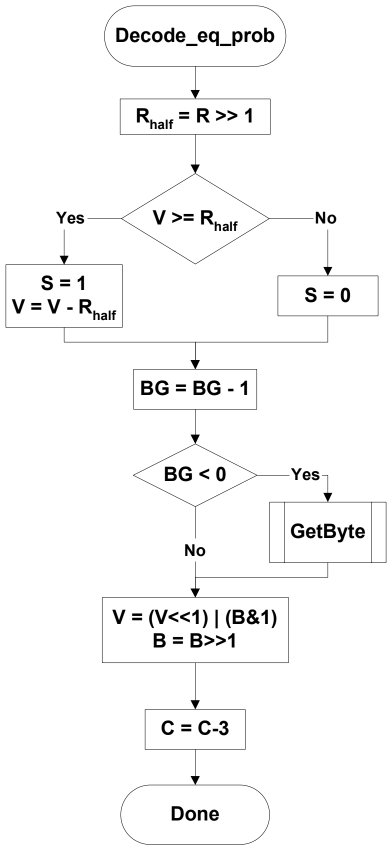


Figure 10-9– Flowchart of decoding bypass

10.6.3.5 Decoder Bypass for Decisions with Uniform PDF (Decode_eq_prob)

This special decoding routine applies to encoding 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 10-qq. with three modifications. Firstly, the rescaling operation *R←(R<<1)* needs not to be done; secondly, the initial comparison (*R<=0x4000?)* can be omitted, and thirdly, the counter *C* is simply decremented by 3.

11 B pictures

11.1 Introduction

The use of B (bi-predictive) pictures is indicated in PTYPE. A B picture is a predictive-coded picture. The substantial difference between a B picture and P picture is that B pictures are coded in a manner in which some macroblocks or blocks may use a weighted average of two distinct motion-compensated prediction values for building the prediction signal. Generally, B pictures utilize two distinct reference frame buffers: a forward reference frame buffer and a backward reference frame buffer. It should be noted that the term “forward reference frame buffer” does not indicate that only temporally preceding pictures are stored in this buffer. The same is true for the backward reference frame buffer. The terms “forward” and “backward” reflect both a historical and most common practice of the two reference frame buffers that have a more general application in this specification. Which pictures are actually located in each reference frame buffer is an issue of the reference frame buffer control (see sec. 8.2 and 9.1).

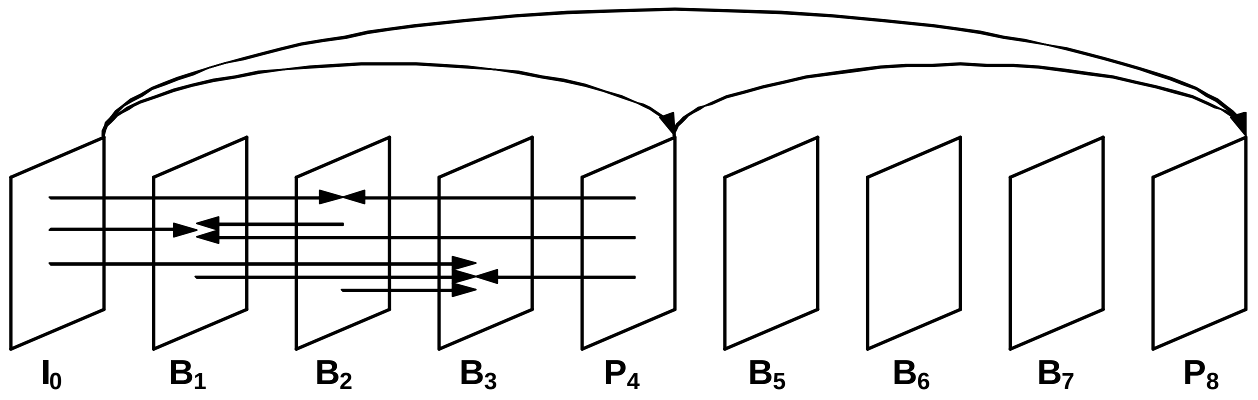


Figure 11-1 – Illustration of B-picture concept

The location of pictures in the bitstream is in a data-dependence order rather than in temporal or display order. Pictures that are dependent on other pictures shall occur in the bitstream after the pictures on which they depend. Figure 11-1 shows a typical example, where three B-pictures are inserted in display order between two successive P-pictures. The P-picture P4 only depends on the first Intra picture I0. The B-picture B2, which is temporally located between I0 and P4, depends on both of these pictures. The B picture B1 depends on I0, P4, and B2; the B picture B3 additionally depends on B1. While the display order for this example is given by I0, B1, B2, B3, P4, …, the only possible transmission order is I0, P4, B2, B1, B3, …

11.2 Macroblock modes and 8x8 sub-partition modes for B-picture

There are five different prediction types supported by B-pictures. They are the forward, backward, bi-predictive, direct, and intra prediction modes. While forward prediction indicates that the prediction signal is formed by utilizing motion compensation from a picture of the forward reference frame buffer, a picture of the backward reference frame buffer is used for building the prediction signal if backward prediction is used. As mentioned above, forward and backward prediction are not restricted to prediction from temporally preceding or subsequent pictures, respectively. Both the direct mode and the bi-predictive mode are bi-predictive prediction modes. The prediction signal is formed by a weighted average of a forward and backward prediction signal. The only difference is that the bi-predictive mode has separate encoded reference frame parameters and motion vectors for forward and backward prediction, whereas the reference frame parameters, as well as the forward and backward motion vectors of the direct mode, are derived from the motion vectors used in the corresponding macroblock of the picture, which is stored at reference index 0 in the backward reference frame buffer. In the direct mode, the same number of motion vectors is used as in the co-located macroblock of the picture that is stored at index 0 in the backward reference frame buffer. To calculate prediction blocks for the direct and bi-predictive prediction mode, the forward and backward motion vectors are used to obtain appropriate blocks from the corresponding reference pictures and then these blocks are averaged by dividing the sum of the two prediction blocks by two. For the bi-predictive modes, the forward and backward prediction signals is always build by utilizing motion compensated prediction form pictures of the forward and backward reference frame buffer, respectively. These prediction signals could even be formed from the same reference picture, if both reference buffers have been assigned to the same picture. Intra predicted macroblocks are spatially predicted, and coded with one of the intra modes.

B-pictures utilize a similar tree-structured macroblock partitioning to P-pictures. Depending on the size of the reference frame buffers, up to two reference frame parameters are transmitted for each 16x16, 16x8, and 8x16 block as well as for any 8x8 sub-partition. Additionally, for each 16x16, 16x8, 8x16 block, and each 8x8 sub-partition, the prediction direction (forward, backward, bi-predictive) can be chosen separately. For avoiding a separate code word to specify the prediction direction, the indication of the prediction direction is incorporated into the codewords for macroblock modes and 8x8 partitioning modes, respectively, as shown in the table Table 11-1 and Table 11-2. An 8x8 sub-partition of a B-picture macroblock can also be coded in direct mode.

Table 11-1 – Macroblock modes for B-pictures

Code number	Macroblock mode	1. block	2. block
0	Direct		
1	16x16	Forw.	
2	16x16	Backw.	
3	16x16	Bidirect.	
4	16x8	Forw.	Forw.
5	8x16	Forw.	Forw.
6	16x8	Backw.	Backw.
7	8x16	Backw.	Backw.
8	16x8	Forw.	Backw.
9	8x16	Forw.	Backw.
10	16x8	Backw.	Forw.
11	8x16	Backw.	Forw.
12	16x8	Forw.	Bidirect.
13	8x16	Forw.	Bidirect.
14	16x8	Backw.	Bidirect.
15	8x16	Backw.	Bidirect.
16	16x8	Bidirect.	Forw.
17	8x16	Bidirect.	Forw.
18	16x8	Bidirect.	Backw.
19	8x16	Bidirect.	Backw.
20	16x8	Bidirect.	Bidirect.
21	8x16	Bidirect.	Bidirect.
22	8x8(split)		

23	MBIntra4x4		
24 …	MBIntra16x16		

Table 11-2 – Modes for 8x8 sub-partitions in B-pictures

Code number	8x8 partition mode	Prediction
0	Direct	
1	8x8	Forw.
2	8x8	Backw.
3	8x8	Bidirect.
4	8x4	Forw.
5	4x8	Forw.
6	8x4	Backw.
7	4x8	Backw.
8	8x4	Bidirect.
9	4x8	Bidirect.
10	4x4	Forw.
11	4x4	Backw.
12	4x4	Bidirect.
13	BlockIntra4x4	

11.3 B-Picture Syntax

Some additional syntax elements are needed for B-pictures. The macroblock layer syntax for B-pictures is shown in Figure 11-2. The fields ref_idx_fwd, ref_idx_bw, mvd_fwd, and mvd_bwd are inserted to enable bi-predictive (and the more general multi-hypothesis) prediction. These fields replace the syntax elements ref_idx and mvd_fwd.

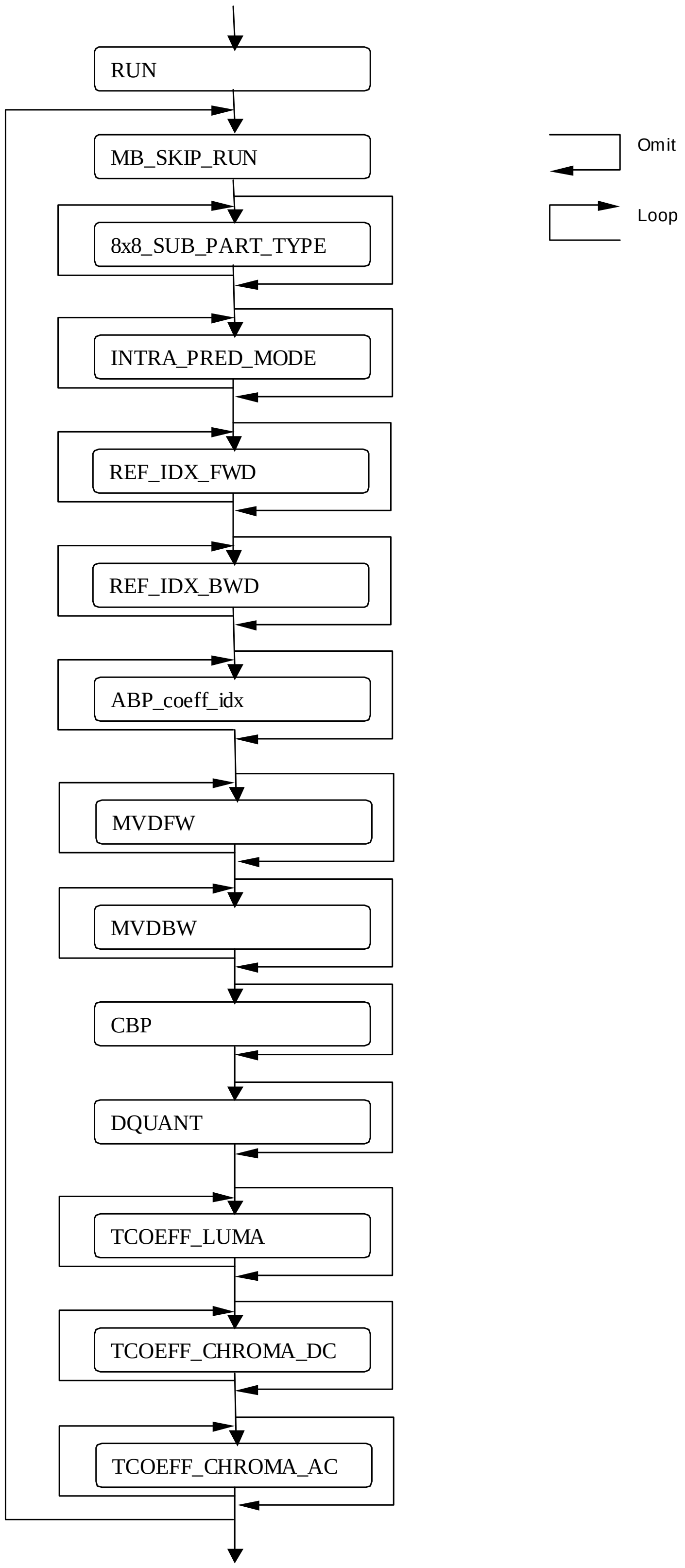


Figure 11-2 – Macroblock syntax diagram for B-pictures

11.3.1 Number of Skipped macroblocks (mb_skip_run)

The syntax element mb_skip_run specifies the number of skipped macroblocks in coding order. For a B-picture macroblock, skip means direct mode without coefficients. See subclause 8.3.1.

11.3.2 Macroblock type (MB_type) and 8x8 sub-partition type

Table 11-1 shows the macroblock modes for B-pictures.

In “Direct” prediction type, no motion vector data is transmitted.

In NxM mode, each NxM block of a macroblock is predicted by using different motion vectors, reference pictures, and prediction directions. As indicated in Table 11-11, three different macroblock modes that differ in their prediction directions exist for the 16x16 mode. For the 16x8 and 8x16 macroblock modi, 9 different combinations of the prediction directions are possible. If a macroblock is transmitted 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).

The “MBintra4x4” and “MBintra16x16” prediction type indicates that the macroblock is encoded by intra coding with different intra prediction modes which are defined in the same manner as in subclause 8.4.2 and Table 9-4. No motion vector data is transmitted for intra macroblocks or 8x8 sub-partitions coded in intra mode.

11.3.3 Intra prediction mode (Intra_pred_mode)

As present, Intra_pred_mode indicates which intra prediction mode is used. Intra_pred_mode is present when MBintra4x4 prediction type is indicated in the MB_type or BlockIntra4x4 is indicated in region8x8_type. The code_number is the same as that described in the Intra_pred_mode entry of Table 9-4.

11.3.4 Reference picture parameters (ref_idx_fwd and ref_idx_bwd)

The ref_idx_fwd parameter indicates the position of the reference picture in the forward reference set to be used for forward motion-compensated prediction of the current macroblock. The ref_idx_fwd parameter is transmitted for each 16x16, 16x8, 8x16 block, and each 8x8 sub-partition if forward or bi-predictive prediction is used and if multi-frame forward prediction is in use. ref_idx_fwd is present only when the PTYPE signals the use of multiple forward reference pictures. The code_number and interpretation for ref_idx_fwd differs for picture_struct indicating field picture or frame picture and is the same as the definition for ref_idx_fwd in subclause 8.4.5.

Similarly, ref_idx_bwd is the index into the backward reference set to determine which frame or field is used for backward prediction when multi-frame backward prediction is in use. ref_idx_bwd is present only when PTYPE indicates the use of multiple backward reference pictures. ref_idx_bwd is used in the same manner as ref_idx_fwd, but is an index into the backward reference set rather than the forward reference set.

11.3.5 ABP coefficient index (abp_coeff_idx)

If the explicit B prediction block weighting is in use, abp_coeff_idx indicates the ABP coefficient index. If number_of_abp_coeff_minus_1 is 0, then abp_coeff_idx is not sent and regarded as 0. For the skipped macroblock, the ABP coefficient corresponding to abp_coeff_idx=0 is used.

11.3.6 Motion vector data (mvd_fwd, mvd_bwd)

mvd_fwd is the motion vector data for the forward vector, if present. mvd_bwd is the motion vector data for the backward vector, if present. If so indicated by MB_type and/or 8x8 sub-partition type, vector data for 1-16 blocks are transmitted. The order of transmitted motion vector data is the same as that indicated in Figure 2. For the code_number of motion vector data, please refer to Table 9-4.

11.4 Decoder Process for motion vector

11.4.1 Differential motion vectors

Motion vectors for forward, backward, or bi-predictively predicted blocks are differentially encoded. A prediction has to be added to the bitstream motion vector differences in order to reconstruct motion vectors for the current macroblock. If the two reference pictures used for the bi-prediction exist in the same direction (either forward or backward), the motion vector (mvd_fwd or mvd_bwd) for the farthest reference picture is coded using the differential motion vectors from the scaled motion vector for the nearest reference picture as described in 11.4.2. Otherwise, the predictions are formed in way similar to that described in section 4.4.6. The only difference is that forward motion vectors are predicted only from forward motion vectors in surrounding macroblocks, and backward motion vectors are predicted only from backward motion vectors in surrounding macroblocks. # [Ed note: what does ‘surrounding’ mean ? Spatially and\or temporally adjacent macroblocks ? Also, without a TR-like mechanism, the decoder does not know which of the two DPCM motion vector coding rules to follow above. A simple replacement for TR is needed -- CF]

Reconstructed motion vectors in direct-mode predicted macroblocks shall be used as prediction for neighbouring macroblocks the same as bi-predictive coded macroblocks. When the co-located macroblock is intra coded, the macroblock is treated as having a different reference from the current block/macroblock.

If a neighbouring macroblock does not have a respective forward and/or backward motion vector candidate predictor for the current block, the predictor is set to zero

11.4.2 Motion vector decoding with scaled MV

The motion vector decoding process is illustrated in figure x-x. First, the scaled motion vector (scaled mv) is calculated from the coded motion vector for the nearest reference picture r1 (MV1). Then, the motion vector for the farthest reference picture r2 (MV2) is calculated by adding the coded differential motion vector (DMV) to the scaled mv.

$$MV_2 = picture\_distance\_1 \cdot MV_1 / picture\_distance\_2 + DMV$$

where picture_distance_1 and picture_distance_2 are motion vector scaling factors for the nearest reference picture (r1) and the farthest reference picture (r2), respectively. picture_distance_1 and picture_distance_2 are sent in the picture header.

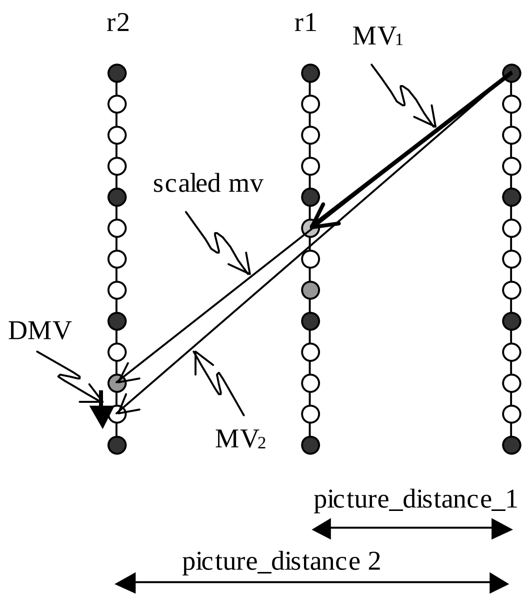


Figure 11-3 Motion vector scaling and differential MV coding

11.4.3 Motion vectors in direct mode

In direct mode the same block structure as for the co-located macroblock in the picture with index 0 of the backward reference set is used. For each of the sub-blocks the forward and backward motion vectors are computed as scaled versions of the corresponding vector components of the co-located macroblock in the first picture (index0) of the backward reference set as described below.

If the multiple reference frame prediction is used, the forward reference picture for the direct mode is the same as the one used for the corresponding macroblock in the picture with index 0 of the backward reference set. Also note that if the picture with index 0 of the backward reference set is an intra-coded picture or the reference macroblock is an intra-coded block or it doesn’t use forward prediction, the motion vectors are set to zero. With possible adaptive switch of frame/field coding at picture level, a B-picture can be coded in either frame structure or field structure.

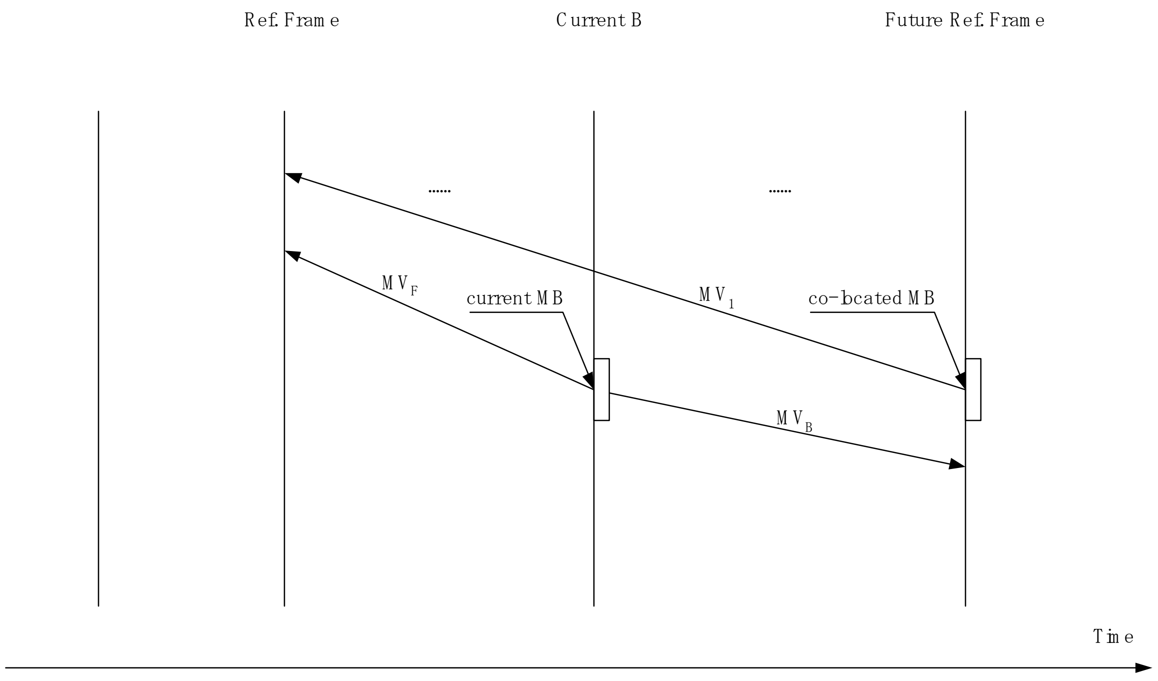


Figure 11-4 – Motion vectors in direct mode

$$MV_F = direct\_mv\_scale\_fwd \cdot MV_1 / direct\_mv\_divisor$$
$$MV_B = direct\_mv\_scale\_bwd \cdot MV_1 / direct\_mv\_divisor$$

direct_mv_scale_fwd, direct_mv_scale_bwd and direct_mv_divisor are sent in picture_layer_rbsp().

11.5 Prediction signal generation procedure

11.5.1 Implicit B Prediction Block Weighting

Prediction signal generation procedure is specified in the following table. If display order of ref_fwd <= display order of ref_bwd, ABP coefficient (1/2, 1/2, 0) is used. Otherwise, (2, -1, 0) is used.

If the reference picture is long-term picture, whose display order is regarded as earlier than all short-term pictures and the picture having larger the default relative index value is regarded as earlier in display order.

Table 11-3 – Prediction signals for implicit bi-prediction weighting

ABP coefficient	prediction signal
(1/2, 1/2, 0)	(P1 + P2)/2
(2, -1, 0)	clip1(2*P1 - P2)

11.5.2 Explicit B Prediction Block Weighting

Prediction signal is generated as the following form:

$$P = clip1\left(\frac{P_1 \times FWF + P_2 \times SWF}{2^{LWD}} + D \right)$$

where

P = ABP prediction signal

$$FWF = \begin{cases} -FWFM, & if \quad FWFS = 0 \\ FWFM, & otherwise \end{cases}$$

$$SWF = \begin{cases} -SWFM, & if \;\; SWFS = 0 \\ SWFM, & otherwise \end{cases}$$

$$D = \begin{cases} -DM, & if \;\; DS = 0 \\ DM, & otherwise \end{cases}$$

P1 = Reference signal corresponding to the reference index ref_idx1

P2 = Reference signal corresponding to the reference index ref_idx2

FWFM = first_weight_factor_magnitude[abp_coeff_idx]

FWFS = first_weight_factor_sign[abp_coeff_idx]

SWFM = second_weight_factor_magnitude[abp_coeff_idx]

SWFS = second_weight_factor_sign[abp_coeff_idx]

DM = constant_factor_magniture[abp_coeff_idx]

DS = constant_factor_sign[abp_coeff_idx]

LWD = logarithmic_weight_denominator[abp_coeff_idx]

To limit the calculation to 16-bit precision, the following conditions shall be met:

$$-128 \le FWF \le 128$$
$$-128 \le SWF \le 128$$
$$-128 \le FWF + SWF \le 128$$

12 SP-pictures

There are two types of S-pictures, namely SP-pictures and SI-pictures. SP-pictures make use of motion-compensated predictive coding to exploit temporal redundancy in the sequence similar to P-pictures and SI-pictures make use of spatial prediction similar to I-pictures. Unlike P-pictures, however, SP-picture coding allows identical reconstruction of a frame even when different reference frames are being used. SI-picture, on the other hand, can identically reconstruct a corresponding SP-picture. These properties of S-pictures provide functionalities for bitstream switching, splicing, random access, VCR functionalities such as fast-forward and error resilience/recovery.

12.1 Syntax

The use of S-pictures is indicated by PTYPE. If PTYPE indicates an SP-picture, a flag is used to indicate it is a transition SP or not. If PTYPE indicates an S-picture, quantisation parameter slice_delta_qp is followed by an additional quantisation parameter delta_qp_sp in the slice header, see Section QQ. The rest of the syntax elements for SP-pictures are the same as those in P pictures. Similarly, the rest of the syntax elements for SI-pictures are the same as those in I-pictures.

12.1.1 Picture type (Ptype) and RUN

See Section QQ for definition.

12.1.2 Macro block type (MB_Type)

MB_Type indicates the prediction mode and the block size used to encode each macroblock. There are different MB types for SP and SI-pictures.

12.1.3 Macroblock modes for SI-pictures

The MB types for SI-pictures are similar to those of I-pictures with an additional MB type, called SIntra 4x4. Table 12-1 depicts the relationship between the code numbers and MB_Type for SI-pictures.

SIntra 4x4	SI Mode.
Intra 4x4	4x4 Intra coding.

Imode, nc, AC.

See definition in Section QQ. These modes refer to 16x16 intra coding.

Table 12-1 – MB Type for SI-Pictures

Code_number	MB_Type (SI-pictures)
0	SIntra 4x4
1	Intra 4x4
2	0,0,0
3	1,0,0
4	2,0,0
5	3,0,0
6	0,1,0
7	1,1,0
8	2,1,0
9	3,1,0
10	0,2,0
11	1,2,0
12	2,2,0
13	3,2,0
14	0,0,1
15	1,0,1
16	2,0,1
17	3,0,1
18	0,1,1
19	1,1,1
20	2,1,1
21	3,1,1
22	0,2,1
23	1,2,1
24	2,2,1
25	3,2,1

1 16x16 based intra mode. The 3 numbers refer to values for (Imode,AC,nc) - see QQ.

12.1.4 Macroblock Modes for SP-pictures

The MB types for SP-pictures are identical to those of P-pictures, see Section QQ and Table 9-4. However, all of the inter mode macroblocks, i.e., Skip and NxM, in SP frames refer to SP mode.

12.1.5 Intra Prediction Mode (Intra_pred_mode)

Intra_pred_mode is present when Intra 4x4 or SIntra 4x4 prediction types are indicated by the MB_Type. Intra_pred_mode indicates which intra prediction mode is used for a 4x4 block in the macroblock. The code_number for SIntra 4x4 is the same as that described in the Intra_pred_mode entry of Table 9-4.

Coding of Intra 4x4 prediction modes

Coding of the intra prediction modes for SIntra 4x4 blocks in SI-pictures is identical to that in I-pictures, i.e., the intra prediction modes of the neighbouring blocks are taken into account as described in Section 3.4.3.1.1. Coding of the intra prediction modes for Intra 4x4 blocks in SI-pictures differs from I-picture coding if the neighbouring block is coded in SIntra 4x4 mode. In this case the prediction mode of the neighbouring Sintra 4x4 block is treated to be "mode 0: DC_prediction".

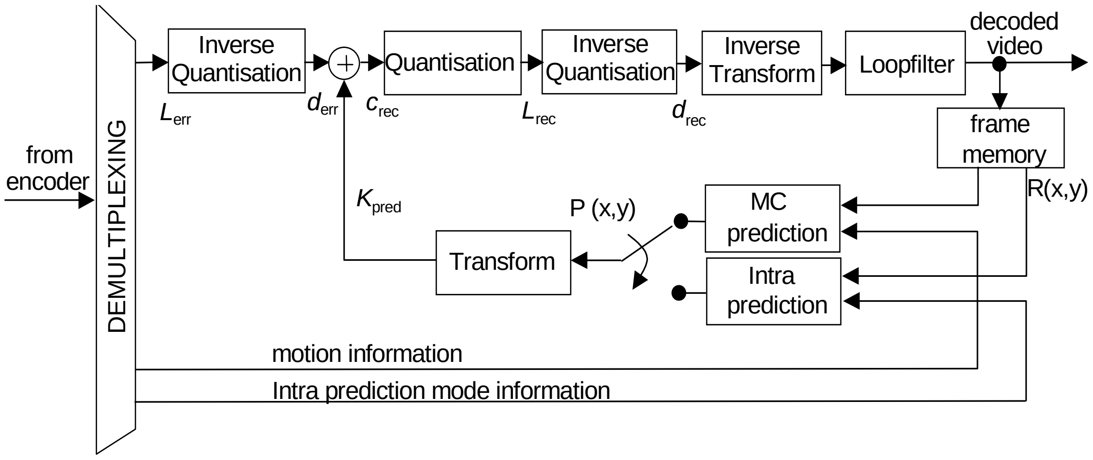


Figure 12-1 – A generic block diagram of S-picture decoder

12.2 S-frame decoding

A video frame in SP-picture format consists of blocks indicated in either Intra mode (Intra 4x4, Intra 8x8 or Intra 16x16 Modes) or in SP mode (Skip or NxM). Similarly, SI-pictures consist of macroblocks indicated in either Intra mode or in SI mode (SIntra 4x4). Intra macroblocks are decoded identically to those in I and P pictures. Figure 26 depicts a generic S-picture decoding for SI and SP modes. In SP mode, the prediction P(x,y) for the current macroblock of the frame being decoded is formed by the motion compensated prediction block in the identical way to that used in P-picture decoding. In SI mode, the prediction P(x,y) is formed by intra prediction block in the identical way to that used in I-picture decoding. After forming the predicted block P(x,y), the decoding of SI and SP-macroblocks follows the same steps. Firstly, the received prediction error coefficients denoted by Y_{QERR} are inverse quantised using quantisation parameter QP_Y and the results are added to the transform coefficients Y_{PRED} of predicted block which are found by applying forward transform, see Section QQ, to the predicted block. The resulting sum is quantised with the quantisation parameter $QPSP_Y$.

$$Y_{QREC}(i,j) = ((Y_{PRED}(i,j) + Y_{QERR} \cdot T(QP_Y,i,j)) \cdot Q(QPSP_Y\%6,i,j) + f) >> 2^{15+QPSP_Y/6} \quad i,j = 0,\dots,3 \tag{11-1}$$

where T( ) is a 16-bit integer table generated with the following formula,

$$T(QP_Y,i,j) = \frac{2^{15+QP_Y/6} + Q(QP_Y\%6,i,j)/2}{Q(QP_Y\%6,i,j)} \tag{11-2}$$

The values of f and the coefficients Q(m,i,j) are defined in Subsection 5.3.3.3. The coefficients Y_{QREC} are then inverse quantised using $QP=QPSP_Y$ and the inverse transform process is performed for these inverse quantised levels, as defined in Subsections 0 and 5.3.1, respectively. Finally, after the reconstruction of a macroblock, filtering of this macroblock takes place as described in subclause 8.2.2.

12.2.1 Decoding of DC values of chroma

The decoding of chroma components for SP- and SI-macroblocks is similar to the decoding of luma components described in subclause 8.2. AC coefficients of the chroma blocks are decoded following the steps described in the earlier subclause 8.2 where the quantisation parameters QP_Y and $QPSP_Y$ are replaced by QP_C and $QPSP_C$ according to the relation between the quantisation values of luma and chroma, as specified in Section QQ. As described in Section QQ, the coding of DC coefficients of chroma components includes an additional 2x2 transform. Similarly, for SP and SI-macroblocks, an additional 2x2 transform is applied to the DC coefficients of the predicted 4x4 chroma blocks and the steps described in subclause 8.2 are applied to the 2x2 transform coefficients.

12.2.2 Deblocking Filter

When applying deblocking filter for macroblocks in S-frames, all macroblocks are treated as Intra macroblocks as described in Section QQ.

13 Hypothetical Reference Decoder

13.1 Leaky Bucket Model

The hypothetical reference decoder (HRD) is a mathematical model for the decoder, its input buffer, and the channel. The HRD is characterized by the channel's peak rate R (in bits per second), the buffer size B (in bits), and the initial decoder buffer fullness F (in bits). These parameters represent levels of resources (transmission capacity, buffer capacity, and delay) used to decode a bitstream.

A closely related object is the leaky bucket (LB), which is a mathematical constraint on a bitstream. A leaky bucket is characterized by the bucket's leak rate R_1 (in bits per second), the bucket size B_1 (in bits), and the initial bucket fullness $B_1^{-}F_1$ (in bits). A given bitstream may be constrained by any number of leaky buckets $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, $N\geq1$. The LB parameters for a bitstream, which are indicated in the bitstream header, precisely describe the minimum levels of the resources R, B, and F that are sufficient to guarantee that the bitstream can be decoded.

13.2 Operation of the HRD

The HRD input buffer has capacity B bits. Initially, the buffer begins empty. At time t_{start} it begins to receive bits, such that it receives $S(t)$ bits through time t. $S(t)$ can be regarded as the integral of the instantaneous bit rate through time t. The instant at which $S(t)$ reaches the initial decoder buffer fullness F is identified as the decoding time t_0 of the first picture in the bitstream. Decoding times t_1, t_2, t_3, …, for subsequent pictures (in bitstream order) are identified relative to t_0, per subclause 13.3. At each decoding time t_i, the HRD instantaneously removes and decodes all d_i bits associated with picture i, thereby reducing the decoder buffer fullness from b_i bits to $b_i - d_i$ bits. Between time t_i and t_{i+1}, the decoder buffer fullness increases from $b_i - d_i$ bits to $b_i - d_i + [S(t_{i+1}) - S(t_i)]$ bits. That is, for $i \geq 0$,

$$b_0 = F \tag{13-1}$$

$$b_{i+1}= b_i - d_i + [S(t_{i+1}) - S(t_i)]. \tag{13-2}$$

The channel connected to the HRD buffer has peak rate R. This means that unless the channel is idle (whereupon the instantaneous rate is zero), the channel delivers bits into the HRD buffer at instantaneous rate R bits per second.

$$b_0 = F \tag{13-3}$$

$$b_{i+1}= b_i - d_i + [S(t_{i+1}) - S(t_i)]. \tag{13-4}$$

13.3 Decoding Time of a Picture

The decoding time t_i of picture i is equal to its presentation time τ_i, if there are no B pictures in the sequence. If there are B pictures in the sequence, then $t_i = \tau_i - m_i$, where $m_i = 0$ if picture i is a B picture; otherwise m_i equals $\tau_i - \tau_i'$, where τ_i' is the presentation time of the I or P picture that immediately precedes picture i (in presentation order). If there is no preceding I or P picture (i.e., if $i = 0$), then $m_i = m_0 = t_1 - t_0$. The presentation time of a picture is determinable from its temporal reference and the picture clock frequency.

13.4 Schedule of a Bitstream

The sequence (t_0,d_0), (t_1,d_1), (t_2,d_2), … 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. A bitstream may be pre-indicated, stored to a file, and later transmitted over channels with different peak bit rates to decoders with different buffer sizes. The schedule of the bitstream is invariant over such transmissions.

13.5 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_0,d_0), (t_1,d_1), (t_2,d_2), … 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

$$b_0 = B_1–F_1 \tag{13-5}$$

$$b_{i+1} = \max\{0, b_i + d_i – R_1(t_{i+1}–t_i)\}. \tag{13-6}$$

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 the Figure below. Note from the Figure that the same bitstream is containable in more than one leaky bucket. Indeed, a bitstream is containable in an infinite number of leaky buckets.

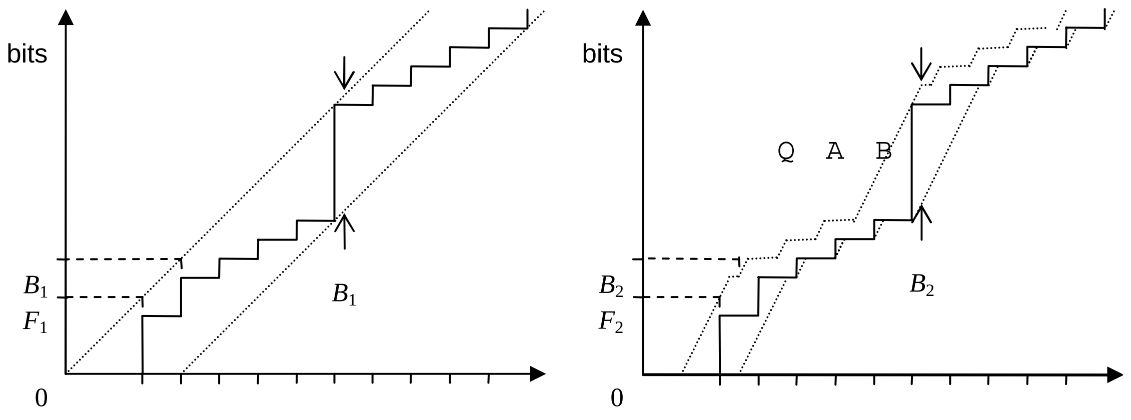


Figure 13-1 – Illustration of the leaky bucket concept

If a bitstream is contained in a leaky bucket with parameters (R_1,B_1,F_1), then when it is communicated over a channel 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.

13.6 Bitstream Syntax

The header of each bitstream shall specify the parameters of a set of $N \geq 1$ leaky buckets, (R_1,B_1,F_1),…,(R_N,B_N,F_N), each of which contains the bitstream. In the current Test Model, these parameters are specified in the first 1+3N 32-bit integers of the Interim File Format, in network (big-endian) byte order:

$$N, R_1, B_1, F_1, \ldots, R_N, B_N, F_N. \tag{13-7}$$

The R_n shall be in strictly increasing order, and both B_n and F_n shall be in strictly decreasing order.

These parameters shall not exceed the capability limits for the profile and level.

13.7 Minimum Buffer Size and Minimum Peak Rate

If a bitstream is contained in a set of leaky buckets with parameters (R_1,B_1,F_1), …, (R_N,B_N,F_N), then when it is communicated over a channel with peak rate R, it is decodable (i.e., the HRD buffer does not overflow or underflow) provided $B \geq B_{min}(R)$ and $F \geq F_{min}(R)$, where for $R_n \leq R \leq R_{n+1}$,

$$B_{min}(R) = \alpha B_n + (1 - \alpha)B_{n+1} \tag{13-8}$$

$$F_{min}(R) = \alpha F_n + (1 - \alpha)F_{n+1} \tag{13-9}$$

$$\alpha = (R_{n+1} - R) / (R_{n+1} - R_n). \tag{13-10}$$

For $R \leq R_1$,

$$B_{min}(R) = B_1 + (R_1 - R)T \tag{13-11}$$

$$F_{min}(R) = F_1, \tag{13-12}$$

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 $R \geq R_N$,

$$B_{min}(R) = B_N \tag{13-13}$$

$$F_{min}(R) = F_N\,. \tag{13-14}$$

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 $\geq R_{min}(B)$ and $F \geq F_{min}(B)$, where for $B_n \geq B \geq B_{n+1}$,

$$R_{min}(B) = \alpha R_n + (1 - \alpha)R_{n+1} \tag{13-15}$$

$$F_{min}(B) = \alpha F_n + (1 - \alpha)F_{n+1} \tag{13-16}$$

$$\alpha = (B - B_{n+1}) / (B_n - B_{n+1}). \tag{13-17}$$

For $B \geq B_1$,

$$R_{min}(B) = R_1 - (B - B_1)/T \tag{13-18}$$

$$F_{min}(B) = F_1. \tag{13-19}$$

For $B \leq B_N$, 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 13-12. The minimum start-up delay necessary to maintain this guarantee is $F_{min}(R)$ / R.

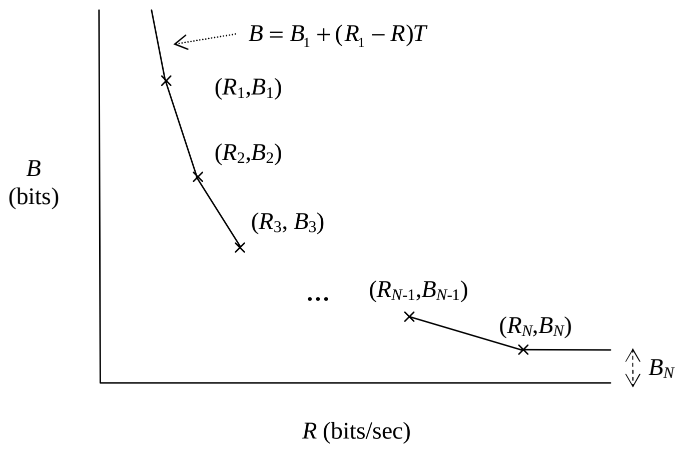


Figure 13-2 – Illustration of the leaky bucket concept

A compliant decoder with buffer size B and initial decoder buffer fullness F that is served by a channel with peak rate R shall perform the tests $B \geq B_{min}(R)$ and $F \geq F_{min}(R)$, as defined above, for any compliant bitstream with LB parameters $(R_1,B_1,F_1),\ldots,(R_N,B_N,F_N)$, and shall decode the bitstream provided that $B \geq B_{min}(R)$ and $F \geq F_{min}(R)$.

13.8 Encoder Considerations (informative)

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 Section 13.5.

13.9 Normative Hypothetical Decoder and Buffering Verifiers

The Hypothetical Reference Decoder represents a set of normative constraints on coded streams. These constraints must be enforced by an encoder, and can be assumed to be true by a decoder or multiplexor. The following description should not be interpreted as guidance for decoder implementers particularly with regard to decode and presentation timing.

The HRD can contain any or all of the following Buffering Verifiers, as shown in Figure 13.1:

- One or more VBR-VBV Buffers
- At most one CBR-VBV Buffer
- At most one TBV Buffer, if at least one VBV Buffer is present

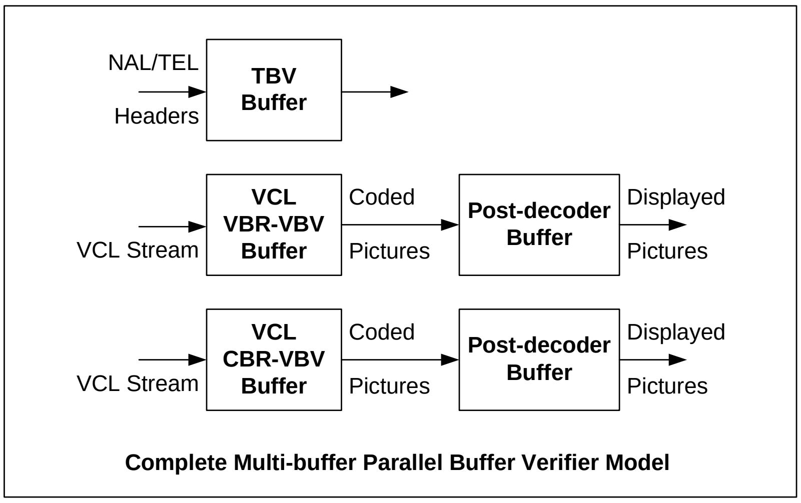


Figure 13-3 – HRD Buffer Verifiers

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 the VBV buffer just prior to or after a removal time is not necessarily an integer. Furthermore, while compliance is guaranteed assuming that all frame-rates and clocks used to generate the Video Elementary Stream match exactly the values signaled in the syntax, each of these may vary from the signaled or defined value according to normative rules. Real systems responsible for multiplexing, editing, splicing, playing back, etc. these streams must account for the deviation of actual values from the signaled or defined ones.

13.9.1 Operation of Pre-Decoder VBV Buffer Verifier

13.9.1.1 Timing of bitstream/packet arrival

This specification applies independently to each VBV Buffer signaled. The buffer is initially empty. If the VBV buffer is not full, data enters the buffer at the bit-rate associated with that VBV buffer. If a VBR-VBV buffer becomes full after filling at its associated data rate for some time, no more data enters the buffer until some data is removed from the buffer.

13.9.1.2 Timing of Coded Picture Removal

In both low-delay and delay-tolerant modes, for the first picture and all pictures which are the first complete picture after receiving a Buffering Period SEI Message, no data the coded data associated with the picture is is removed from the VBV buffer during after a time period equal to the following:

rrts_xxx (13.1)

where rrts_xxx is the relative removal time stamp corresponding to the VBV buffer.

After the first picture is removed, the buffer is examined at subsequent points of time.

In the low-delay buffering mode, the following behavior holds. The subsequent points of time are increments of the field period. If at least one complete coded picture is in the buffer and if at least one picture memory buffer is free in the post-decoder buffer, then all the data for the earliest complete picture in coding order is removed from buffer.

In the delay-tolerant mode, the following behavior holds. All pictures must be coded. If T(n-1) represents the removal time of the previous picture, then the following pseudocode defines T(n).

```
// field_period and frame_period have the expected meaning, and are
// defined in Section 3
if ((pstruct==1) || (pstruct==2))      {     // Field picture
      P = field_period
}
else                                   // Frame picture
P = frame_period
```

```
T(n) = T(n-1) + P
if ( (film_state) && (film_mode==A or film_mode==C))
{
T(n) += P/2
}
```

In the low-delay buffering mode, if at least one complete coded picture is in the buffer and if at least one picture memory buffer is free in the post-decoder buffer, then all the data for the earliest complete picture in coding order is removed from buffer. At this time, the coded picture which has been in the buffer longest is removed from the VBV buffer at that time.

13.9.1.3 Compliance Constraints on Coded Bitstreams or Packet Streams

A transmitted or stored stream of coded data compliant with this Recommendation | International Standard fulfills either the low-delay mode requirements or the delay-tolerant mode requirements.

Delay-tolerant Mode Requirements

- Each picture must be available in the VBV buffer at its computed removal time.
- If VBVI_CBR==1, at no time shall the occupancy of the CBR VBV buffer exceed the buffer size.

Low-delay Mode Requirements

- After each removal, the time equivalent of the buffer occupancy (in ticks of a 90kHz clock) must be below rrts_max.

13.9.2 Operation of the TBV Buffer Verifier

The TEL Buffering Verifier operates synchronously with the VBV Buffer which has the earliest removal time. (After the initial delay at the start of sequence or after a Buffering Period SEI Message, the removal times of all VBV Buffers increment with the same values; they simply have different initial times.)

13.9.2.1 TBV Arrival Timing

All TEL bits preceding a given picture enter the TBV instantaneously when the first VCL byte associated with that picture enters the earliest VBV Buffer.

13.9.2.2 TBV Removal Timing

All TEL bits preceding a given picture are removed from the TBV when the corresponding picture is removed from the earliest VBV Buffer.

13.9.2.3 TBV Compliance Constraints

- TEL data associated with each picture must be available in the TBV buffer at its computed removal time.
- If TBVI =1, at no time shall the occupancy of the TBV buffer exceed the buffer size.

13.9.3 Operation of the Post-Decoder Buffer Verifier

13.9.3.1 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.

13.9.3.2 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.

13.9.3.3 Compliance Constraints

The occupancy of the post-decoder buffer shall not exceed the default or signaled buffer size.

Each picture shall be available in the post-decoder buffer before or on its presentation time.

14. Adaptive Block Size Transforms

14.1 Introduction

The concept of variable block size transform coding is described. The scheme is called Adaptive Block size Transforms (ABT) indicating the adaptation of the transform block size to the block size used for motion compensation. For Intra coding, the transform block size is adapted to the properties of the intra prediction signal. Hence, for both inter and intra coding, the maximum feasible signal length can be exploited by the transform. With ABT, the block transform as commonly used in the former image and video coding standards is generalized from a fixed-size transform to a signal adaptive tool for increased overall coding performance.

ABT is applied to frame macroblocks if frame motion compensation is used. ABT is applied to field macroblocks if field motion compensation is used. Therewith, inter ABT is always used on the blocks used for motion compensation.

First, the syntax changes necessary for ABT coding are given. Only for ABT intra coding, a new symbol is introduced into the macroblock layer syntax. For encoding of the transform coefficients, the existing syntax is used. The necessary modifications are described below. The syntax is given in tabular form. Then, the ABT decoding process is described.

14.2 ABT Syntax Elements

14.2.1 Slice Header Syntax

slice_header( ) {	Category	Descriptor
qq		
}		

14.2.2 Prediction data for 8x8 modes

prediction8x8(mb_type) {	Category	Descriptor
region8x8_type	4	ecselbf
if (region8x8_type = = BlockIntra4x4) {		
if (useABT > 1)		
intra_block_modeABT **/*** determines num_ipred */	4	ecselbf
for (i4x4=0; i4x4< num_ipred; i4x4++)		
intra_pred_mode	4	ecselbf
} else {		
….		

14.2.3 Prediction data for 16x16 modes

prediction16x16(mb_type) {	**Category**	**Descriptor**
if (mb_type = = MBintra4x4)		
if (useABT > 1)		
intra_block_modeABT **/*** determines num_ipred */	4	ecselbf
for (i4x4=0; i4x4< num_ipred; i4x4++)		
intra_pred_mode	4	ecselbf
else if (mb_type != MBintra16x16) {		
…		

14.2.4 Residual Data Syntax

residual(mb_type) {	Category	Descriptor
if (mb_type = = MBintra16x16)		
do		
luma_dc_zigzag_token	5 | 6	ecselbf
while (lum_dc_zigzag_token != EOB)		
for (i8x8=0; i8x8<4; i8x8++) /* each luma 8x8 region */		
/* ABT number of sub-blocks num_blks depends on block mode non-ABT: num_blks=4 */		
for (i4x4=0; i4x4< num_blks; i4x4++) /* each sub-block of region */		
if (cbp & (1<< i8x8)		
if (double_scan()) {		
do		
luma_dblscan0_token	5 | 6	ecselbf
while (luma_dblscan0_token != EOB)		
do		
luma_dblscan1_token	5 | 6	ecselbf
while (luma_dblscan1_token != EOB)		
} else		
do		
luma_zigzag_token	5 | 6	ecselbf
while (luma_zigzag_token != EOB)		
if (cbp & 0x30) { /* chroma DC residual coded */		
…		

14.3 ABT Decoding process

14.3.1 Luma intra coding

If useABT = = 2, ABT intra coding is used. ABT intra coding is very similar to 4x4 intra coding in Section QQ. The block size used for prediction is 4x4, 8x4, 4x8, or 8x8 pixels. The transform block size is chosen according to intra_block_modeABT. The DC prediction and the directional prediction mode defined in Section QQ are used for ABT intra blocks. MBintra16x16 is not used with ABT intra.

14.3.1.1 Prefiltered prediction

Before employed for prediction, the edge pixels of (available) neighbouring blocks are gathered in a prediction vector EP.

Let P denote the predicted block of N pixels in M lines. P[m,n] is the pixel at vertical position m, 0<=m<M, and horizontal position n, 0<=n<N. The pixels used for prediction shall be identified by R[m,n], with n=-1 & -1<=m and –1<n & m=-1. Only decoded pixels of the same slice are valid for prediction.

The prediction vector EP contains the reconstructed luma at valid positions of R. EP is constructed according to the following pseudo code:

```
offset = 0;
valid_left_down = valid(R[M:M+N-1,-1]);
valid_left      = valid(R[0:M-1,-1]);
valid_up        = valid(R[-1,0:N-1]);
valid_up_right  = valid(R[-1,N:N+M-1]);

if valid_left_down
{
  offset += N;
  for (k=0; k<N; k++)
    EP'[offset-k] = R[k,-1];
}

if valid_left
{
  offset += M;
  for (k=0; k<M; k++)
    EP'[offset-k] = R[k,-1];
  EP'[offset+1] = R[0,-1];
}

offset+=1;
if valid_up
{
  for (k=0; k<N; k++)
    EP'[offset+k+1] = R[-1,k];
  EP'[offset] = R[-1,0];
}

if valid_up_right
{
  for (k=N; k<N+M; k++)
    EP'[offset+k+1] = R[-1,k];
}

if (valid_left & valid_up)
  EP'[offset] = R[-1,-1];
```

To enhance prediction quality, EP' is filtered prior to prediction the following way:

$$EP[k] = (EP'[k-1] + 2*EP'[k] + EP'[k+1] + 2) >> 2, \quad (14\text{-}1)$$

with constant extension at the boundaries.

14.3.1.2 Prediction Modes

14.3.1.2.1 Mode 0: DC-Prediction

All positions of P are predicted by sum(EP), i.e. the sum of all elements of EP. If there are no valid edge pixels, the block is predicted by P[m,n]=128, 0<=n,m<N,M.

14.3.1.2.2 Mode 1: Vertical Prediction

```
if(valid_up)
  for(y=0;y<M;y++)
```

```
    for(x=0;x<N;x++)
      P[y,x]=EP[offset+1+x];
```

14.3.1.2.3 Mode 2: Horizontal Prediction

```
if(valid_left)
  for(y=0;y<M;y++)
    for(x=0;x<N;x++)
      P[y,x]=EP[offset-1-y];
```

14.3.1.2.4 Mode 3: Down-Right Prediction

```
if(valid_left&&valid_up)
  for(y=0;y<M;y++)
    for(x=0;x<N;x++)
      P[y,x]=EP[offset+x-y];
```

14.3.1.2.5 Mode 4: Up-Right Prediction (Bidirectional)

```
if(valid_left&&valid_up)
  for(y=0;y<M;y++)
    for(x=0;x<N;x++)
      P[y,x]=(EP[offset+2+x+y]+EP[offset-2-(x+y)])>>1;
```

14.3.1.2.6 Mode 5: Down-Right-Down Prediction

```
if(valid_left&&valid_up)
{
  for(y=0;y<M;y+=2)//even lines
    for(x=0;x<N;x++)
      if( (i=x-(y>>1)) >= 0 )
        P[y,x]=(EP[offset+i]+EP[offset+1+i])>>1;
      else
        P[y,x]=EP[offset+1+2*x-y];
  for(y=1;y<M;y+=2)//odd lines
    for(x=0;x<N;x++)
      if( (i=x-(y>>1)) >= 0 )
        P[y,x]=EP[offset+i];
      else
        P[y,x]=EP[offset+1+2*x-y];
}
```

14.3.1.2.7 Mode 6: Down-Left-Down Prediction

```
if(valid_left&&valid_up)
{
  for(y=0;y<M;y+=2)//even lines
    for(x=0;x<N;x++)
      P[y,x]=(EP[offset+1+x+(y>>1)]+EP[offset+2+x+(y>>1)])>>1;
  for(y=1;y<M;y+=2)//odd lines
    for(x=0;x<N;x++)
      P[y,x]=EP[offset+2+x+(y>>1)];
}
```

14.3.1.2.8 Mode 7: Right-Up-Right Prediction

```
if(valid_left&&valid_up)
{
  for(y=0;y<M;y++)//even columns
    for(x=0;x<N;x+=2)
      P[y,x]=(EP[offset+-1-(y+(x>>1))]+EP[offset+-2-(y+(x>>1))])>>1;
  for(y=0;y<M;y++)//odd columns
    for(x=1;x<N;x+=2)
      P[y,x]=EP[offset+-2-(y+(x>>1))];
}
```

14.3.1.2.9 Mode 8: Right-Down-Right Prediction

```
if(valid_left&&valid_up)
{
  for(y=0;y<M;y++)//even columns
    for(x=0;x<N;x+=2)
      if( (i=-y+(x>>1)) <= 0 )
```

```
        P[y,x]=(EP[offset+i]+EP[offset+i-1])>>1;
      else
        P[y,x]=EP[offset-1-2*y+x];
  for(y=0;y<M;y++)//odd columns
    for(x=1;x<N;x+=2)
      if( (i=-y+(x>>1)) <= 0 )
        P[y,x]=EP[offset+i];
      else
        P[y,x]=EP[offset-1-2*y+x];
}
```

14.3.1.3 Coding of ABT intra prediction modes

For encoding, the ABT intra prediction modes of the blocks of a macroblock are grouped in pairs. Since Intra blocks of variable block size might be indicated in one macroblock, a rule has to be defined for grouping the prediction modes: The prediction modes are grouped left to right, top to bottom.

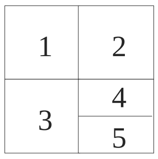


Figure 14-1 – Macroblock in 8x8 Mode with 4 ABT intra subblocks

If the number of intra coded blocks is odd, the prediction mode of the last intra subblock is grouped with a '0' for encoding. For example, in Figure 14-1, a macroblock in 8x8 mode containing blocks with block modes 8x8, 8x8, 8x8, 8x4, is given. The 8x8 prediction modes of blocks 1 and 2 are grouped together for encoding. The prediction mode of block 3 is grouped with prediction mode of block 4. The prediction mode of block 5 is grouped with a prediction mode '0' for encoding. For encoding of the intra prediction modes, the base line encoding method (Section QQ) is used.

14.3.2 ABT Transform Coefficient Decoding

Two transforms are used. One transform is used for signals of length 8 and the other transform is used for signals of length 4. In the following, matrix notation is used to describe the transform process. The 4x4 transform is used if useABT = = 0. The 4x4 transform has different norms in the even and odd basis functions, resulting in a set of 3 different norms for a 2-dimensional transform. The 8x8 transform used for ABT is given as a matrix. In the following, applying this transform to a block is expressed as a matrix multipication. It has one norm for all basis functions. The 8x8 transform T8 is given below.

$$T8 = \begin{bmatrix} 13 & 13 & 13 & 13 & 13 & 13 & 13 & 13 \\ 19 & 15 & 9 & 3 & -3 & -9 & -15 & -19 \\ 17 & 7 & -7 & -17 & -17 & -7 & 7 & 17 \\ 9 & 3 & -19 & -15 & 15 & 19 & -3 & -9 \\ 13 & -13 & -13 & 13 & 13 & -13 & -13 & 13 \\ 15 & -19 & -3 & 9 & -9 & 3 & 19 & -15 \\ 7 & -17 & 17 & -7 & -7 & 17 & -17 & 7 \\ 3 & -9 & 15 & -19 & 19 & -15 & 9 & -3 \end{bmatrix} \qquad (14.2)$$

14.3.2.1 Scanning method

ABT transform coefficients are indicated using the (Run,Level) model. For each transform block size, a distinct coefficient scan is defined. Two general scans methods are used, one for progressive and one for interlaced material.

14.3.2.1.1 Progressive Scan

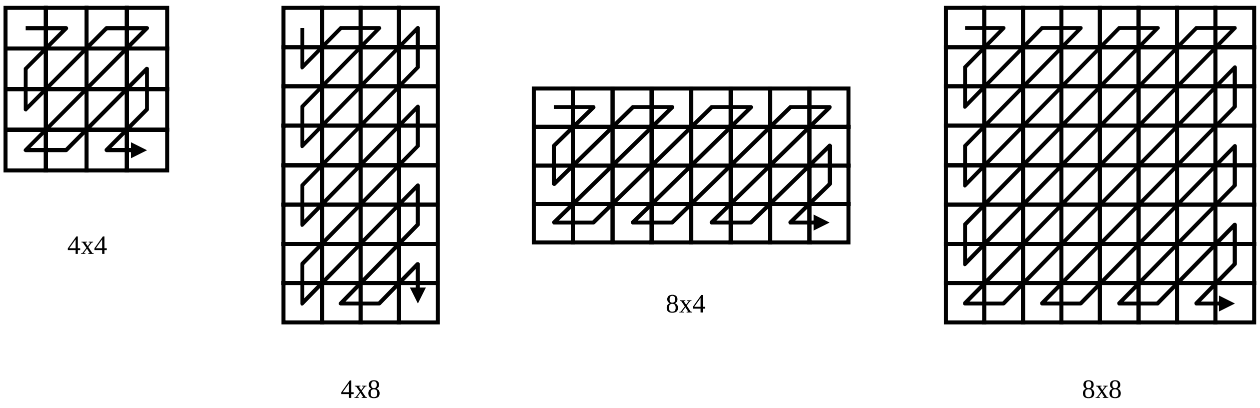


Figure 14-2 – Progressive scan for 4x4, 4x8, 8x4, and 8x8 blocks

14.3.2.1.2 Interlaced Scan

1	3	9	13
2	6	10	14
4	7	11	15
5	8	12	16

Figure 14-3 – 4x4 interlaced scan

1	5	13	21
2	6	14	22
3	7	15	23
4	12	20	28
8	16	24	29
9	17	25	30
10	18	26	31
11	19	27	32

Figure 14-4 – 4x8 interlaced scan

1	3	7	11	15	19	23	27
2	6	10	14	18	22	26	30
4	8	12	16	20	24	28	31
5	9	13	17	21	25	29	32

Figure 14-5 – 8x4 interlaced scan

1	4	9	16	23	31	39	53
2	5	15	22	30	38	46	54
3	8	17	24	32	40	47	59
6	10	21	29	37	45	52	60
7	14	25	33	41	48	55	61
11	18	26	34	42	49	56	62
12	19	27	35	43	50	57	63
13	20	28	36	44	51	58	64

Figure 14-6 – 8x8 interlaced scan

14.3.2.2 Scaling and transform

The scaling of the ABT transform coefficients for each mode uses the scaling values given in the Table 14-1 below. The design is given for quantization parameters QP=0,…,51.

The coefficient R(k,i,j) used in the following is mode dependent and chosen from the table below. Additionally, a bit shift parameter is defined.

Table 14-1 – ABT dequantization mantissa values

k	S_{8x8}	$S_{8x4,4x8}$		S_{4x4}		
0	15	9	11	40	64	51
1	17	10	12	45	72	57
2	19	11	14	50	81	64
3	22	12	16	57	91	72
4	24	14	17	63	102	80
5	27	15	20	71	114	90

Mode 8x8:

```
R[k][i][j] = S8x8[k] for all i,j
Bshift = 7;
```

Mode 8x4:

```
R[k][i][j] = S8x4,4x8[k][0] for all even j
R[k][i][j] = S8x4,4x8[k][1] for all odd j
Bshift = 2;
```

Mode 4x8:

```
R[k][i][j] = S8x4,4x8[k][0] for all even I
R[k][i][j] = S8x4,4x8[k][1] for all odd I
Bshift = 2;
```

Mode 4x4:

```
R[k][i][j] = S4,4[k][0] for (i,j) = {(0,0),(0,2),(2,0),(2,2)},
R[k][i][j] = S4,4[k][1] for (i,j) = {(1,1),(1,3),(3,1),(3,3)},
R[k][i][j] = S4,4[k][2]  otherwise;
Bshift = 0;
```

The reconstructed coefficients are multiplied with the scaling value R

$$YD(i,j) = (YQ(i,j) * R(QP\%6,i,j)) << (QP/6); \quad i=0,..,N; j=0,...,M \qquad (14\text{-}3)$$

The resulting block YD of size MxN is then transformed horizontally. If N=4, the one-dimensional transform is performed according to the Section QQ. If N=8, a matrix multiply is performed and the result is rounded,

$$Z = (YD * TN)>>Bshift, \qquad (14\text{-}4)$$

where (YD * TN) denotes the appropriate horizontal transform operation. The notation (YD * TN)//Bshift means that the resulting value for each element Z(i,j) is down-shifted by Bshift bits with proper rounding.

$$Z(i,j) = sign(Z'(i,j))*[abs(Z'(i,j)) + 2Bshift\text{-}1]>>Bshift. \qquad (14\text{-}5)$$

This operation is performed to stay within 16 bit precision. After the vertical transform step the result is finally rounded

$$X' = (TM^T * Z), \qquad (14\text{-}6)$$

$$X''(i,j) = [X'(i,j) + 25] >> 6. \qquad (14\text{-}7)$$

Here, $(TM^T * Z)$ denotes the appropriate vertical transform. Finally, the reconstructed sample residual values X"(i,j) are added to the prediction values P(i,j) from motion compensation or spatial prediction and clipped to the range of 0 to 255 to form the final decoded sample result:

$$S'(i,j) = clip1(P(i,j) + X''(i,j)). \qquad (14\text{-}8)$$

14.3.3 Deblocking Filter

Deblocking is employed on block and macroblock boundaries. No modifications are introduced to the deblocking filtering process besides a modification of the determination of the threshold parameters α and β, see Section QQ. No filtering is performed at 4x4 block boundaries inside transform blocks.

According to Section QQ, the threshold parameters α and β are driven by the quantization parameter QP. For ABT blocks, the index to the threshold table is increased by IQP. IQP depends on the depth and the width of the neighboring blocks. The depth is the distance to the next block boundary on each side, while the width of the block is the number of pixels at the current boundary.

```
IQP=0;
if (depth(Block p)==8)
  IQP++;
if (depth(Block q)==8)
  IQP++;
if (width(Block p)==8)
  IQP++;
if (width(Block q)==8)
  IQP++;
clip3(0,3, IQP);
```

14.3.4 Entropy Coding

For entropy coding of the ABT coefficients a VLC mode and a CABAC mode are defined.

14.3.4.1 VLC

In VLC mode, all coefficients are indicated as (Level,Run) symbols. Inter blocks use the EOB symbol. For Intra coding, a Coeff_Count symbol is indicated.

Golomb codes of several degrees are employed for encoding of the symbols. The Golomb-0 code is used for encoding of all non-luma symbols as described in Section QQ. For ABT, additional Golomb-1, 2, and 3 codes are employed. The general structure of these codes is depicted in the Table 14-3.

Table 14-3 – Golomb Codes used for encoding ABT symbols

Layer	Golomb-0	Golomb-1	Golomb-2	Golomb-3
0	1	1x	1xx	1xxx
1	01x	01xx	01xxx	01xxxx
2	001xx	001xxx	001xxxx	001xxxxx
3	0001xxx	0001xxxx	0001xxxxx	0001xxxxxx
…	…	…	…	…

Generally, a Golomb codeword consists of a sync word 0…01 and an info word x0x1…xn. For ABT coding, a finite number of symbols is indicated using Golomb codes. Symbols that are not in the finite set are indicated using an Escape symbol and separate coding of Level and Run. The Escape symbol is 59 for all used codes. Since finite alphabets are indicated, the '1'-bit of the sync word is omitted in the layer containing the Escape symbol.

14.3.4.1.1 Intra coding using Coeff_Count

For encoding intra symbols, the EOB symbol is omitted and the Coeff_Count symbol is used. Since EOB is the first symbol in all (inter-) tables, the table entries for encoding intra (Level,Run) symbols are shifted down by one. Intra Coeff_Count is indicated using the Golomb-2 code:

Table 14-4 – Golomb Codes used for encoding ABT Coeff_Count symbols

Coeff_Count	Golomb-2 symbol
0	100
1	101
2	110
3	111
4	01000
…	…

14.3.4.1.2 2D (Level,Run) Symbols

The Golomb code used for encoding the (Level,Run) symbols is block mode dependent. The degree of the Golomb code and the number of layers being used is given in Table I-5. For all modes, the first 60 code words of these codes are used, with code word 59 beeing the Escape symbol. In the highest layer, the terminating sync bit is omitted.

Table 14-5 – Connection of ABT modes and Golomb code degrees and the number of layers

	Inter		Intra	
Block Mode	Degree	Layers	Degree	Layers
8x8	0	6	2	4
8x4,4x8	1	5	2	4
8x8	2	4	2	4

14.3.4.1.3 Code Tables

Four codeword tables are used to adapt the VLC code to the symbol statistics. These tables are given in Table 14-7. The intra tables are chosen QP-dependent. The tables contain the (odd) codeword numbers for Level>0. Level<0 is indicated using the even numbers. EOB is indicated as code word 0. In case of Intra coding, no EOB symbol is indicated and the symbols are shifted down by one in the table.

Symbols that are not in the table are indicated using the Escape Symbol 59 first and then Level using Golomb-3 and Run using Golomb-2. Here, no finite alphabet is assumed. The Levels are arranged as given in Table 14-6.

Table 14-6 – Codeword numbers for Level symbols after escape code

Level	Code word
1	0
-1	1
2	2
-2	3
3	4
-3	5
4	6
…	…

Table 14-7 – Inter and Intra (Level,Run) code word table. Cells marked '*' are not valid.

		Level (Inter) →								abs(Level)-1 (Intra, QP<26) →								abs(Level)-1 (Intra, 26<=QP<34) →								abs(Level)-1 (Intra, QP≥34) →							
		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
	0	0	1	5	13	21	31	39	47	0	1	3	7	9	13	19	21	0	1	3	9	13	19	23	31	0	1	5	13	21	33	43	57
	1	*	3	15	33	51	*	*	*	*	5	15	25	31	39	45	49	*	5	15	27	37	49	57	*	*	3	17	31	49	*	*	*
	2	*	7	25	53	*	*	*	*	*	11	29	41	51	*	*	*	*	7	25	43	*	*	*	*	*	7	25	47	*	*	*	*
	3	*	9	35	*	*	*	*	*	*	17	35	55	*	*	*	*	*	11	35	*	*	*	*	*	*	9	35	*	*	*	*	*
	4	*	11	45	*	*	*	*	*	*	23	43	*	*	*	*	*	*	17	45	*	*	*	*	*	*	11	41	*	*	*	*	*
	5	*	17	55	*	*	*	*	*	*	27	57	*	*	*	*	*	*	21	51	*	*	*	*	*	*	15	51	*	*	*	*	*
	6	*	19	*	*	*	*	*	*	*	33	*	*	*	*	*	*	*	29	*	*	*	*	*	*	*	19	*	*	*	*	*	*
Run ↓	7	*	23	*	*	*	*	*	*	*	37	*	*	*	*	*	*	*	33	*	*	*	*	*	*	*	23	*	*	*	*	*	*
	8	*	27	*	*	*	*	*	*	*	47	*	*	*	*	*	*	*	39	*	*	*	*	*	*	*	27	*	*	*	*	*	*
	9	*	29	*	*	*	*	*	*	*	53	*	*	*	*	*	*	*	41	*	*	*	*	*	*	*	29	*	*	*	*	*	*
	10	*	37	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	47	*	*	*	*	*	*	*	37	*	*	*	*	*	*
	11	*	41	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	53	*	*	*	*	*	*	*	39	*	*	*	*	*	*
	12	*	43	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	55	*	*	*	*	*	*	*	45	*	*	*	*	*	*
	13	*	49	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	53	*	*	*	*	*	*
	14	*	57	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	55	*	*	*	*	*	*
	15	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

14.3.4.2 CABAC

Level and Run are separated for CABAC. Coeff_Count is indicated instead of a EOB symbol. The ABT Levels are indicated as defined in Section QQ. For encoding of Coeff_Count and the Run values of 4x8, 8x4, and 8x8 blocks, an additional binarization is introduced. For both, a separate set of contexts is defined. 4x4 Coeff_Count and Run are indicated as described in Section QQ.

14.3.4.2.1 Binarization for 4x8, 8x4, and 8x8 block Coeff_Count and Run Values

Coeff_Count and Run are denoted 'Symbol' in the following. Unary binarization is applied for Symbol<16. For Symbol >=16, the trailing sync bit is omitted and a fixed length binary representations of (Coeff_Count-16) is attached. The bits of the representation are indicated from low to high, i.e. 20, 21,… . The binarization is shown in Table 14-2.

Table 14-8 – Binarization for ABT Coeff_Count and Run values

n	Binarization				
0	1				
1	0	1			
2	0	0	1		
3	0	0	0 1		
4	0	0	0 0	1	
…				…	
16	0	0	0 0	0 …	000
17	0	0	0 0	0 …	100
18	0	0	0 0	0 …	010
…					…
Ctx	0	1	2	3	4

14.3.4.2.2 Truncated Binarization

Both, Coeff_Count and Run values are always bound by a maximum value, see Section QQ. The maximum value max_value is use to truncate the binarization displayed in Table 14-8. If Symbol < 16, the terminating '1' of the binarization is omitted. For Symbols >= 16, the maximum value determines the number of bits nf used for the fixed length binarization,

$$nf = \text{ceil} (\log2(\text{max\_value})); \qquad \text{(E-9)}$$

where ceil(×) denotes up-rounding to the next integer.

14.3.4.2.3 Context Models for ABT Coeff_Count

The enumeration of the contexts is according to Table I-2. Four contexts models are used for the first context. The model is determined by

$$\text{ctx\_coeff\_count}(C) = ((CC(A)==0) ? 0 : 1) + 2*((CC(B)==0)?0:1); \qquad \text{(E-10)}$$

where CC(X) , X=A,B, is Coeff_Count of the left and top neighbouring decoded blocks A and B. One model is assigned to each of the remaining contexts.

14.3.4.2.4 Context Models for ABT Run Values

The enumeration of the contexts is according to Table 14-8. Run is indicated exploiting the information of Coeff_Count and the previously coded Runs. The binarization of Coeff_Count is used for Runs as well. Here, only one context is used for the first bin. The remaining contexts are defined as with Coeff_Count. Two sets of Run contexts can be chosen, depending on the activity of the block, which is measured by Coeff_Count

$$\text{ctx\_run} = ((\text{COEFF\_COUNT}) >= 4) ? 1 : 0. \qquad \text{(E-11)}$$

Annex A

Profile and Level Definitions

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

A.1 General

Profiles and Levels are used to summarize the capability of decoders, and to indicate interoperability points between individual decoder implementations. This Specification does not include individually selectable “options” at the decoder, as this would increase interoperability difficulties.

Each Profile defines a set of algorithmic features which shall be supported by all decoders compliant with that Profile. Note that encoders are not required to make use of any particular set of features supported in a Profile.

Each Level defines a set of limits on the values which may be taken by the parameters of this Specification. 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.

All video decoders compliant with this Specification shall support the Baseline Profile. The support of other Profiles is optional.

A.2 Requirements on Video Decoder Capability

All video decoders compliant with this Specification shall express their capability to decode video in the format of a list of one or more Profiles from 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.

All video decoders compliant with this Specification shall support the Baseline Profile. The Level supported for the Baseline Profile shall not be less than the Level supported for any higher Profile. The support of higher Profiles (those which contain features in addition to those of the Baseline Profile) is optional.

A.3 Baseline Profile

All video decoders compliant with this Specification shall support the Baseline Profile.

The Baseline Profile consists of the following features:

a) I and P picture types
b) In-loop deblocking filter
c) Progressive pictures
d) Interlaced pictures (only for decoders supporting Level 2.1 and above)
e) 1/4-sample motion compensation
f) Tree-structured motion segmentation down to 4x4 block size
g) VLC-based entropy coding
h) Flexible macroblock ordering (maximum 8 slice groups)
i) Motion Vectors in the range of
j) Chrominance format 4:2:0

A.4 Main Profile

Video decoders may optionally support the Main Profile.

The Main Profile consists of the following features:

a) All features included in the Baseline Profile
b) B pictures
c) CABAC
d) Adaptive block-size transforms
e) Motion Vectors in the range of

NOT in either Baseline or Main Profile:

- 1/8-sample motion compensation
- Mixing intra and inter coding types within a macroblock
- Data partitioning
- SP & SI “switching” pictures
- All TBD features

Remaining open issues for Main Profile:

- Whether to allow smaller than 8x8 bi-predictive motion in B pictures
- Whether to include adaptive B picture interpolation coefficients

--Dave Lindbergh]

A.5 Level Definitions

A.5.1 General

Level limits are expressed in units of whole luminance macroblocks. If a particular picture sample 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 compliance with this section.

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) Both picture Height and picture Width (in whole macroblocks) are <= sqrt(LevelLimitMaxPictureSize * 8), and,

d) Maximum bit rate of the video bitstream is <= the Level limit given.

Regardless of the ratio of sample processing rate to picture size, decoders are not required to decode frame rates greater than 172 Hz. Note that in most cases the sample processing rate limit will impose a much lower maximum frame rate.

The definitions of each Level include the requirements of all lower numbered Levels. 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 Specification; 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.5.2 Level Limits

Table A.1 below gives the parameter limits for each Level.

Table A.1 – Level Limits

Level Number	Maximum Picture Size (macroblocks)	Maximum Processing Rate (macroblocks/second)	Reference Frames Supported	Maximum Video Bitrate	Maximum HRD/VBV Buffer Size
1	99	1,485	3	TBD	TBD
1.1	396	2,970	5	TBD	TBD
1.2	396	5,940	5	TBD	TBD
2	396	11,880	5	TBD	TBD
2.1	792	19,800	5	TBD	TBD
2.2	1,620	20,250	4	TBD	TBD
3	1,620	40,500	4	TBD	TBD
3.1	3,600	108,000	4	TBD	TBD
3.2	5,120	216,000	3	TBD	TBD
4	9,660	245,760	3	TBD	TBD
5	19,200	491,520	3	TBD	TBD

"Reference Frames Supported" means the number of frames of the maximum picture size supported by the Level. Note that P pictures require one reference frame, and B pictures require two reference frames; any remaining reference frames may be used as multiple reference frames.

Levels with non-integer Level numbers are called "intermediate Levels". All Levels have the same status, but note that some applications may choose to use only the integer-numbered Levels.

Non-normative subclause A.6 shows the effect of these limits on frame rates for several example picture formats.

A.5.3 Number of Reference Frames Supported

For frames of the maximum picture size for each Level, decoders shall support the number of reference frames given in the Level limits above.

For smaller frames, the number of reference frames supported shall be (units are whole macroblocks):

Number of reference frames = min(floor(Level Limit Value * (Maximum Picture Size / Actual Picture Size)), 15)

A.6 Non-normative – Effect of Level Limits on Frame Rate

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**	**9,660**	**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	2,472,960	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

	Sample		**MB**	**MB**											
Format	**Width**	**Height**	**Wide**	**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 CCIR 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.1 Introduction

This annex defines a byte stream format specified for use by systems that transmit the video content primarily as an ordered stream of bytes or bits, in which the locations of synchronization boundaries need to be identifiable from patterns in the data, such as ITU-T Rec. H.222.0 | ISO/IEC 13818-1 systems or ITU-T Rec. H.320 systems. For bit-oriented transmission, the network bit order for the byte stream format is defined to start with the MSB of the first byte and proceeds to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte_stream_unit() structures. Each byte_stream_unit() contains one start code prefix (SCP) and one nal_unit(). Optionally, at the discretion of the encoder if not prohibited by a system-level specification, the byte_stream_unit() may also contain additional "stuffing" zero-valued bytes.

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. Use of the long SCP is required for nal_units().

B.2 Byte stream NAL unit syntax

byte_stream_unit() {	**Category**	**Mnemonic**
while (next_bits() != 0x0001 && next_bits() != 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.3 Byte stream NAL unit semantics

zero_byte is a single byte (8 bits) having the value zero (0x00). Optionally, at the discretion of the encoder if not prohibited by a system-level specification, 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 value of the PNUT of the nal_unit_type as defined in Table 8-1, in order to ensure the use of the long SCP for certain nal_unit_type values. At least two zero_byte syntax elements shall be present in each byte_stream_unit() when (PNUT – picture_header_flag) is less than 3. This ensures use of the long SCP in the byte_stream_unit() for these nal_unit() strucutres. The use of the long SCP for nal_unit() structures with other values of nal_unit_type is optional.

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.

B.4 Example Encoder procedure (Non-normative)

The encoder can produce an EBSP 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'

This process can allow any RBSP data to be sent in an EBSP while ensuring that no long SCP and no byte-aligned short SCP is emulated in the EBSP.

Note that the effect of concatenating two or more RBSP's and then encapsulating them into an EBSP is the same as first encapsulating each individual RBSP and then concatenating the result (because the last byte of an RBSP is never zero). This allows the association of individual EBSP's to nal_units() to be altered without affecting the content of the EBSP.

B.5 Decoder byte-alignment recovery (Non-Normative)

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 00000001' (15 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: The use of the byte sequence 0x00 0x02 is reserved for further study.

Note: Many systems are inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this sub-clause.

Note: The byte alignment detection procedure described in this sub-clause 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 following this byte sequence is the last bit of a byte-aligned SCP.

Annex C
Supplemental Enhancement Information

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

C.1 Introduction

This annex defines supplemental enhancement information that provides a data delivery mechanism that is synchronous with the video data content. Each sei_message() defines PayloadType and PayloadSize parameters

C.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		
reserved		variable
if(!byte_aligned()) {		
bit_equal_to_one		f(1)
while(!byte_aligned())		
bit_equal_to_zero		f(1)
}		
}		

C.2.1 Temporal reference syntax

temporal_reference(PayloadType, PayloadSize) {	**Category**	**Descriptor**
progressive_scan		u(1)
bottom_field_indicator /* zero if progressive_scan is 1 */		u(1)
six_reserved_one_bits		f(6)
temporal_reference_value		u(v)
}		

C.2.2 Clock timestamp syntax

clock_timestamp(PayloadType, PayloadSize) {	**Category**	**Descriptor**
progressive_scan		u(1)
bottom_field_indicator /* zero if progressive_scan is 1 */		u(1)
six_reserved_one_bits		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 = 19		
} else {		
seconds_flag	7	u(1)
bit_count = 9		
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)
}		

C.2.3 Pan-scan rectangle syntax

pan_scan_rect (PayloadType, PayloadSize) {	**Category**	**Descriptor**
pan_scan_rect_identifier	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)
}		

C.3 SEI payload semantics

reserved: This syntax element is reserved for future use by ITU-T | ISO/IEC. It shall not be present in a bitstream conforming to this Specification. If this syntax element is encountered by a decoder, it may be skipped (removed and discarded).

C.3.1 Temporal reference semantics

progressive_scan: This parameter indicates whether the current picture is in progressive or interlaced scan format.

bottom_field_indicator: When progressive_scan is 0, this parameter indicates whether the temporal reference is for the top (0) or bottom (1) field.

six_reserved_one_bits: Reserved for future use by ITU-T | ISO/IEC. Shall be equal to one (1). A decoder conforming to this Specification shall ignore the value of these bits.

temporal_reference_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_reference_value shall remain constant for the video stream. For a temporal_reference_value encoded using *n* bytes, the temporal_reference contains the remainder of a clock tick counter modulo 256^n.

C.3.2 Clock timestamp semantics

progressive_scan: This parameter indicates whether the current picture is in progressive or interlaced scan format.

bottom_field_indicator: When progressive_scan is 0, this parameter indicates whether the temporal reference is for the top (0) or bottom (1) field.

six_reserved_one_bits: Reserved for future use by ITU-T | ISO/IEC. Shall be equal to one (1). A decoder conforming to this Specification shall ignore the value of these bits.

C.3.3 Pan-scan rectangle semantics

These parameters define the coordinates of a rectangle relative to the sequence-level cropping rectangle. Each coordinate of this rectangle is defined in units of $1/16^{th}$ sample spacing relative to the luma sampling grid.