SMPTE 421M-2006

SMPTE STANDARD

VC-1 Compressed Video Bitstream Format and Decoding Process

3 Barker Ave.
White Plains, NY 10601
+1 914 761 1100
Fax +1 914 761-3115
E-mail eng@smpte.org
Web http://www.smpte.org

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in Part XIII of its Administrative practices.

This Standard 421M was prepared by Technology Committee C24.

Introduction

This document was prepared for the primary purpose of documenting the bitstream format and decoding process used in the VC-1 video decoder. It defines the bitstream syntax, semantics and constraints for compressed video bitstreams and describes the complete process required to decode them.

VC-1, as defined in this document, consists of three profiles: Simple, Main, and Advanced. Simple and Main profile were originally developed for use in lower-bit-rate networked computing environments. As such, certain assumptions were made regarding the display environment (e.g. square pixel aspect ratio) to improve compression efficiency. The Advanced Profile adds extensive in-band metadata support to allow for optimized experience on a wide range of display devices.

SMPTE Standard: VC-1 Compressed Video Bitstream Format and Decoding Process

Table of Contents

Table of Figures

Table of Tables

1 Scope

This document defines the bitstream syntax and semantics for compressed video data in VC-1 format, and specifies constraints that are required for conformant bitstreams. It also describes the complete process required to decode the bitstream. The compression algorithm is not specified in this standard. The video formats supported by VC-1 include progressive and interlaced video sampled in the form of Y luma samples and C_b, C_r color-difference in 8-bit per component sample values resulting from a 4:2:0 sampling grid. The decoding process outputs 8-bit per component video samples corresponding to the original 4:2:0 sampling grid. The display rendering process by which decoded Y, C_b, C_r samples are converted to a visible image or to a video output signal in a complete decoding system or device are not specified in VC-1. A VC-1 bitstream may convey additional metadata and user data which shall be accounted for in the buffer model. Metadata included in VC-1 streams is not used by the decoding process, but is passed to the display rendering process for the identification and reconstruction of the sampled video format, sample aspect ratio, color space, etc.

2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the standards indicated below.

Recommendation ITU-R BT.1700[1], "Characteristics of video signals for conventional analogue television systems", 2005.

Recommendation ITU-R BT.601-5, "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios", 1995.

Recommendation ITU-R BT.709-5, "Parameter values for the HDTV standards for production and international programme exchange", 2002.

ISO/IEC 13818-1:2000, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Systems" (2nd Edition).

SMPTE 274M-2005, Television – 1920 x 1080 Image Sample Structure, Digital Representation and Digital Timing Reference Sequences for Multiple Picture Rates.

SMPTE 293M-2003, "Television – 720 x 483 Active Line at 59.94-Hz Progressive Scan Production – Digital Representation".

SMPTE 296M-2001, Television – 1280 x 720 Progressive Image Sample Structure – Analog and Digital Representation and Analog Interface.

Recommendation ITU-R BT.1361, "Worldwide unified colorimetry and related characteristics of future television and imaging systems".

3 Overview

This section gives an overview of the syntax, transport requirements, and the organization of this document.

[1] Note: Recommendation ITU-R BT.1700 Part A references SMPTE 170M. Recommendation ITU-R BT.1700 also replaces Recommendation ITU-R BT. 470-6.

3.1 Syntax Overview (Informative)

The syntax of this standard consists of the hierarchical layers: sequence, entry-point, picture, slices, macroblocks (MB), and blocks. In the simple and main profile, a sequence consists of a series of one or more coded pictures. In the advanced profile, a sequence consists of a series of one or more entry-point segments, where each entry-point segment consists of a series of one or more pictures, and where the first picture in each entry-point segment provides random access. A picture is decomposed into macroblocks, each of which consists of four luma blocks, and two color-difference blocks. A slice comprises one or more contiguous rows of macroblocks.

An overview of the bitstream syntax for the sequence layer, entry-point layer and picture layer is shown in Figure 1. The entry-point layer is present only in the advanced profile. An overview of the picture layer, slice layer and macroblock layer is shown in Figure 2. The syntax of block layer is shown in Figure 27 and Figure 28. The bitstream syntax and semantics of the sequence layer and the entry-point layer are described in Section 6.

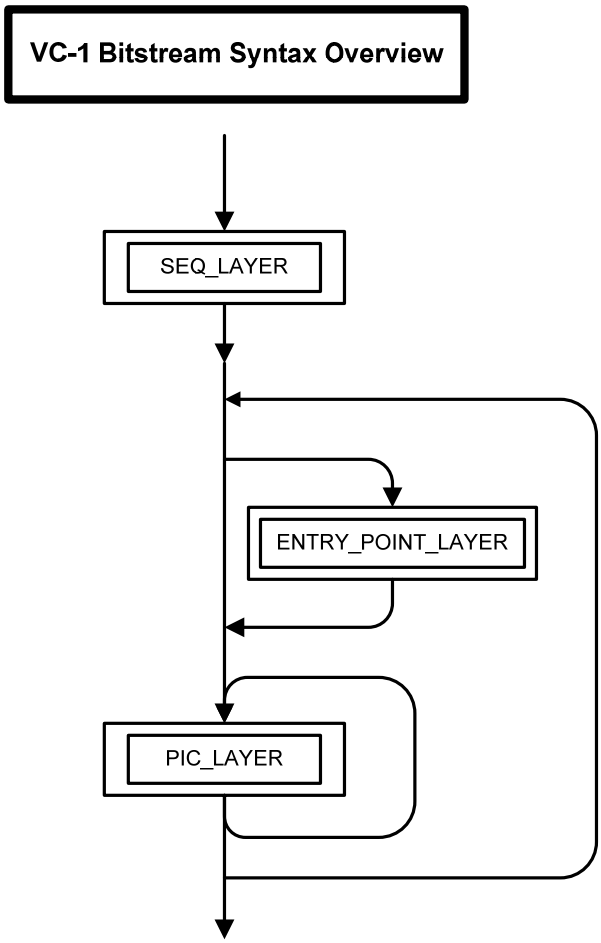


Figure 1: Bitstream Syntax Overview (Entry_Point_Layer is present only in advanced profile)

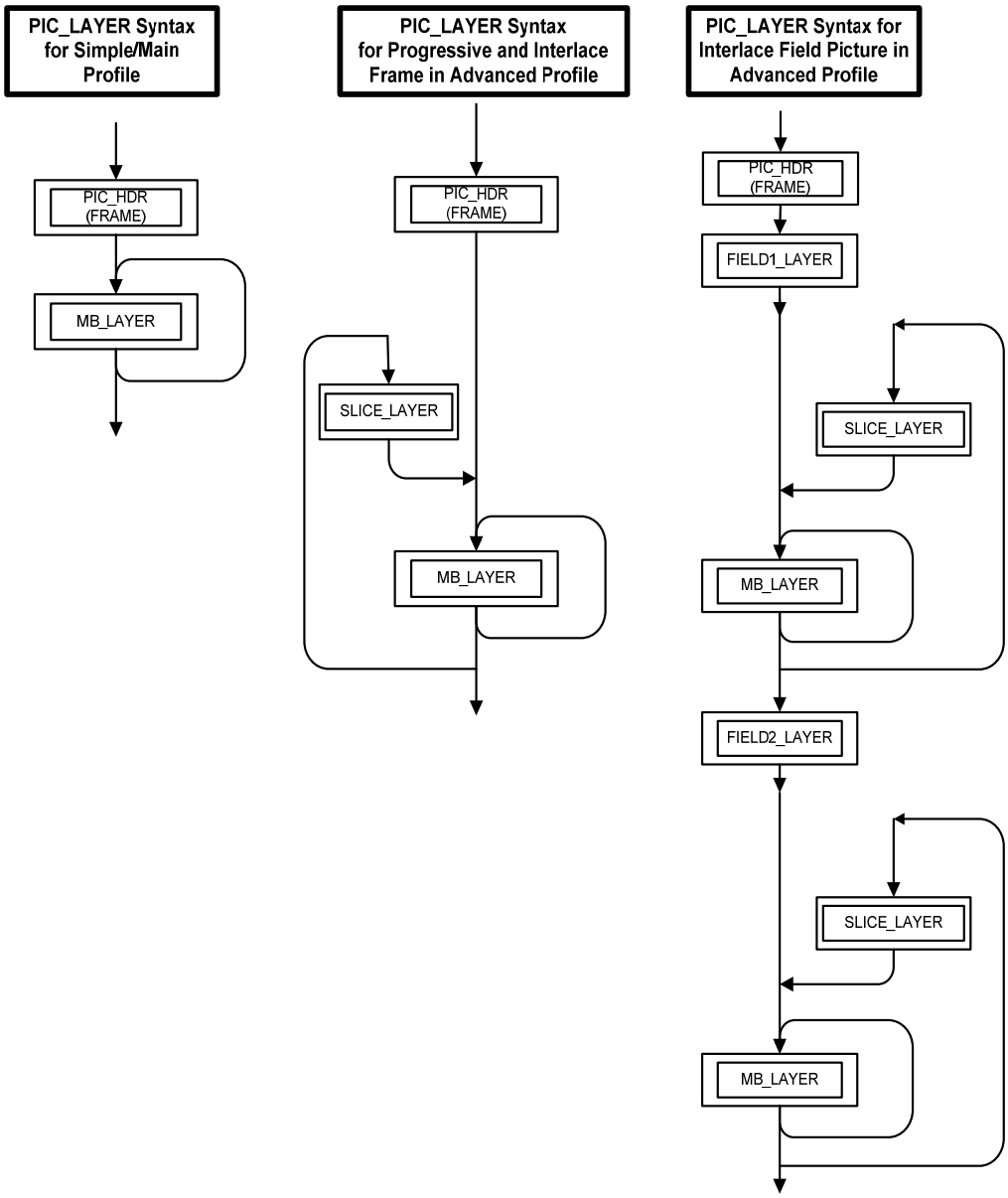


Figure 2: Picture Layer Syntax Overview

Both progressive and interlace syntax are supported. Pictures which are coded using the interlace syntax can be coded as a single frame, or as two fields. Pictures which are coded using the progressive syntax are coded as a single frame. Pictures coded using the progressive and pictures coded using the interlace syntax can be mixed in the same sequence. Each picture can be coded as an I-picture, P-picture, skipped picture, BI-picture, or as a B-picture as defined in section 4.12 and summarized below.

- An I-picture (intra-coded picture) is a picture that is coded using information only from itself, and does not depend on information from any other picture. All the macroblocks in an I-picture are intra-coded.
- A P-picture is a picture that is coded using motion compensated prediction from past reference pictures. A P picture can contain macroblocks that are inter-coded (i.e. coded using prediction) and macroblocks that are intra-coded.
- A B-picture is a picture that is coded using motion compensated prediction from past and/or future reference pictures. A B picture can contain macroblocks that are inter-coded, and macroblocks that are intra-coded.
- A BI-picture is a B picture that contains only intra-coded macroblocks.
- A skipped picture is a P-picture that is identical to its reference picture.

There are three profiles: simple, main and advanced. Simple and main profiles support only progressive pictures. Each profile contains multiple levels. There are two levels in simple profile, three levels in main profile, and five levels in advanced profile. For details on profiles and levels, and their relation to coding tools, see Annex D.

The bitstream syntax of the picture layer as well as the syntax of slice, macroblock and block layers are described in Section 7 (for progressive pictures) and in Section 9 (for interlace frame and interlace field pictures).

3.2 Decoding Process Overview

An overview of the decoding process, as defined in this document, is shown in Figure 3 (for simple and main profiles), and in Figure 4 (for advanced profile).

Note: Simple profile does not use all the processes illustrated in Figure 3.

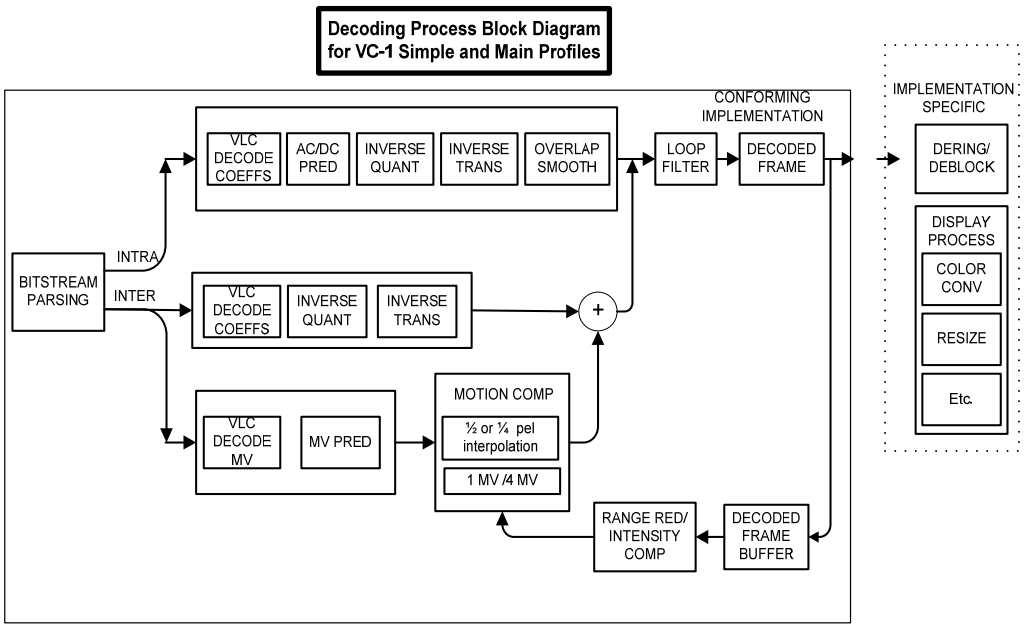


Figure 3: Decoding Process Block Diagram for Simple and Main Profile

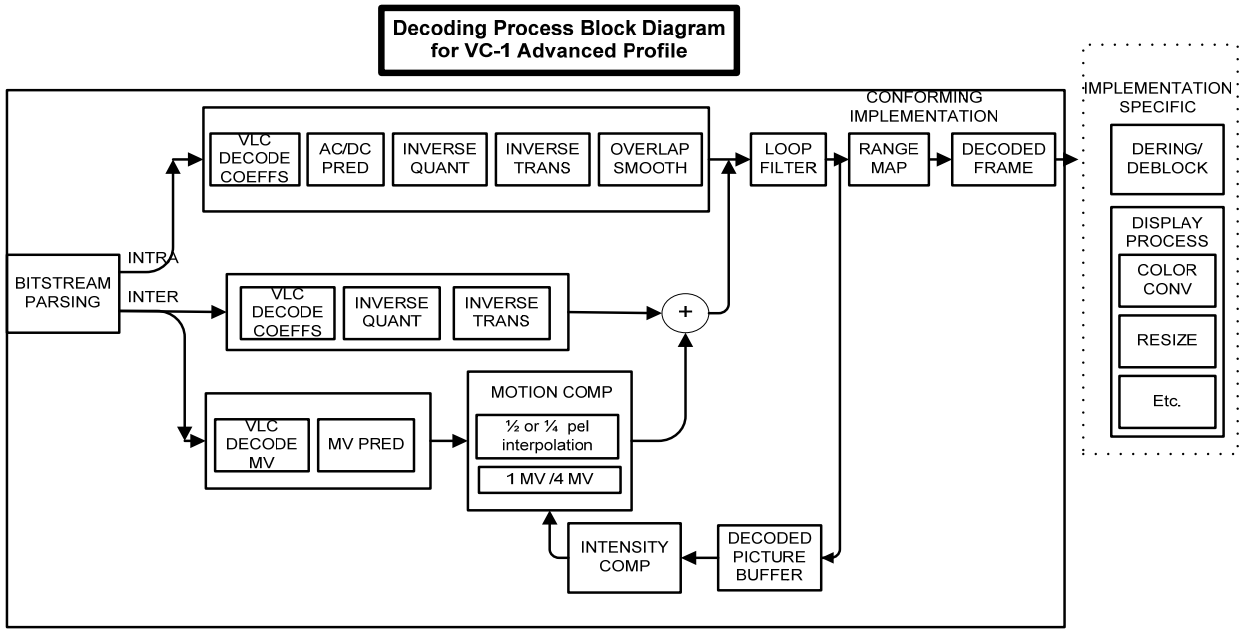


Figure 4: Decoding Process Block Diagram for Advanced Profile

The various decoding processes are described in Section 8 (for progressive picture), and in Section 10 (for interlace frame and interlace field picture). Post-processing and display processing are assisted by information carried in the compressed bitstream (e.g. sample aspect ratio). In addition to the output 4:2:0 samples, the decoding process produces output metadata (which could be used in post-processing and during display processing). This is illustrated in Figure 5. Note that range-reduction block present in simple/main profile is part of the prediction loop, while range map block present in the advanced profile is outside the prediction loop.

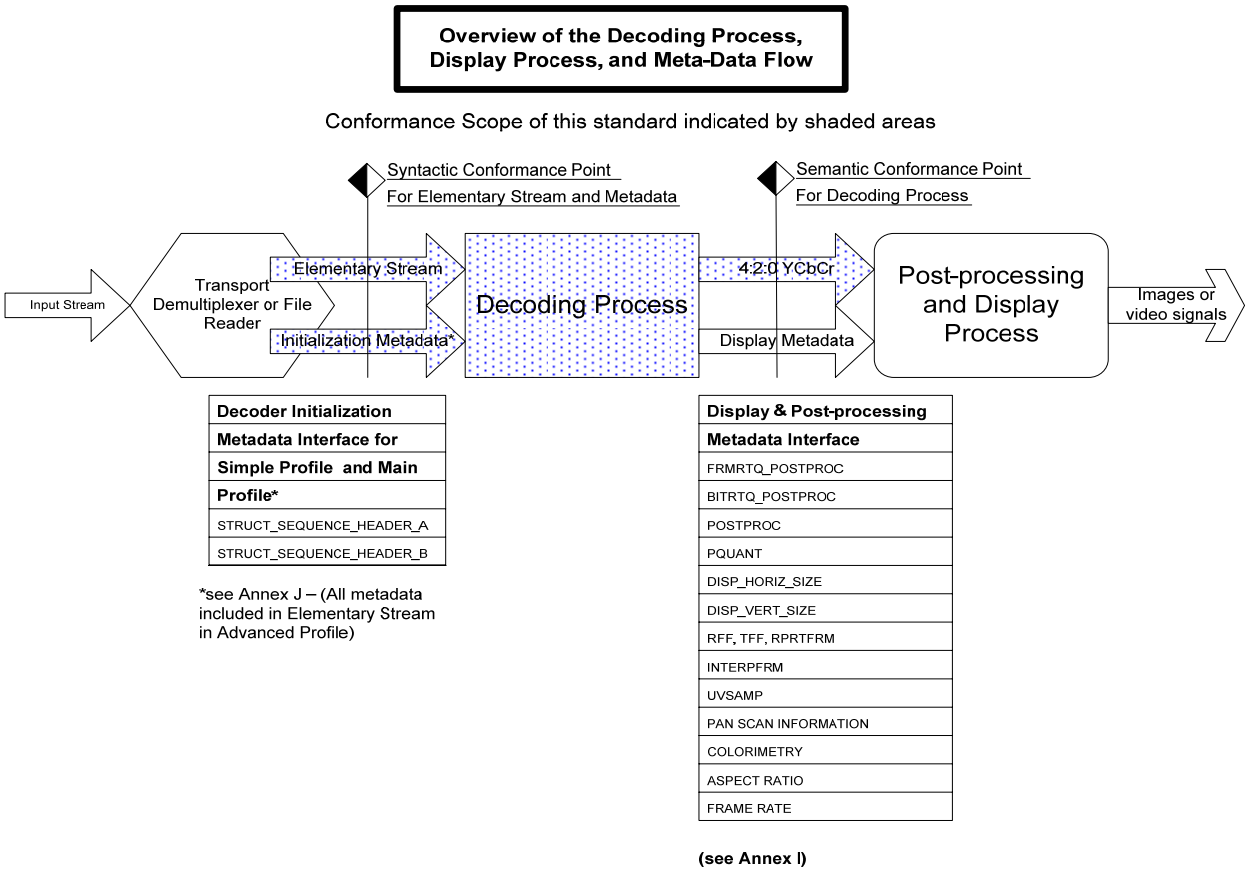


Figure 5: Overview of the Decoding Process and Meta-Data

3.3 Encoding Process Overview (Informative)

An informative overview of the encoding process is shown in Figure 6 which depicts some of the major blocks in the encoder. Annex K presents additional details on some of the encoding blocks.

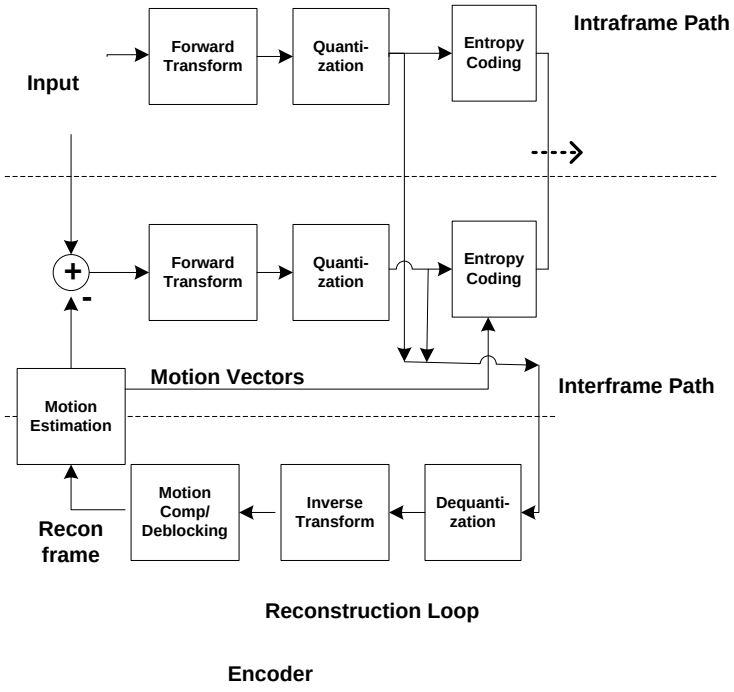


Figure 6: Overview Block Diagram of the Encoding Process (Informative)

3.4 Document structure (Informative)

Section 4 presents notation and definition of terms used in this document. Section 5 describes the input source format, and the hierarchical elements of the syntax. Section 6 describes the syntax and semantics of the sequence and entry-point layer. Section 7 describes the syntax and semantics of the picture, slice, macroblock, and block layers of a progressive picture. Section 8 describes the decoding process of a progressive picture. Section 9 describes the syntax and semantics of the picture, slice, macroblock and block layers of an interlace frame and interlace field picture. Section 10 describes the decoding process of an interlace frame and interlace field picture. Table 1 shows the sections where the individual functional elements are described in detail.

Table 1: Functional Elements in VC-1 and their section numbers

Number	Function Name	Simple/Main Profile (Progressive only)	Advanced Profile, Progressive	Advanced Profile, Interlace
1	Sequence level Bitstream Parsing	N/A	6.1	6.1
2	Entry Point Bitstream Parsing	N/A	6.2	6.2
3	Picture Level Bitstream Parsing	7.1	7.1	9.1
4	VLC Decode Coeffs (VLC tables)	0,8.3.6.2,11	0,8.3.6.2,11	10.1.2.5,11

5	VLC Decode MV (VLC tables)	8.3.5.2.1,11	8.3.5.2.1,11	10.3.5.4,10.7.3.6,11
6	MV PRED	8.3.5.3	8.3.5.3	10.3.5.4.3,10.7.3.5
7	Motion Comp (1-MV/4-MV)	8.3.5.4,8.3.6.5(P) 8.4.5.11(B)	8.3.5.4,8.3.6.5(P) 8.4.5.11(B)	10.3.5.4.4 (Field P); 10.4.6.3.2 (Field B); 10.7.4.1 (Frame P) 10.8.6.9(Frame B)
8	DC Prediction	8.1.3.2	8.1.3.2	10.1.2.2 (Field) 10.5.2.1 (Frame)
9	AC Prediction	8.1.3.7	8.1.3.7	10.1.2.6(Field), 10.5.2.2(Frame)
10	Inverse Quantization	8.3.6.3 (P pic) 8.1.3.3(I pic DC) 8.1.3.8 (I pic AC)	8.3.6.3 (P pic) 8.1.3.3 (I pic DC) 8.1.3.8 (I pic AC)	
11	Inverse Transform	Annex A	Annex A	Annex A
12	Loop Filter	8.6	8.6	10.10
13	Intensity Comp	8.3.8	8.3.8	10.3.8
14	Range map	N/A	6.2.15,6.2.16	6.2.15,6.2.16
15	Sync Markers	8.8	N/A	N/A
16	Bitplane Coding	8.7	8.7	8.7
17	Pan Scan	N/A	8.9	8.9

4 Notation

The following notation is used in this document.

4.1 Conformance Notation

Documents consist of normative text and optionally, informative text. Normative text is that text which describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should" or "may".

Informative text is text that is potentially helpful to the user, but not indispensable and can be removed, changed or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in the document is normative except: the Introduction, any section explicitly labeled as "Informative", or individual paragraphs that start with "Note:".

Normative references are those external documents referenced in normative text and are indispensable to the user. Bibliographic references are those references made from informative text or are otherwise not indispensable to the user.

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate a course of action permissible within the limits of the document.

The keyword, "reserved" indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword, "forbidden" indicates "reserved" and in addition indicates that the provision shall never be defined in the future.

A conformant implementation is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

4.2 Arithmetic Operators

+	Addition.
–	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
– –	Decrement.
+=	a += b is defined as a = a + b
–=	a –= b is defined as a = a – b
*	Multiplication.
/	Integer division with truncation towards zero. For example, 7/4 and –7/–4 are truncated to 1 and –7/4 and 7/–4 are truncated to –1.
÷	Real division.
//	Integer division with rounding to the nearest integer. Half-integer values are rounded away from zero unless otherwise specified. For example 3//2 is rounded to 2, and (-3)//2 is rounded to -2. // is also used as a comment marker in pseudo-code and syntax tables.
##	Rest of the line is a comment.
| |	Absolute value.
abs ( )	Absolute value | x | = x , when x > 0 | x | = 0, when x == 0 | x | = –x, when x < 0
%	x%a is defined as the modulus operator for x>= 0, a>0. x%a = -((-x%a)) for x< 0, a>0. x%a is defined only for positive values of a.
sign( )	Sign. sign(x) = 1, when x >= 0 sign(x) = –1, when x < 0
int( )	Truncation to integer operator. Returns the integer part of the real-valued argument.

nint ( )	Nearest integer operator. Returns the nearest integer value to the real-valued argument. Half-integer values are rounded away from zero.
clip ( )	clip(n) = 255 if n > 255, clip(n) = 0 if n < 0, clip(n) = n otherwise
ceil ( )	Ceiling operator. Returns the smallest integer which is greater than or equal to the real-valued argument. For example, ceil (1.5) returns 2 and ceil (3.0) returns 3.
max	Maximum of the arguments.
min	Minimum of the arguments.
√	Square root.
$\log_2$	Logarithm to base 2.
median3 ()	Median of 3 values (see section 4.11 for definition)
median4 ()	Median of 4 values (see section 4.11 for definition)
smod	Signed modulus operator (see section 4.11 for definition)

4.3 Logical operators

||	Logical OR.
&&	Logical AND.
!	Logical NOT

TRUE/FALSE Convention: The syntax uses the convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is TRUE and a variable or expression evaluating to a zero value is equivalent to a condition that is FALSE.

4.4 Relational operators

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

4.5 Bitwise operators

A twos complement number representation is assumed where the bitwise operators are used.

&	AND
|	OR
^	XOR.
>>	Shift right with sign extension.
<<	Shift left with zero fill.

^=	a ^= b is defined as a = a ^ b

4.6 Assignment

=	Assignment operator.

4.7 Precedence Order of Operators

The precedence order of operators is defined as follows:

Operators	Type of operation	Associativity
()	Expression	Left to Right
++, ––	Postfix operators	Right to Left
–,!	Unary	
*, /, ÷,%, //	Multiplicative	Left to Right
+, –	Additive and Subtractive	Left to Right
<<, >>	Shift	Left to Right.
< , >, <=, >=	Relational	Left to Right
==, !=	Equality	Left to Right
&, |, ^	Bitwise operator	Left to Right
&&, ||,	Logical operators	Left to Right
=, +=, –=,^=	Assignment operators	Right to Left

Operators are listed in descending order of precedence. If several operators appear in the same line, they have equal precedence. When several operators of equal precedence appear at the same level in an expression, evaluation proceeds according to the associativity of the operator either from right to left or from left to right.

4.8 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream.

uimsbf	Unsigned integer, most-significant bit first.
bslbf	Bit String, left bit first.
vlclbf	Variable length prefix code, left bit first, where "left" refers to the order in which the VLC codes are written.
VLC	Variable-length code
FLC	Fixed-length code

4.9 Pseudo-code operations

The following operations are used in the pseudo-code to define the decoding process.

- // is a comment to the line end (note – not integer division)
- /* this is a comment start and end */
- A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement.
- while (condition)

(statement)

/* specifies repeated execution of statement until condition is no longer TRUE. */

- for(initial statement; condition; subsequent statement)

(primary statement)

/* specifies evaluation of initial statement followed by evaluation of condition, and if condition is TRUE, specifies repeated execution of primary statement followed by subsequent statement until condition is no longer TRUE */

- if(condition)

(statement)

else

(alternate statement)

/* statement is executed if condition is TRUE, alternate statement is executed otherwise */

- goto Label

/* jumps to the labeled statement represented as "Label: statement" */

4.10 Bitstream Parsing Operations

The bitstream is formatted as an ordered sequence of bytes. These bytes contain sequences of bits. The syntax elements appear within a sequence of bits in the order specified in the syntax tables of section 7 and section 9, and for each syntax element, the most-significant bit of the syntax element value is the left-most bit in the sequence of bits that represents the syntax element and the least-significant bit of the syntax element value is the right-most bit. The bits of the syntax elements shall be extracted from the bytes that represent them by extracting the most-significant bit of the first syntax element from the most-significant bit of the first byte, the next bit of the syntax element from the next less significant bit of the byte, etc., proceeding through to the least-significant bit of the byte and then the most-significant bit of the following byte, etc. After the bits of the first syntax element, the same convention shall be followed, starting at the next bit, for the bits of the next syntax element and then for the subsequent syntax elements.

Unless otherwise specified in a system-level specification, the bytes of the bitstream are ordered such that the first byte shall be placed first, the second byte shall be placed second, etc.

Note: The byte order described above is sometimes referred to as "network byte order".

The pseudo-code examples use the following bitstream parsing operations

new_bit()	// Returns the next bit from the bitstream
	static CurrentByte = 0;
	static CurrentBit = 7;
	
	extern unsigned char *ElementaryStream;
	
	new_bit() {
	Value = (ElementaryStream[CurrentByte] >>
	CurrentBit) & 1;
	if(CurrentBit != 0)
	CurrentBit--;
	else{
	CurrentByte++;
	CurrentBit = 7;
	}
	}

get_bits(n)	//Reads n bits from the bitstream and returns the value.
	// get_bits(0) is zero.
	get_bits(n) {
	DecodedBitString = 0;
	for(i = 0; i < n; i++)
	DecodedBitString = (DecodedBitString << 1) +
	new_bit();
	return(DecodedBitString);
	}

more_bits (BitString, Len, Tab)	Returns TRUE if the symbol BitString of length Len is not found in the code table Tab. Otherwise, it returns FALSE.

vlc_decode()	//Decodes the next variable-length codeword in the bitstream and returns the decoded symbol
	// Tab is set to appropriate variable length code table.
	vlc_decode() {
	DecodedBitString = new_bit();
	DecodedLength = 1;
	while(more_bits(DecodedBitString, DecodedLength,
	Tab) {
	DecodedBitString = (DecodedBitString << 1) +
	new_bit();
	DecodedLength++;
	}
	return(DecodedBitString);
	}

byte_aligned()	Returns TRUE if the correct position in the bitstream is on a byte boundary. Otherwise it returns FALSE.

4.11 Function Definitions

The functions smod (signed modulus), median3() and median4() are used in some of the pseudo-code examples in this spec. The functions median3 and median4 are computed as illustrated in the following pseudo-code examples.

median3 ()	median3 (a, b, c)
	{
	if (a > b)
	{
	if (b > c)
	median = b
	else if (a > c)
	median = c
	else
	median = a
	}
	else if (a > c)

	median = a
	else if (b > c)
	median = c
	else
	median = b
	
	return median
	}

median4 ()	median4 (a, b, c, d)
	{
	max = min = a
	if (b > max)
	max = b
	else if (b < min)
	min = b
	if (c > max)
	max = c
	else if (c < min)
	min = c
	if (d > max)
	max = d
	else if (d < min)
	min = d
	median = (a + b + c + d - max - min) / 2
	return median
	}

The smod (signed modulus) operator is used in computation of motion vector, and is computed as follows.

A smod b = ((A + b) & (2b – 1)) – b

A smod b lies within –b and b-1.

Note: The smod function is defined only for values of b that are powers of 2.

4.12 Definition of Terminology

For the purposes of this standard, the following definitions apply.

anchor picture: An I or a P picture or a skipped picture that is used as a reference picture for a B picture. There are two anchor pictures for a B picture.

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

B field picture: A B picture coded with interlace field coding mode. A B Field picture cannot be used for predicting any other picture except the opposite field of the same picture.

B frame picture: A frame structure B picture coded with interlace frame coding mode. A B Frame picture cannot be used for predicting any other picture.

B picture; **bidirectionally predictive-coded picture**: A picture that is coded using motion compensated prediction from past and/or future reference fields or frames. A B picture can contain macroblocks that are inter-coded, and macroblocks that are intra-coded. A B picture cannot be used for predicting any other picture.

BI picture: A B picture where all the macroblocks are intra-coded. A BI picture cannot be used for predicting any other picture.

backward motion vector: A motion vector that is used for motion compensation from a reference frame or reference field at a later time in display order.

backward prediction: Prediction from the future reference frame (field).

bitplane coding: Technique by which macroblock level information is coded as part of the frame header information.

backward reference frame distance (BRFD): As computed from syntax elements coded in the bitstream, this is nominally equal to one plus the number of frames between the current frame, and the subsequent (in display order) reference anchor.

bitstream: An ordered series of bits that forms the coded representation of the data.

bitstream data unit (BDU): A unit of the compressed data which may be parsed (i.e. syntax decoded) independently of other information at the same hierarchical level. A BDU could be, for example, a sequence header, an entry-point header, a coded picture or a slice. An Encapsulation Mechanism (EM) is described to prevent emulation of the start code prefix in the bitstream. The compressed data before encapsulation is called Raw Bitstream Decodable Unit (RBDU), while Encapsulated BDU (EBDU) refers to the data after encapsulation.

bitrate: The rate at which the coded bitstream is delivered to the input of a decoder.

block: An 8-row by 8-column matrix of samples, or 64 transform coefficients.

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

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.

byte: Sequence of 8 bits.

channel: A digital medium that stores or transports a bitstream.

clamping: The process of limiting values to a certain range, and can be implemented using the clip operator.

color-difference sub-sampling: The sampling grid used to sample the color-difference signals.

coded block pattern (CBP): A symbol indicating the presence or absence of residual information in blocks within a macroblock.

coded block pattern of color-difference and luma blocks (CBPCY): The six-bit pattern representing CBP of the two color-difference blocks and four luma blocks within a macroblock, and obtained from decoding the corresponding variable-length syntax element.

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.

coded video bitstream: A coded representation of a series of one or more sequences.

coded order: The order in which the pictures are transmitted and decoded. This order is the same as the display order if there are no B frames in the sequence. The coded order is not the same as the display order if there are B frames in the sequence.

coding parameters: The set of user-definable parameters that characterize a coded video bitstream.

coding set: The set of VLC tables and constants that are used to decode the AC coefficients.

color-difference: The C_b, C_r signals resulting from the matrix equations defined in the image source documents.

component: A matrix, block or single sample from one of the three matrices (luma and two color-difference) that make up a picture.

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

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

DC differential: The DC coefficient of a block in the bitstream that is differentially coded with respect to the DC-coefficient of a neighboring block.

decoder: An embodiment of a decoding process.

decoding process, decoding algorithm: The process defined whereby a serialized bitstream is converted to an array of 8-bit Y, C_b, C_r samples with 4:2:0 color subsampling. The decoding process does not include the display rendering process, which may convert these samples to images in another color space (such as RGB), may apply format specific black and white levels, color primaries, Y, C_b, C_r matrix coefficients, sample aspect ratios, etc., and may display the images with frequency and timing different from the sampled rate.

dequantization: The process of rescaling the quantized transform coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse transform.

direct prediction: Prediction from the both past reference frame (field), and the future reference frame (field) where the motion vectors are derived from the collocated block in the future frame (field).

display order: The order in which the decoded pictures are displayed. Normally this is the same order in which they were presented at the input of the encoder.

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

encoder: An embodiment of an encoding process.

encoding (process): A process which reads a stream of input pictures and produces a valid coded bitstream.

entry-point: A point in the bitstream that offers random access.

entry-point segment: The compressed bitstream (and the corresponding coded pictures) that is present between one entry-point, and the following entry-point.

escape code, ESCAPECODE: The escape code is used to represent a symbol for which is there is no direct representation in a VLC table.

extended motion vectors: Extended motion vectors are motion vectors which lie outside the default range. The default range of motion vectors is [-64 63.f] X [-32 31.f], where f is the fractional motion vector ¾ for ¼ pixel motion and ½ for ½ pixel motion resolution.

field: For an interlaced video signal, 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.

forward motion vector: A motion vector that is used for motion compensation from a reference frame or reference field at an earlier time in display order.

forward prediction: Prediction from the past reference frame (field).

forward reference frame distance (FRFD): As computed from syntax elements coded in the bitstream, this is nominally equal to one plus the number of frames between the current frame, and the previous (in display order) reference anchor.

frame: A frame contains 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.

Frame coding mode (FCM): The syntax element which indicates whether the frame is coded in progressive mode, interlace-field mode, or interlace-frame mode.

frame interpolation: The process of creating intermediate video frames (for display) based on the data in two consecutive frames of decoded video. Frame interpolation is not part of the decoding process.

frame rate: The rate at which frames are output from the decoding process.

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.

frame re-ordering: The process of re-ordering the reconstructed frames when the coded order is different from the display order. Frame re-ordering occurs when B frames are present in a bitstream. There is no frame re-ordering when decoding low delay bitstreams.

half pel, hpel: A position that is half pixel away from integer pixel position.

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.

hypothetical reference decoder (HRD): hypothetical reference decoder is an alternate term for video buffering verifier.

II picture: A Picture coded with interlace field coding mode where both fields are coded as an I Field.

inter-coded block, inter-block: A block that been coded using information both from itself, and from blocks and pictures occurring at other times.

inter coding: Coding of a macroblock or picture that uses information both from it and from macroblocks and pictures occurring at other times.

interlace: The property of 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 may be the top field or the bottom field of the frame.

interlace field coding: The coding mode used when the two fields of an interlace frame are coded separately. The pictures coded using this mode are called interlace field pictures, or field pictures.

interlace frame coding: The coding mode used when the two fields of an interlace frame are coded together. The pictures coded using this mode are called interlace frame pictures, or frame pictures.

inter macroblock, inter MB: A macroblock that has been coded using information both from itself, and from pictures occurring at other times.

interpolation: the process used to generate subpixel values when the motion vectors are not integers.

interpolated prediction: Prediction from the both past reference frame (field), and the future reference frame (field) where the motion vectors are explicitly coded in the bitstream.

I field picture: An I Picture coded with interlace field coding mode.

I interlace frame picture: An I Picture coded with interlace frame coding mode.

IP picture: A Picture coded with interlace field coding mode where the first field is coded as an I Field, and the second field is coded as a P Field.

I picture; intra-coded picture: A picture coded using information only from itself. All the macroblocks in an I-picture are intra-coded.

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

intra-coded block, intra-block: A block that been coded using information only from that block or picture.

intra macroblock, intra MB: A macroblock that has been coded using information only from that block or picture.

level: A defined set of constraints on the values which may be taken by the parameters (such as bit rate and buffer size) within a particular profile. A profile may contain one or more levels. Levels are hierarchical. A bitstream compliant to a particular combination of level and profile is compliant to all higher levels at the same profile.

In a different context, level is the absolute value of a non-zero coefficient (see "run").

luma; Y': is the value resulting from a weighted sum of 3 nonlinear (gamma pre-corrected) R,G,B components. It is often carelessly called luminance and given the symbol Y.

macroblock: The four 8 by 8 blocks of luma data and the two corresponding 8 by 8 blocks of color-difference data coming from a 16 by 16 section of the luma component of the picture.

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.

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

motion vector (MV): 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.

natural scan order: The scan order in which a two-dimensional array of symbols is scanned row-wise from left to right, and the rows are scanned from top to bottom.

non-skipped macroblock: A macroblock which is not skipped, and for which data is coded.

opposite field: A field whose polarity is opposite to that of the current field. Note: the current and opposite fields are not bound to a single frame.

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

out-of-loop processing: Out-of-loop processing consists of operations such as resizing the output video, color-difference upsampling, and frame interpolation which are outside the decoding loop.

overlap smoothing: The filtering operation that is conditionally performed across edges of two neighboring Intra blocks, after inverse transform and prior to the loop filter.

overscan: The amount of picture area that gets cropped off along the edges.

P field picture: A P Picture coded with interlace field coding mode.

P interlace frame picture: A P Picture coded with interlace frame coding mode.

PI Field picture: A Picture coded with interlace field coding mode where the first field is coded as a P Field, and the second field is coded as an I Field.

PP picture: A Picture coded with interlace field coding mode where both fields are coded as a P Field.

P picture; predictive-coded picture: A picture that is coded using motion compensated prediction from past reference fields or frame. A P picture can contain macroblocks that are inter-coded (i.e. coded using prediction) and macroblocks that are intra-coded.

pan scan window: The portion of video displayed on a screen as a result of the view selection.

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

parity (of field): The parity of a field may be top or bottom.

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.

pel: an alternate term for pixel.

picture: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luma and two color-difference signals. For progressive video, a picture is identical to a frame, while for interlaced video, a picture may refer to a frame, or the top field or the bottom field of the frame depending on the context.

post-processing: Post-processing consists of two operations: deblocking and deringing. Decoder can use either one, or both, of these operations to mitigate the effect of compression artifacts, and improve perceptual quality of video.

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

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

prediction loop: The parts of the decoding process that have an effect in producing the reference frame(s) are said to be part of the prediction loop; other parts of the decoding process are said to be outside the decoding loop.

previous entry-point: The closest entry-point that temporally precedes the current entry-point in the sequence.

profile: A defined subset of the syntax of the standard, with a specific set of coding tools, algorithms, and syntax associated with it. There are three profiles: simple, main and advanced.

progressive: The property of frames where all the samples of the frame represent the same instance in time.

pull-back operation: An operation in which a motion vector, which points to a region entirely outside the frame boundaries of a reference frame, is adjusted so that it points to a region which is at least partially inside the frame boundaries.

pull-down: Pull-down is a process where frame rate is increased through frame or field replication, such as when 24-frame-per-second film is expanded to 60-frame-per-second video. Compression efficiency can be improved by coding only the unique frames and providing hints to the display process to re-create the original sequence.

quantize, quantization: A process in which the continuous range of values of an input signal is divided into non-overlapping (but not necessarily equal) subranges, and a discrete, unique value is assigned to each subrange. A unique index is generated to represent this value.

quarter pel, qpel: A position that is quarter pixel away from integer pixel position.

random access: A random access point in the bitstream is defined by the following guarantee: If decoding begins at this point, all frames needed for display after this point will have no decoding dependency on any data preceding this point, and are also present in the decoding sequence after this point. A random access point is also called an entry-point.

range mapping: The process of rescaling decoded pixel values in advanced profile. Luma and color-difference values may be scaled differently, and the coefficients used for scaling are transmitted in the entry point header. This process is outside the prediction loop, and is performed as the last stage in decoding. This technique can be used to reduce the bitrate.

range reduction: The process of rescaling decoded pixel values in main profile. Luma and color-difference values are scaled by a factor of 2, if range reduction is signaled for that picture. This process is part of the prediction loop. This technique can be used to reduce the bit rate.

raw mode: A mode of bitplane coding in which the corresponding macroblock level information is coded as part of the macroblock syntax, and not as part of the picture header.

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.

re-ordering delay: A delay in the decoding process that is caused by frame re-ordering.

repeat pad: The process of repeatedly padding the boundary pixels of the reference picture in the horizontal and vertical direction to allow the motion vectors to point outside the frame boundary.

repeat pad region: The region beyond the frame boundary of the reference picture which has been repeat-padded with boundary pixels.

rounding: The process of adjusting the bias before a division (or shift) operation.

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".

same field: A field whose polarity is same as that of the current field.

sample aspect ratio: Specifies, for assisting the display process, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the luma sample array in a frame. Sample aspect ratio is expressed as h:v, where h is horizontal width and v is vertical height (in arbitrary units of spatial distance).

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

sequence: A coded representation of a series of one or more pictures. In the advanced profile, a sequence consists of a series of one or more entry-point segments, where each entry-point segment consists of a series of one or more pictures, and where the first picture in each entry-point segment provides random access. In the simple and main profiles, the first picture in each sequence shall be an intra coded picture.

size of coded picture: The size (in number of bytes) of coded bitstream that represents one picture.

skipped macroblock: A macroblock for which no data are encoded.

skipped picture: A skipped picture is a P-picture that is identical to its reference picture.

slice: A consecutive series of macroblock rows in a picture, which are encoded as a single unit.

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

start codes (SC): 32-bit codes embedded in that coded bitstream that are unique, and identify the beginning of a BDU. Start codes consist of a unique three-byte Start Code Prefix (SCP), and a one-byte Start Code Suffix (SCS).

stuffing bytes: Zero-byte code-words that may be inserted into the coded bitstream, before a start-code, and after flushing bits, that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream which would otherwise be lower than the desired bitrate.

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.

variable bitrate: Operation where the bitrate varies with time during the decoding of a coded bitstream.

variable-sized transform decoding: Technique where an 8x8 error block may be transformed using one 8x8 transform, or divided vertically and transformed with two 8x4 transforms or divided horizontally and transformed with two 4x8 transforms, or divided into 4 quadrants and transformed with four 4x4 transforms.

variable length coding (VLC): A reversible procedure for coding that assigns shorter code-words to symbols of higher probability and longer code-words to symbols of lower probability.

The acronym VLC also indicates a variable length code.

VC-1: This is the name of the standard described here.

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.

video sequence: The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded frames.

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

4.13 Intermediate Variables

The following is a list of important intermediate variables that are used in the decoding process:

PQUANT: is the picture quantizer scale (step size) that is derived from the syntax element PQINDEX as defined in 7.1.1.6.

ALTPQUANT: is the alternate picture quantizer step size that is derived, from the syntax elements that are part of VOPDQUANT, as defined in 7.1.1.31.

MQUANT: is the macroblock quantizer step size that is derived from the syntax elements PQUANT, ALTPQUANT, MQDIFF and ABSMQ as defined in 7.1.3.4.

RND: is the variable used for rounding control and is derived as defined in 8.3.7.

CodedWidth: is the width of the coded frame, and derived from entry point header (6.2.13.1) or sequence header (6.1.6) in advanced profile, and derived from meta-data element HORIZ_SIZE (Annex J.1.3) in simple/main profile.

CodedHeight: is the height of the coded frame, and derived from entry point header (6.2.13.2) or sequence header (6.1.7) in advanced profile, and derived from meta-data element VERT_SIZE (Annex J.1.4) in simple/main profile.

4.14 Acronym Definitions

The following acronyms are commonly used.

BDU: Bitstream Data Unit.

BDU: Coded Block Pattern.

CBPCY: Coded Block Pattern of Color-difference and Luma blocks.

FCM: Frame Coding Mode.

HRD: Hypothetical Reference Decoder.

ITrans: Inverse Transform.

LSB: Least Significant Bit.

MB: MacroBlock.

MSB: Most Significant Bit.

MV: Motion Vector.

NA: Not Applicable.

RLD: Run Length Decode.

SC: Start Code.

Trans: Transform.

VBV: Video Buffering Verifier.

VLC: Variable Length Code.

VLD: Variable Length Decoding.

4.15 Guide to Interpreting Syntax Diagrams and Syntax Elements

A guide for interpretation of the diagrams consists of the following:

1. Arrow paths show the possible flows of syntax elements. Any syntax element which has zero length is considered absent for arrow path diagramming
2. Abbreviations and semantics for each syntax element are as defined in later clauses.
3. Syntax elements shown with square-edged boundaries indicate fixed-length syntax elements; those with rounded boundaries indicate variable-length syntax elements and those with a rounded boundary within an outer rounded boundary indicate a syntax element made up of simpler syntax elements which are elaborated on in another section.
4. A fixed-length syntax element is defined to be a syntax element for which the length of the syntax element is not dependent on the data in the content of the syntax element itself. The length of this syntax element is either always the same, or is determined by the prior data in the syntax flow.

The term "layer" is used to refer to any part of the syntax that may be understood and diagrammed as a distinct entity. The next-lower layer element in a layer diagram is indicated by a rectangle within a rectangle.

It is often convenient to denote the elements in binary representation. To avoid confusion with decimal representation, whenever a number is expressed in binary format, a suffix of lower case 'b' is used.

Unless specified otherwise, the most-significant bit is transmitted first. This is bit 1 and is the leftmost bit in the code tables in this document. Unless specified otherwise, all unused or spare bits are set to "0". All values of syntax not explicitly defined in this document are SMPTE reserved for future use.

5 Picture Sampling and Overall Bitstream Structure

5.1 Introduction

There are two fundamental picture sampling structures: progressive and interlaced. The progressive sampling structure assumes that all sample rows of the frame are sequential. The interlaced sampling structure assumes that top fields are acquired within one time interval and the bottom fields are acquired within a different time interval.

5.2 Progressive Coding Mode

5.2.1 Input/output Format

Picture sampling shall use the Y, C_b, C_r color model with 4:2:0 sampling. Figure 7 below shows the spatial relationship between the luma and color difference samples of the Y, C_b, C_r 4:2:0 interlaced format in the horizontal and vertical axes. The figure also shows the spatial relationship between the luma and color-difference samples.

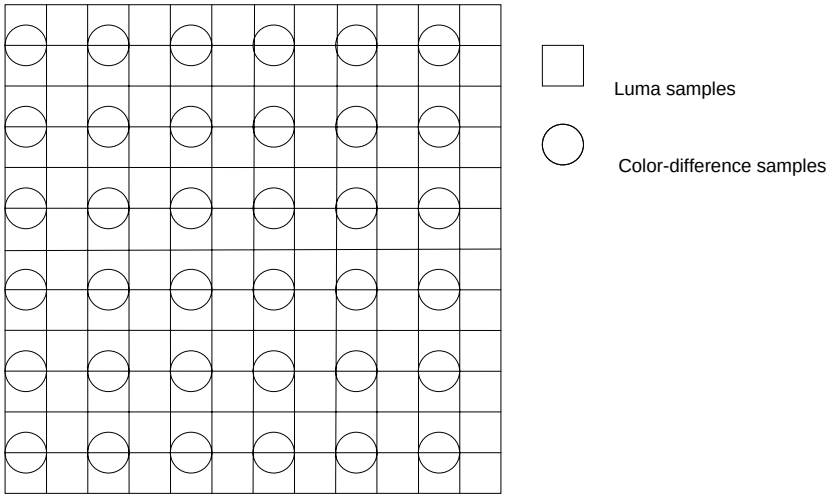


Figure 7: 4:2:0 Luma and color-difference sample horizontal and vertical positions

Note: Many interfaces operate using 4:2:2 and 4:4:4 sampling. For such interfaces, a sample down-converter will be required prior to the encoder and a sample up-converter following the decoder. The down/up converters required for interfaces using 4:2:2 and 4:4:4 sampling do not form part of this standard.

5.2.2 Hierarchical Elements

The bitstream syntax consists of the hierarchical layers:

- sequence
- entry-point
- picture
- slices
- macroblocks (MB)
- blocks

In the advanced profile, entry-point layer shall be present between the sequence and picture layers to signal a random access point in the bitstream. Further, in the advanced profile, an optional slice layer may be present between the picture layer and the macroblock layer. A slice contains one or more contiguous rows of macroblocks in their original left-to-right order. A slice shall begin at the first macroblock of a row, and end at the last macroblock of the same or another row. The entry-point and slice layers are present only in advanced profile. Figure 8 illustrates the picture, macroblock, slice and block layers.

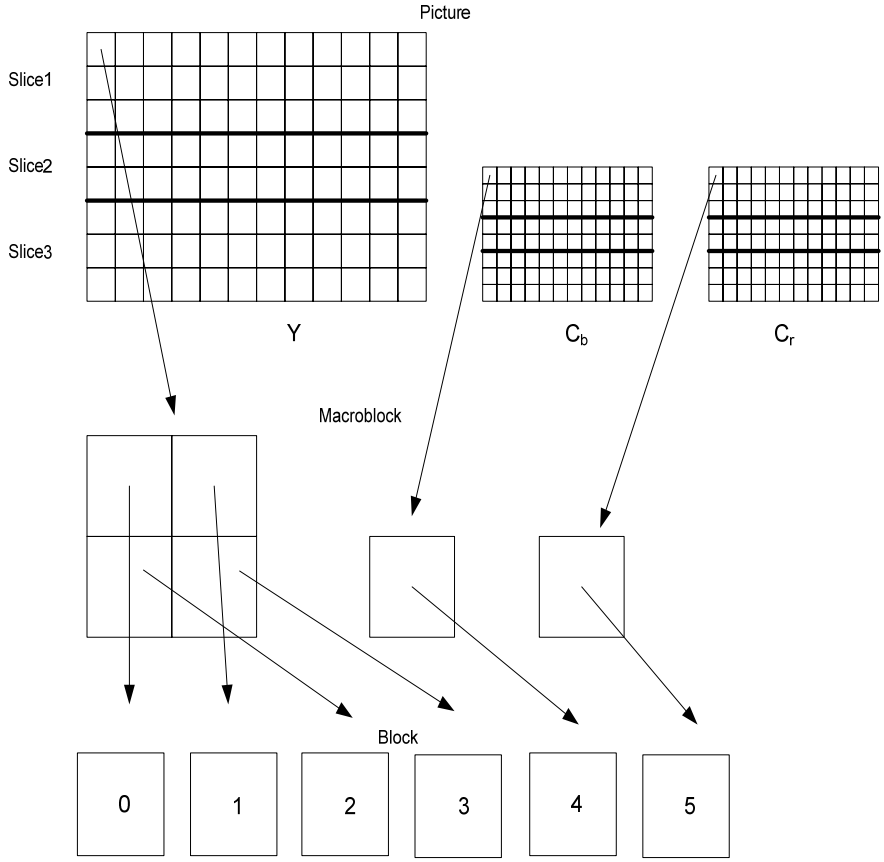


Figure 8: Coding Hierarchy showing Picture, Slice, Macroblock and Block layers

5.3 Interlace Coding Mode

5.3.1 Input/Output Format for 4:2:0 Interlace

Interlaced picture sampling shall use the Y, C_b, C_r color model with 4:2:0 sampling. Figure 7 shows the spatial relationship between the luma and color-difference samples in the Y, C_b, C_r 4:2:0 interlaced format in the horizontal and vertical axes. Figure 9 shows the relationship between vertical sample position and sampling time instant. The vertical axis in the figure corresponds to the vertical axis in the frame, and the horizontal axis in the figure corresponds to the temporal axis. Each frame is represented with an "edge-on" view. Note that Figure 9 does not show the spatial relationship between horizontal and vertical sampling positions.

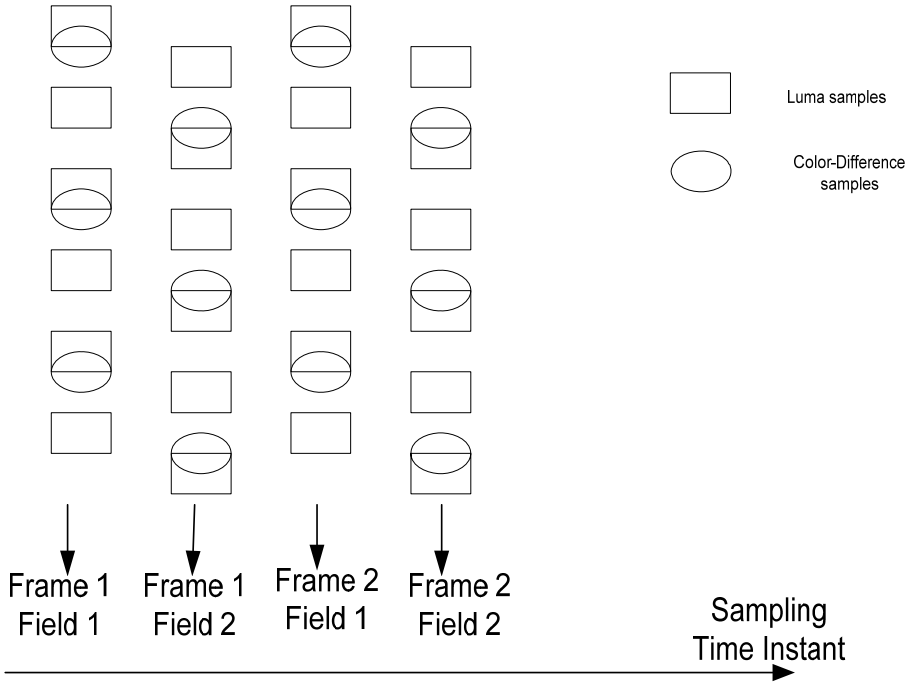


Figure 9: 4:2:0 Luma and color-difference temporal and vertical sample positions shown relative to sampling time instant (where from left to right is shown a top field, bottom field, top field, and bottom field)

5.4 Frame Ordering

Dependence of B frames on temporally past and future anchor I or P frames dictates the ordering of coded and displayed frames. The following rules shall apply to the ordering of frames in a VC-1 bitstream:

1. A frame in a VC-1 bitstream shall be entirely decodable from the information contained in the current and previously coded frames.
2. The display or output order of frames in a VC-1 bitstream shall be identical to the coded order when B frames are not indicated in the bitstream.
3. The display or output order of frames in a VC-1 bitstream shall be different from the coded order when B frames are indicated in the bitstream (whether or not they are actually present), as follows:
    a. B frames shall be output or displayed in the same order as they are encoded.
    b. Anchor (I and P) frames shall be output or displayed at the sequence slot of the subsequently coded anchor frame. Therefore, there shall be no output frame at the first coded instant. The last coded anchor frame shall be displayed subsequent to the last coded frame.

These rules are illustrated in Table 2:

Table 2: Frame Ordering Rules for bitstreams containing B-pictures

	Input (coded) order	Output (display) order
No B frames	**I0 P1 P2 P3 I4 …**	**I0 P1 P2 P3 I4 …**
B frames indicated	**I0 P1 P2 P3 I4 …**	**X I0 P1 P2 P3 I4…**
B frames indicated	**I0 P1 P2 B3 B4 P5 …**	**X I0 P1 B3 B4 P2 …**
B frames indicated	**… P6 B7 B8 <end>**	**… Pn B7 B8 P6 <end>**
B frames indicated	**… P6 B7 B8 P9 <end>**	**… Pn B7 B8 P6 P9 <end>**

Pn represents the nth P picture, where the value of n is inferred from the context. X represents the absence of an output (display) picture.

Note: On the encoder side, the input frames may be reordered prior to coding so as to ensure that the decoded order is identical to the input order, subject to a constant offset. Typically, this offset is equal to the maximum number of B frames in the sequence.

5.5 Constraints

5.5.1 Minimum and maximum frame sizes

For progressive frames, the frame height and frame width shall be a multiple of 2. For interlaced frames, the frame height shall be a multiple of 4, and the frame width shall be a multiple of 2. The maximum dimensions of the frame are limited by the target profile and level of the bitstream as listed in Annex D. For more details on internal representation of frames, and handing of frame dimensions which are not multiples of 16, see Annex K.2.

5.5.2 Maximum size of compressed bits

The data size corresponding to any macroblock row shall not exceed the greater of: (i) 6144 bits, or (ii) 1536 bits times the number of macroblocks in the horizontal direction.

Compressed data corresponding to a macroblock row shall be defined to contain all the contiguous entropy coded information required to decode the entire row of macroblocks, subject to availability of causal information from the preceding macroblock row, and frame, field or slice-level header data. Therefore, the macroblock row contains – besides the coded transform coefficients – motion vectors, and macroblock header elements such as the coded block pattern and field/frame coding type.

If the macroblock information (such as 1 or 4-MV) is coded as part of the frame header (as a bitplane), the bits used for this information are outside of the constraint. Instead, if this information is coded as part of the macroblock layer syntax, bits used in coding this information are to be included in the macroblock row size calculation.

Slice header information, where present, is not included in the calculation of the macroblock row data size. Any zero-valued stuffing bytes, and start-codes, are also not included in the macroblock row data size.

The following three examples illustrate this constraint:

Example 1 – Frame size 300×200, coded as progressive: Number of horizontal macroblocks is ceil(300/16) = 19. Maximum compressed data size of macroblock row = max(6144, 19×1536) =29184 bits.

Example 2 – Frame size 720×480, coded as interlace: Number of horizontal macroblocks is ceil(720/16) = 45. Maximum compressed data size of macroblock row = max(6144, 45×1536) = 69120 bits.

Example 3 – Frame size 40×40, coded as progressive: Number of horizontal macroblocks is ceil(40/16) = 3. Maximum compressed data size of macroblock row = max(6144, 3×1536) = 6144 bits.

5.5.3 Bitstream Construction Constraints

The first frame in a bitstream conformant to either the simple profile or the main profile shall be an I frame. A bitstream conformant to the advanced profile shall be constructed and constrained according to Annex G.

6 Sequence And Entry-Point Bitstream Syntax and Semantics

The bitstream syntax and semantics of the sequence and entry-point layer of the advanced profile are described in this section.

In the simple and main profiles, the sequence-related metadata shall be communicated to the decoder by the transport layer or other means. Annex J defines the syntax and semantics of this metadata required for the decoder. Note: For an example of a sequence-level parameter transport, see SMPTE RP 227.

In the advanced profile, the sequence-related metadata is part of the video data bitstream and the syntax and semantics are defined in this section. The presence of this sequence metadata is subject to the rules defined in Annex G.

6.1 Sequence-level Syntax and Semantics

A sequence-level header contains sequence-level parameters used to decode the sequence of compressed pictures.

Figure 10 shows the bitstream elements that make up the sequence header for the advanced profile and Table 3 defines the bitstream syntax.

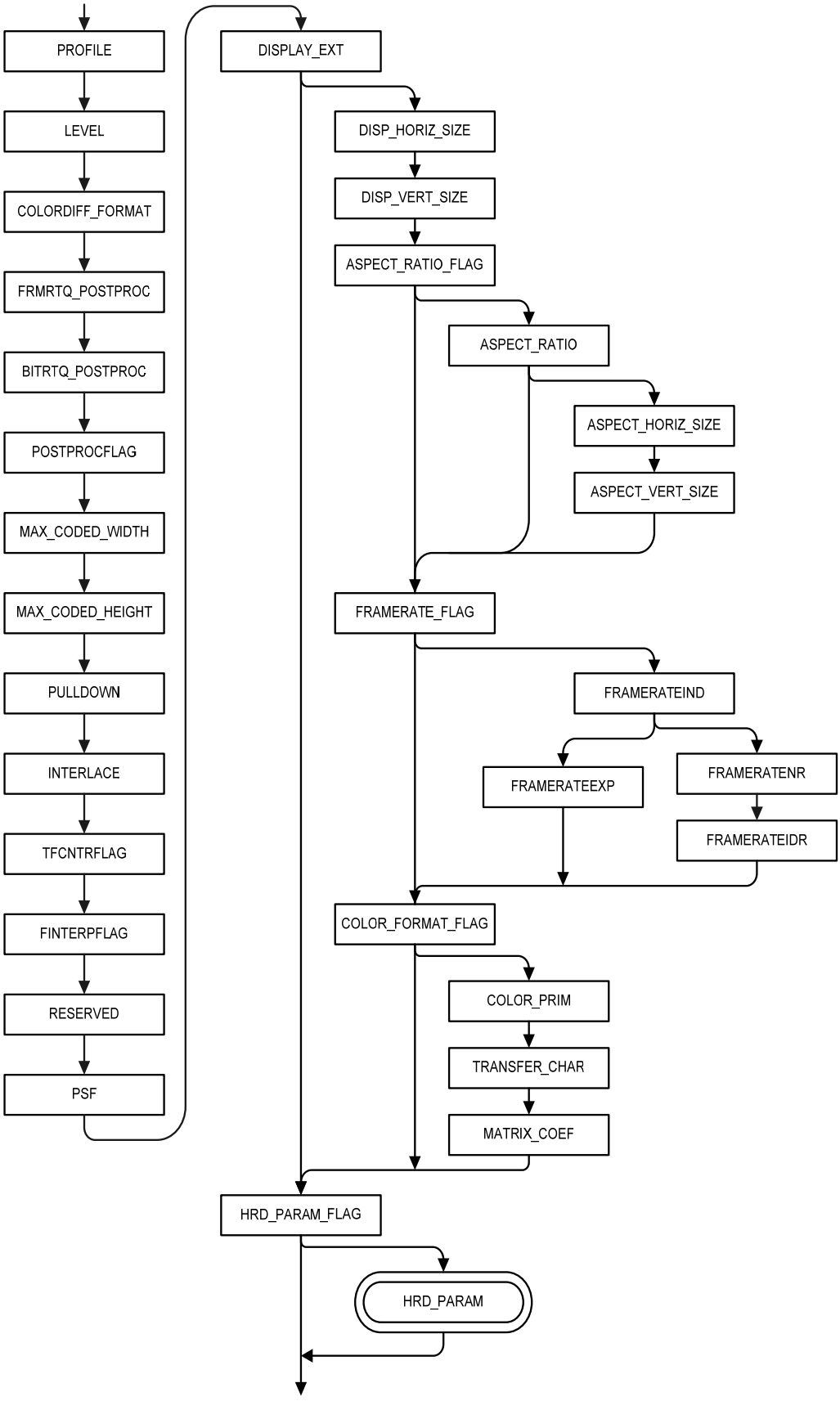


Figure 10: Syntax diagram for the sequence layer bitstream for the Advanced Profile

The following table shows the syntax elements of the sequence layer used for the advanced profile. This structure is not present for simple and main profiles.

Table 3: Sequence layer bitstream for Advanced Profile

SEQUENCE LAYER( ) {	Number of bits	Descriptor	Reference
PROFILE	2	uimsbf	6.1.1
LEVEL	3	uimsbf	6.1.2
COLORDIFF_FORMAT	2	uimsbf	6.1.3
FRMRTQ_POSTPROC	3	uimsbf	6.1.4.1
BITRTQ_POSTPROC	5	uimsbf	6.1.4.2
POSTPROCFLAG	1	uimsbf	6.1.5
MAX_CODED_WIDTH	12	uimsbf	6.1.6
MAX_CODED_HEIGHT	12	uimsbf	6.1.7
PULLDOWN	1	uimsbf	6.1.8
INTERLACE	1	uimsbf	6.1.9
TFCNTRFLAG	1	uimsbf	6.1.10
FINTERPFLAG	1	uimsbf	6.1.11
RESERVED	1	uimsbf	6.1.12
PSF	1	uimsbf	6.1.13
DISPLAY_EXT	1	uimsbf	6.1.14
if (DISPLAY_EXT == 1) {			
DISP_HORIZ_SIZE	14	uimsbf	6.1.14.1
DISP_VERT_SIZE	14	uimsbf	6.1.14.2
ASPECT_RATIO_FLAG	1	uimsbf	6.1.14.3
if (ASPECT_RATIO_FLAG == 1) {			
ASPECT_RATIO	4	uimsbf	6.1.14.3.1
if (ASPECT_RATIO == ‘15’) {			

ASPECT_HORIZ_SIZE	8	uimsbf	6.1.14.3.2
ASPECT_VERT_SIZE	8	uimsbf	6.1.14.3.3
}			
}			
FRAMERATE_FLAG	1	uimsbf	6.1.14.4
if (FRAMERATE_FLAG == 1) {			
FRAMERATEIND	1	uimsbf	6.1.14.4.1
if (FRAMERATEIND == 0) {			
FRAMERATENR	8	uimsbf	6.1.14.4.2
FRAMERATEDR	4	uimsbf	6.1.14.4.3
} else {			
FRAMERATEEXP	16	uimsbf	6.1.14.4.4
}			
}			
COLOR_FORMAT_FLAG	1	uimsbf	6.1.14.5
if (COLOR_FORMAT_FLAG ==1) {			
COLOR_PRIM	8	uimsbf	6.1.14.5.1
TRANSFER_CHAR	8	uimsbf	6.1.14.5.2
MATRIX_COEF	8	uimsbf	6.1.14.5.3
}			
}			
HRD_PARAM_ FLAG	1	uimsbf	6.1.15
if (HRD_PARAM_ FLAG == 1) {			
HRD_PARAM( )			6.1.15.1
}			
}			

6.1.1 Profile (PROFILE)(2 bits)

PROFILE is a 2-bit syntax element that specifies the profile used to encode the sequence, and shall be set to 3 to indicate advanced profile. The values 0, 1, and 2 are SMPTE Reserved. The SMPTE reserved values may be used in future to define additional profiles. The relation of profiles to coding tools is summarized in Annex D.

6.1.2 Level (LEVEL)(3 bits)

LEVEL is a 3-bit syntax element and specifies the encoding level for the clip in the advanced profile. The codes that are used to signal the levels in the advanced profile shall be as defined in Table 4.

Table 4: Meaning of LEVEL syntax element

LEVEL	Meaning
000	Level 0
001	Level 1
010	Level 2
011	Level 3
100	Level 4
101-111	SMPTE Reserved

The coding limitations of the levels are defined in Annex D.3

6.1.3 Color-Difference Format (COLORDIFF_FORMAT) (2 bits)

The COLORDIFF_FORMAT syntax element is a 2-bit syntax element that indicates the color-difference/luma format used to represent each picture. The formats shall be as defined in Table 5.

Table 5: Meaning of COLORDIFF_FORMAT syntax element

COLORDIFF_FORMAT	Format
0	SMPTE Reserved
1	4:2:0
2	SMPTE Reserved
3	SMPTE Reserved

Only the value 1 corresponding to format 4:2:0 is permitted for this field. All other values are SMPTE Reserved.

6.1.4 Post processing Indicators

The post-processing indicators, described in section 6.1.4.1 and 6.1.4.2 provide a mechanism which may be used to control the post-processing operation. The pseudo-code in Table 6, defines the decoding procedure for post-processing indicators. Annex I.6 describes a mechanism for controlling post-processing using the post-processing indicators.

6.1.4.1 Quantized Frame Rate for Post processing Indicator (FRMRTQ_POSTPROC)(3 bits)

FRMRTQ_POSTPROC is a 3-bit syntax element that signals the (quantized) frame rate information as described in Table 6.

6.1.4.2 Quantized Bit Rate for Post processing Indicator (BITRTQ_POSTPROC)(5 bits)

BITRTQ_POSTPROC is a 5-bit syntax element that signals the (quantized) bit rate information as described in Table 6.

Table 6: Decoding Procedure for Post-processing Indicators in Advanced Profile

if (FRMRTQ_POSTPROC == 0) && (BITRTQ_POSTPROC == 31) {
Post processing indicators for Frame Rate and Bit Rate are undefined
}
else if ((FRMRTQ_POSTPRC == 0) && (BITRTQ_POSTPROC == 30)) {
“frame rate” is around 2 frames/second
“bit rate” is around 1952 kbps or more
}
else if ((FRMRTQ_POSTPROC == 1) && (BITRTQ_POSTPROC == 31)) {
“frame rate” is around 6 frames/second
“bit rate” is around 2016 kbps or more
}
else {
if (FRMRTQ_POSTPROC == 7) {
“frame rate is around 30 frames/second or more
}
else {
“frame rate is around “(2+FRMRTQ_POSTPROC*4)” frames/second
}
if (BITRTQ_POSTPROC == 31) {
“bit rate” is around 2016 kbps or more
}
else {
“bit rate” is around “(32 + BITRTQ_POSTPROC * 64)” kbps
}
}

6.1.5 Post processing Flag (POSTPROCFLAG) (1 bit)

POSTPROCFLAG is a 1-bit syntax element that shall indicate whether syntax element POSTPROC is present in picture headers. If POSTPROCFLAG == 1, then POSTPROC shall be present in picture headers. If POSTPROCFLAG == 0, then POSTPROC shall not be present in picture headers.

6.1.6 Maximum Horizontal Size of Picture (MAX_CODED_WIDTH)(12 bits)

MAX_CODED_WIDTH specifies the maximum horizontal size of the coded picture within the sequence in units of pixels. This syntax element shall be a 12-bit unsigned integer encoding of size. The maximum horizontal size of the picture shall be equal to the value of this field multiplied by 2, plus 2. The horizontal size of the coded picture may change at an entry point and shall be less than, or equal to, MAX_CODED_WIDTH.

6.1.7 Maximum Vertical Size of Picture (MAX_CODED_HEIGHT)(12 bits)

MAX_CODED_HEIGHT specifies the maximum vertical size of the coded picture within the sequence in units of pixels. This syntax element shall be a 12-bit unsigned integer encoding of size. The maximum vertical size of the picture shall be equal to the value of this field multiplied by 2, plus 2. The vertical size of the coded picture may change at an entry point and shall be less than, or equal to, MAX_CODED_HEIGHT.

Note: The maximum vertical size of the coded picture is also subject to the constraints specified in 5.5.1.

6.1.8 Pull down Flag (PULLDOWN) (1 bit)

PULLDOWN is a 1-bit syntax element that shall indicate if the syntax elements RPTFRM, or TFF and RFF are present in picture headers. If PULLDOWN == 0, these syntax elements shall not be present. If PULLDOWN == 1, the presence of these syntax elements is defined in 7.1.1.17, 7.1.1.18 and 7.1.1.19.

6.1.9 Interlace Content (INTERLACE) (1 bit)

INTERLACE is a 1-bit syntax element. The individual frames may be coded using the progressive or interlace syntax when INTERLACE == 1. If INTERLACE == 0, pictures shall be coded as single frames using the progressive syntax.

6.1.10 Frame Counter Flag (TFCNTRFLAG) (1 bit)

TFCNTRFLAG is a 1-bit syntax element. TFCNTRFLAG == 1 indicates that the syntax element TFCNTR shall be present in the advanced profile picture headers. TFCNTRFLAG == 0 indicates that TFCNTR shall not be present in the picture header.

6.1.11 Frame Interpolation Flag (FINTERPFLAG)(1 bit)

FINTERPFLAG is a 1-bit syntax element that indicates if the syntax element INTERPFRM is present in the picture header. If FINTERPFLAG == 1, then INTERPFRM shall be present in picture headers. If FINTERPFLAG == 0, then INTERPFRM shall not be present in picture headers.

6.1.12 Reserved Advanced Profile Flag (RESERVED)(1 bit)

RESERVED is a 1-bit syntax element and shall be set to 1. The value 0 is SMPTE Reserved.

6.1.13 Progressive Segmented Frame (PSF)(1 bit)

PSF is a 1-bit syntax element. If PSF == 1, the video source was Progressive Segmented Frame (PsF), and the display process should treat the decoded frames (field-pairs) as progressive. If PSF == 0, the display process may treat the decoded frames (field-pairs) according to the value of the INTERLACE syntax element.

6.1.14 Display Extension Flag (DISPLAY_EXT) (1 bit)

DISPLAY_EXT is a 1-bit syntax element. If DISPLAY_EXT == 1 then DISP_HORIZ_SIZE, DISP_VERT_SIZE, and ASPECT_RATIO_FLAG shall be present in the sequence header. If DISPLAY_EXT == 0, then these elements shall not be present.

6.1.14.1 Horizontal Display Size of Picture (DISP_HORIZ_SIZE)(14 bits)

DISP_HORIZ_SIZE is a 14-bit syntax element that shall be present only if DISPLAY_EXT is 1. It specifies the horizontal display size of the picture in pixels. This syntax element is a 14-bit unsigned integer value representing (size-1), and may represent sizes ranging from 1 to 16384.

Note: The horizontal display size is not used for decoding, but is used during display.

6.1.14.2 Vertical Display Size of Picture (DISP_VERT_SIZE)(14 bits)

DISP_VERT_SIZE is a 14-bit syntax element that shall be present only if DISPLAY_EXT is 1. It specifies the vertical display size of the picture in pixels. This syntax element is a 14-bit unsigned integer value representing (size-1), and may represent sizes ranging from 1 to 16384.

Note: DISP_VERT_SIZE is not used for decoding, but is used for display.

6.1.14.3 Sample Aspect Ratio Indicator Flag (ASPECT_RATIO_FLAG)(1 bit)

ASPECT_RATIO_FLAG is a 1-bit syntax element that shall be present only if DISPLAY_EXT == 1. If ASPECT_RATIO_FLAG == 1, the syntax element ASPECT_RATIO shall be present. If ASPECT_RATIO_FLAG == 0, the syntax element ASPECT_RATIO shall not be present.

6.1.14.3.1 Sample Aspect Ratio (ASPECT_RATIO)(4 bits)

ASPECT_RATIO is a 4-bit syntax element that shall be present only if the ASPECT_RATIO_FLAG == 1 and DISPLAY_EXT == 1. ASPECT_RATIO specifies the encoded sample aspect ratio for the sequence. If ASPECT_RATIO takes the value ‘15’, the syntax elements ASPECT_HORIZ_SIZE and ASPECT_VERT_SIZE shall be present.

Note: The sample aspect ratio is often referred to as the pixel aspect ratio.

The value of the sample aspect ratio for each value of the ASPECT_RATIO syntax element shall be as defined in **Table 7**.

ASPECT_RATIO	Sample Aspect Ratio	(Informative) Examples of Use
0	Unspecified	
1	1:1 (“square”)	1280x720 16:9 frame without overscan Coded Image 1920x1088 Source Image 1920x1080 16:9 frame without overscan 640x480 4:3 frame without overscan
2	12:11	Coded Image 704x576 Source Image 720x576 (4:3 frame with horizontal overscan) 352x288 4:3 frame without overscan
3	10:11	Coded Image 704x480 Source Image 720x486 (4:3 frame with horizontal overscan) 352x240 4:3 frame without overscan
4	16:11	Coded Image 704x576 Source Image 720x576 (16:9 frame with horizontal overscan) 540x576 4:3 frame with horizontal overscan
5	40:33	Coded Image 704x480 Source Image 720x486 (16:9 frame with horizontal overscan) Coded Image 528x480 Source Image 540x486 4:3 frame with horizontal overscan
6	24:11	352x576 4:3 frame without overscan 480x576 16:9 frame with horizontal overscan
7	20:11	352x480 4:3 frame without overscan 480x480 16:9 frame with horizontal overscan
8	32:11	352x576 16:9 frame without overscan
9	80:33	352x480 16:9 frame without overscan
10	18:11	480x576 4:3 frame with horizontal overscan
11	15:11	480x480 4:3 frame with horizontal overscan
12	64:33	Coded Image 528x576 Source Image 540x576 16:9 frame with horizontal overscan
13	160:99	Coded Image 528x480 Source Image 540x486 16:9 frame with horizontal overscan
14	SMPTE Reserved	
15	Aspect width and height transmitted.	

Table 7: Meaning of ASPECT_RATIO syntax element

Note: For those entries in **Table 7** that indicate "with horizontal overscan" in the Examples of Use column, the active image area is not expected to fill all of the active pixels on each line. Consequently, the calculation of aspect ratio given in the sample aspect ratio column is based on the coded picture area and cannot be determined directly from the ratio of the size values of the overall pixel matrix. Rather, the aspect ratio is determined from the coded picture area.

6.1.14.3.2 Aspect Width (ASPECT_HORIZ_SIZE)(8 bits)

ASPECT_HORIZ_SIZE is an 8-bit syntax element that shall be present only if ASPECT_RATIO_FLAG == 1, ASPECT_RATIO == ‘15’, and DISPLAY_EXT == 1. ASPECT_HORIZ_SIZE specifies the horizontal aspect size of the sample. This syntax element shall be a binary encoding of sizes ranging from 1 to 256.

6.1.14.3.3 Aspect Height (ASPECT_VERT_SIZE)(8 bits)

ASPECT_VERT_SIZE is an 8-bit syntax element that shall be present only if ASPECT_RATIO_FLAG == 1, ASPECT_RATIO == '15', and DISPLAY_EXT == 1. ASPECT_VERT_SIZE specifies the vertical aspect size of the sample. This syntax element shall be a binary encoding of sizes ranging from 1 to 256. The sample aspect ratio is defined as the ratio of ASPECT_HORIZ_SIZE to ASPECT_VERT_SIZE.

6.1.14.4 Frame Rate Flag (FRAMERATE_FLAG)(1 bit)

The syntax element FRAMERATE_FLAG is a 1-bit syntax element that shall be present only if DISPLAY_EXT == 1. If FRAMERATE_FLAG == 1, the syntax element FRAMERATEIND shall be present. If FRAMERATE_FLAG == 1, frame rate information may be obtained from subsequent syntax elements. If FRAMERATE_FLAG == 0, the syntax element FRAMERATEIND shall not be present. In this case, the display process may rely on the underlying synchronization layer (such as the presentation time stamp in an MPEG-2 transport stream) to estimate the frame rate.

Note: If the video sequence is signaled as progressive (either implicitly as when the PROFILE syntax element takes the value corresponding to simple or main profile, or explicitly as when the PROFILE syntax element is set to advanced profile and the INTERLACE syntax element is set to zero), the period between two successive frames at the output of the decoding process is the reciprocal of the frame rate indicated by the FRAMERATE syntax element. If the video sequence is signaled as interlace, the period between two successive fields at the output of the decoding process is half the reciprocal of the frame rate indicated by the FRAMERATE syntax element.

6.1.14.4.1 Frame Rate Indicator (FRAMERATEIND)(1 bit)

The syntax element FRAMERATEIND is a 1-bit syntax element that shall be present only if FRAMERATE_FLAG == 1 and DISPLAY_EXT == 1. If FRAMERATEIND == 1, the frame rate shall be signaled explicitly by a 16 bit FRAMERATEEXP field. If FRAMERATEIND == 0, the frame rate shall be signaled by a numerator field (FRAMERATENR) and a denominator field (FRAMERATEDR), and the ratio of the two fields shall be taken to be the frame rate.

6.1.14.4.2 Frame Rate Numerator (FRAMERATENR)(8bits)

The syntax element FRAMERATENR is an 8-bit syntax element that shall be present only if FRAMERATEIND == 0 and FRAMERATE_FLAG == 1 and DISPLAY_EXT == 1. FRAMERATENR indicates the frame rate numerator of the encoded video sequence. The FRAMERATENR syntax element shall be as defined in Table 8.

Table 8: Meaning of FRAMERATENR syntax element

FRAMERATENR	Value of Frame Rate Numerator
0	Forbidden
1	24 * 1000
2	25 * 1000
3	30 * 1000
4	50 * 1000
5	60 * 1000
6	48 * 1000
7	72 * 1000
8-255	SMPTE Reserved

6.1.14.4.3 Frame Rate Denominator (FRAMERATEDR)(4 bits)

The syntax element FRAMERATEDR is a 4-bit syntax element that shall be present only if FRAMERATEIND == 0 and FRAMERATE_FLAG == 1 and DISPLAY_EXT == 1. FRAMERATEDR indicates the frame rate denominator of the encoded video sequence. The FRAMERATEDR syntax element shall be as defined in Table 9.

Table 9: Meaning of FRAMERATEDR syntax element

FRAMERATEDR	Value of Frame Rate Denominator
0	Forbidden
1	1000
2	1001
3-15	SMPTE Reserved

6.1.14.4.4 Frame Rate Explicit (FRAMERATEEXP)(16bits)

The syntax element FRAMERATEEXP is a 16-bit syntax element that shall be present only if FRAMERATEIND == 1 and FRAMERATE_FLAG == 1 and DISPLAY_EXT == 1. FRAMERATEEXP explicitly indicates the frame rate of the encoded video sequence. This syntax element shall be an encoding of frame rate ranging from 0.03125 Hz to 2048 Hz in uniform steps of 0.03125 Hz which shall be defined as follows:

frame rate = (FRAMERATEEXP + 1) / 32.0 Hz.

6.1.14.5 Color Format Indicator Flag (COLOR_FORMAT_FLAG)(1 bit)

COLOR_FORMAT_FLAG is a 1-bit syntax element that shall be present only if DISPLAY_EXT == 1. If COLOR_FORMAT_FLAG == 1, the syntax elements COLOR_PRIM, TRANSFER_CHAR and MATRIX_COEF shall be present. These syntax elements may be used to derive color format information, such as Color Primaries, Transfer Characteristics, and Matrix Coefficients. If COLOR_FORMAT_FLAG == 0, no color format information is present in the bitstream, and these syntax elements shall be set to the default values specified below.

6.1.14.5.1 Color Primaries (COLOR_PRIM)(8 bits)

COLOR_PRIM is an 8-bit syntax element that shall be present only if COLOR_FORMAT_FLAG == 1 and DISPLAY_EXT == 1. COLOR_PRIM describes the chromaticity coordinates of the color primaries. The COLOR_PRIM syntax element shall be as defined in Table 10 indicating the technical specifications where the chromaticity coordinates are specified. The default value is 1, ITU-R BT. 709-5.

Table 10: Meaning of COLOR_PRIM syntax element

COLOR_PRIM	Color Primaries Specification
0	Forbidden
1	ITU-R BT.709-5, SMPTE 274M-2005, ITU-R BT.1361 (Conventional Color Space), and SMPTE 296M-2001 primary x y green 0.300 0.600 blue 0.150 0.060 red 0.640 0.330 white (CIE D65) 0.3127 0.3290
2	Color primaries not present
3-4	SMPTE Reserved
5	ITU-R BT.1700 Part B -625 PAL. primary x y green 0.29 0.60 blue 0.15 0.06 red 0.64 0.33 white (CIE D65) 0.3127 0.3290
6	SMPTE C Primaries from ITU-R BT.1700 Part B – 525 PAL. Used in SMPTE 293M-2003. primary x y green 0.310 0.595 blue 0.155 0.070 red 0.630 0.340 white (CIE D65) 0.3127 0.3290
7-255	SMPTE Reserved

6.1.14.5.2 Transfer Characteristics (TRANSFER_CHAR)(8 bits)

TRANSFER_CHAR is an 8-bit syntax element that shall be present only if COLOR_FORMAT_FLAG == 1 and DISPLAY_EXT == 1. TRANSFER_CHAR describes the opto-electronic transfer characteristics of the source picture. The TRANSFER_CHAR syntax element shall be as defined in Table 11, indicating the technical specification where the transfer characteristics are specified. The default value is 1, ITU-R BT.709-5.

Table 11: Meaning of TRANSFER_CHAR syntax element

TRANSFER_CHAR	Transfer Characteristics Specification
0	Forbidden
1	ITU-R BT.709-5, SMPTE 274M-2005, SMPTE 296M-2001, SMPTE 293M-2003 $V = 1.099\ L_c^{0.45} - 0.099$ for $1 >= L_c >= 0.018$ $V = 4.500\ L_c$ for $0.018 > L_c >= 0$
2	Transfer Characteristics not present
3	SMPTE Reserved
4	ITU-R BT. 1700 Part A, Assumed display gamma 2.2
5	ITU-R BT. 1700 Part B and Part C. Assumed display gamma 2.8
6	ITU-R BT.1700 Part A. $V = 1.099\ L_c^{0.45} - 0.099$ for $1 >= L_c >= 0.018$ $V = 4.500\ L_c$ for $0.018 > L_c >= 0$
7	SMPTE Reserved
8	ITU-R BT.1361 (Conventional Color Space) $V = 1.099L_c^{0.45} - 0.099$ for $0.018 <= L_c < 1.33$ $V=4.50L_c$ for $-0.0045 <= L_c < 0.018$ $V=-\{1.099(-4L_c)^{0.45}-0.099\}/4$ for $-0.25 <= L_c < -0.0045$
9-255	SMPTE Reserved

6.1.14.5.3 Matrix Coefficients (MATRIX_COEF)(8 bits)

MATRIX_COEF is an 8-bit syntax element that shall be present only if COLOR_FORMAT_FLAG == 1 and DISPLAY_EXT == 1. MATRIX_COEF describes the matrix coefficients used to derive Y, C_b, C_r signals from the color primaries. The MATRIX_COEF syntax element shall be as defined in **Table 12** indicating the technical specification where the matrices are specified. The default value is 6, ITU-R BT. 601-5.

MATRIX_COEF	Matrix Coefficients Specification
0	Forbidden
1	ITU-R BT.709-5 (1125/60/2:1 only), SMPTE 274M-2005 and SMPTE 296M-2001, ITU-R BT.1361 (Conventional Color Space). $K_R = 0.2126$; $K_B = 0.0722$
2	Matrix not present
3-5	SMPTE Reserved
6	ITU-R BT.1700, ITU-R BT.601-5, and SMPTE 293M-2003. $K_R = 0.299$; $K_B = 0.114$
7-255	SMPTE Reserved

Table 12: Meaning of MATRIX_COEF syntax element

The interpretation of matrix coefficients shall be defined as follows:

$E'_Y = K_R * E'_R + (1 - K_R - K_B) * E'_G + K_B * E'_B$

$E'_{PB} = 0.5 * (E'_B - E'_Y) \div (1 - K_B)$

$E'_{PR} = 0.5 * (E'_R - E'_Y) \div (1 - K_R)$

– E'_Y is analogue with values between 0 and 1;

– E'_{PB} and E'_{PR} are analog between the values – 0.5 and 0.5;

– E'_R, E'_G and E'_B are analog with values between 0 and 1;

– Y, C_b, C_r are related to E'_Y, E'_{PB} and E'_{PR} by the following formulae: $Y = (219 * E'_Y) + 16$, $C_b = (224 * E'_{PB}) + 128$, $C_r = (224 * E'_{PR}) + 128$.

Note: The decoding process given by this specification limits output sample values for Y, C_b, C_r to the range [0:255]. Thus, sample values outside the range implied by the above equations may occasionally occur at the output of the decoding process. In particular the sample values 0 and 255 may occur.

6.1.15 Hypothetical Reference Decoder Indicator Flag (HRD_PARAM_FLAG)(1 bit)

The HRD_PARAM_FLAG is a 1-bit flag that indicates the presence of HRD_PARAM parameters in the bitstream. If this HRD_PARAM_FLAG == 0, HRD parameters shall not be present. If HRD_PARAM_FLAG == 1, syntax elements of the HRD shall be present as detailed in 6.1.15.1.

6.1.15.1 Hypothetical Reference Decoder (HRD_PARAM)(Variable size)

The HRD_PARAM structure shall be present only if HRD_PARAM_FLAG == 1. The syntax elements that make up the HRD_PARAM structure shall be as defined below.

See Annex C for additional details on the semantics and use of these syntax elements.

HRD_PARAM( )	Number of Bits	Descriptor
{		
HRD_NUM_LEAKY_BUCKETS	5	uimsbf
BIT_RATE_EXPONENT	4	uimsbf
BUFFER_SIZE_EXPONENT	4	uimsbf
for(n=1; n <= HRD_NUM_LEAKY_BUCKETS; n++)		
{		
HRD_RATE[n **]**	16	uimsbf
HRD_BUFFER[n **]**	16	uimsbf
}		

Table 13: Syntax elements for HRD_PARAM structure

HRD_NUM_LEAKY_BUCKETS – Shall be a number between 0 and 31 that specifies the number of leaky buckets *N*. The value of *N* shall be encoded as a fixed length code in binary using 5 bits, i.e., the number of leaky buckets *N* = (**HRD_NUM_LEAKY_BUCKETS**)

HRD_RATE[n] and **BIT_RATE_EXPONENT** – These syntax elements define the peak transmission rate R_n in bits per second for the *n*th leaky bucket. The mantissa of R_n shall be encoded in the syntax element HRD_RATE**[**n**]** using a fixed-length code of 16 bits, and has the range from 1 to 2^{16}. The base-2 exponent of R_n shall be encoded in the syntax element BIT_RATE_EXPONENT in fixed length using 4 bits, and shall take the range from 6 to 21.

Thus, R_n = (**HRD_RATE[n]** + 1) * 2 $^{(\mathbf{BIT\_RATE\_EXPONENT} + 6)}$.

The rates shall be ordered from smallest to largest, i.e., HRD_RATE**[**n**]** < HRD_RATE**[**n+1**]**.

HRD_BUFFER[n] and **BUFFER_SIZE_EXPONENT** – These syntax elements define the buffer size B_n in bits for the nth leaky bucket. The mantissa of B_n shall be encoded in the syntax element HRD_BUFFER**[**n**]**, using a fixed length code of 16 bits, and has the range 1 to 2^{16}. The value of the base-2 exponent of B_n shall be encoded in the syntax element BUFFER_SIZE_EXPONENT using a fixed length of 4 bits, and shall take the range from 4 to 19.

Thus, B_n = (**HRD_BUFFER[n]** + 1) * 2 $^{(\mathbf{BUFFER\_SIZE\_EXPONENT} + 4)}$.

The buffer sizes shall be ordered from largest to smallest, i.e., HRD_BUFFER**[**n**]** >= HRD_BUFFER**[**n+1**]**.

6.2 Entry-Point Header Syntax and Semantics

An entry-point header shall be present only in advanced profile. The entry point has two purposes.

First, it shall be used to signal a random access point within the bitstream. See 4.12 for the definition of random access. An entry point guarantees that subsequent pictures can be decoded starting from the entry point.

Second, it shall be used to signal changes in the coding control parameters. An entry-point header contains syntax elements specifying the buffer fullness of the HRD leaky bucket and it contains coding control parameters that are used to signal which compression tools are enabled for the entry point segment.

The syntax elements that make up the entry-point layer shall be as defined in Figure 11, and Table 14. The use of the entry-point header is defined in Annex G.

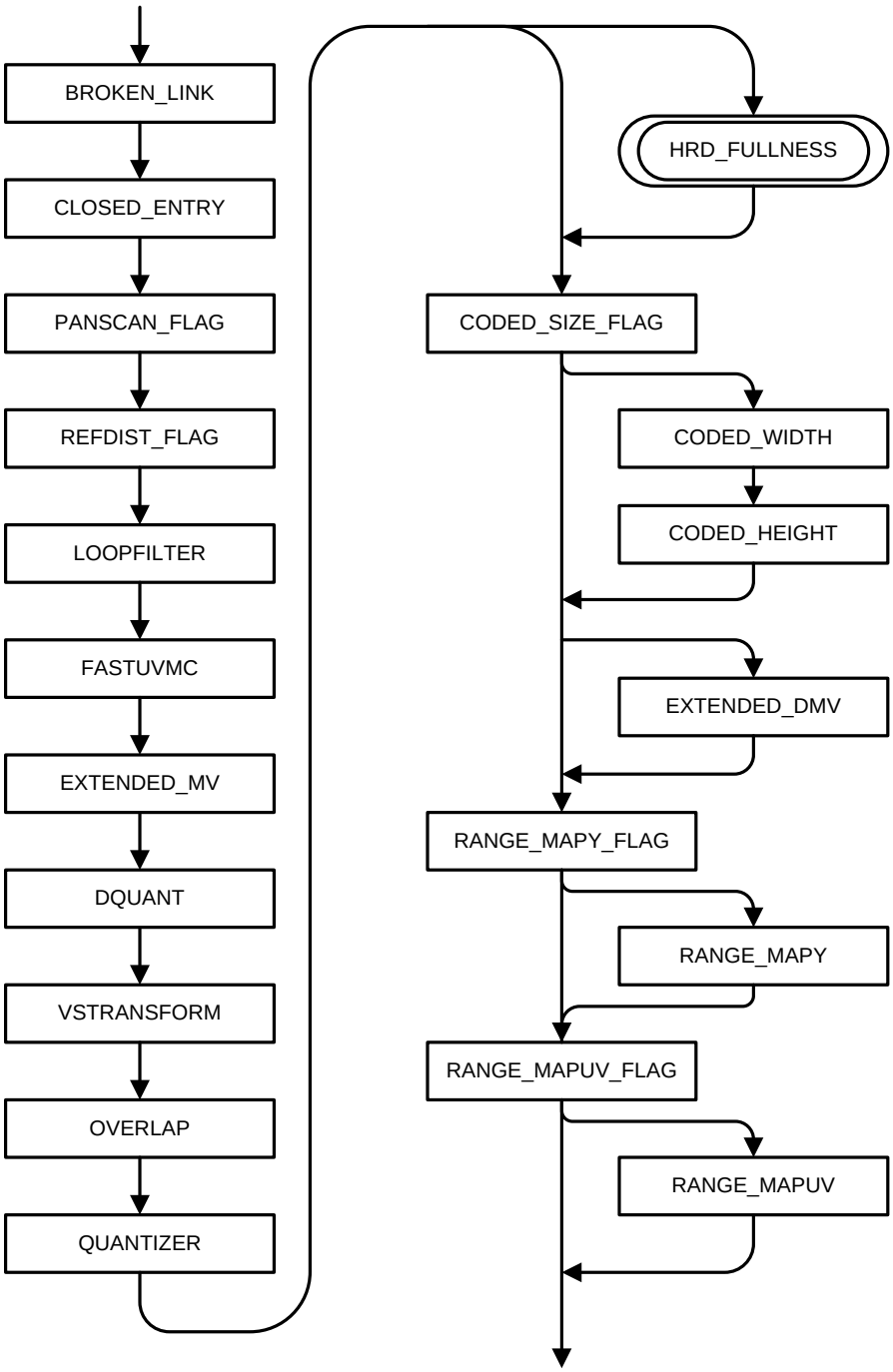


Figure 11: Syntax diagram for the entry-point layer bitstream for the Advanced Profile

Table 14: Entry-point layer bitstream for Advanced Profile

ENTRYPOINT LAYER( ) {	Number of bits	Descriptor	Reference
BROKEN_LINK	1	uimsbf	6.2.1
CLOSED_ENTRY	1	uimsbf	6.2.2
PANSCAN_FLAG	1	uimsbf	6.2.3
REFDIST_FLAG	1	uimsbf	6.2.4
LOOPFILTER	1	uimsbf	6.2.5
FASTUVMC	1	uimsbf	6.2.6
EXTENDED_MV	1	uimsbf	6.2.7
DQUANT	2	uimsbf	6.2.8
VSTRANSFORM	1	uimsbf	6.2.9
OVERLAP	1	uimsbf	6.2.10
QUANTIZER	2	uimsbf	6.2.11
if (HRD_PARAM_ FLAG == 1) {			6.1.15
HRD_FULLNESS ( )			6.2.12
}			
CODED_SIZE_FLAG	1	uimsbf	6.2.13
if (CODED_SIZE_FLAG == 1) {			
CODED_WIDTH	12	uimsbf	6.2.13.1
CODED_HEIGHT	12	uimsbf	6.2.13.2
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
EXTENDED_DMV	1	uimsbf	6.2.14
}			
RANGE_MAPY_FLAG	1	uimsbf	6.2.15
if (RANGE_MAPY_FLAG == 1) {			
RANGE_MAPY	3	uimsbf	6.2.15.1
}			
RANGE_MAPUV_FLAG	1	uimsbf	6.2.16
if (RANGE_MAPUV_FLAG == 1) {			
RANGE_MAPUV	3	uimsbf	6.2.16.1
}			
}			

6.2.1 Broken Link Flag (BROKEN_LINK) (1 bit)

BROKEN_LINK is a 1-bit syntax element. When CLOSED_ENTRY == 1, then BROKEN_LINK shall be equal to zero. When CLOSED_ENTRY == 0, BROKEN_LINK indicates if the previous entry-point segment does not contain the appropriate anchor frames that can be used to decode any dependent B frames in the current entry-point segment.

The combination of CLOSED_ENTRY == 0 and BROKEN_LINK == 1 shall indicate that the current entry point contains B pictures which require an I or P anchor picture in the previous entry point to decode, and the previous entry point segment no longer contains these anchor pictures (usually because of an edit). The combination of CLOSED_ENTRY == 0 and BROKEN_LINK == 0 shall indicate that the previous entry point segment contains the appropriate anchor frames that are required to decode any dependent B pictures.

When CLOSED_ENTRY == 0, and if the B pictures that follow an entry-point lack a reference anchor picture, these B pictures shall be discarded.

If the first frame after an entry-point header is coded as a PI Field picture, the P Field is dependent on pictures in the previous entry-point segment. In this case, the combination of CLOSED_ENTRY == 0 and BROKEN_LINK == 0 shall indicate that the previous entry point segment contains the appropriate reference frames to decode this dependent P Field in the PI Field picture that follows the entry-point. If CLOSED_ENTRY == 0, and if this P Field lacks the reference picture, the P Field in the PI Field picture that follows the entry-point shall be discarded.

6.2.2 Closed Entry Point (CLOSED_ENTRY) (1 bit)

CLOSED_ENTRY is a 1-bit syntax element. CLOSED_ENTRY == 1 shall indicate that the current entry point segment does not contain any B pictures that require reference to an I or P picture in the previous entry point segment to decode. Further, if CLOSED_ENTRY == 1, the first frame after the entry-point header shall not be a PI Field picture. CLOSED_ENTRY == 0 shall indicate that the entry point segment may contain B pictures that require reference to an I or P picture in the previous entry point segment to decode.

6.2.3 Pan Scan Present Flag (PANSCAN_FLAG) (1 bit)

PANSCAN_FLAG is a 1-bit syntax element. PANSCAN == 1 shall indicate that pan/scan information is present in the picture headers within the entry point segment. PANSCAN == 0 shall indicate that no pan/scan information is present in the picture headers within the entry point segment.

6.2.4 Reference Frame Distance Flag (REFDIST_FLAG) (1 bit)

REFDIST_FLAG is a 1-bit syntax element. REFDIST_FLAG == 1 shall indicate that the REFDIST syntax element is present in II, IP, PI or PP field picture headers. REFDIST_FLAG == 0 shall indicate that the REFDIST syntax element is not present in II, IP, PI or PP field picture headers.

6.2.5 Loop Filter Flag (LOOPFILTER) (1 bit)

As defined in Annex J.1.9.

6.2.6 Fast UV Motion Compensation Flag (FASTUVMC) (1 bit)

As defined in Annex J.1.11.

6.2.7 Extended Motion Vector Flag (EXTENDED_MV)(1 bit)

As defined in Annex J.1.12.

6.2.8 Macroblock Quantization Flag (DQUANT)(2 bit)

As defined in Annex J.1.13.

6.2.9 Variable Sized Transform Flag (VSTRANSFORM)(1 bit)

As defined in Annex J.1.14.

6.2.10 Overlapped Transform Flag (OVERLAP) (1 bit)

As defined in Annex J.1.15.

6.2.11 Quantizer Specifier (QUANTIZER) (2 bits)

As defined in Annex J.1.19.

6.2.12 HRD Buffer Fullness (HRD_FULLNESS)(Variable Size)

HRD_FULLNESS is a variable size structure that shall be present only if the HRD_PARAM_FLAG in the sequence header is set to 1. If the HRD_PARAM_FLAG in the sequence header is set to zero, HRD_FULLNESS structure shall not be present. The structure shall be as defined in Table 15. See below and Annex C.2 for additional details on the semantics and use of the HRD parameters.

Table 15: Syntax Elements for HRD_FULLNESS structure

HRD_FULLNESS( )	Number of bits	Descriptor
{		
for(n=1; n <= HRD_NUM_LEAKY_BUCKETS; n++)		
{		
HRD_FULL[n **]**	8	uimsbf
}		

HRD_FULL**[**n**]** – This syntax element defines the decoder buffer fullness. The decoder buffer fullness for the *n*th leaky bucket shall be equal to (HRD_FULL[n] + 1) * B_n/256, where B_n is the buffer size for the *n*th leaky bucket.

6.2.13 Coded Size Flag (CODED_SIZE_FLAG) (1 bit)

CODED_SIZE_FLAG is a 1-bit syntax element. CODED_SIZE_FLAG == 1 shall indicate that the CODED_WIDTH and CODED_HEIGHT syntax elements are present in the entry header. CODED_SIZE_FLAG == 0 shall indicate that the CODED_WIDTH and CODED_HEIGHT syntax elements are not present in the entry header and the width and height of the frames within the entry point segment shall be specified by the MAX_CODED_WIDTH and MAX_CODED_HEIGHT syntax elements in the sequence header. The calculation of width and height of the frames is shown in Figure 12.

If either the frame width (specified by either the CODED_WIDTH or MAX_CODED_WIDTH as described above) or the frame height (as specified by the CODED_HEIGHT or the MAX_CODED_HEIGHT) of an entry point segment are different from the frame width or the frame height of the previous entry point segment, then either CLOSED_ENTRY shall be equal to 1, or BROKEN_LINK shall be equal to 1.

Note: This constraint prevents prediction from a different resolution picture, when the resolution changes at an entry point.

6.2.13.1 Coded Frame Width (CODED_WIDTH) (12 bits)

CODED_WIDTH is a 12 bit unsigned integer that shall be present if CODED_SIZE_FLAG == 1. The coded width of the frames within the entry point segment shall be equal to the value of this field multiplied by 2, plus 2. Therefore the range is 2 to 8192. CODED_WIDTH shall be less than or equal to MAX_CODED_WIDTH.

6.2.13.2 Coded Frame Height (CODED_HEIGHT) (12 bits)

CODED_HEIGHT is a 12 bit unsigned integer that shall be present if CODED_SIZE_FLAG == 1. The coded height of the frames within the entry point segment shall be equal to the value of this field multiplied by2, plus 2. Therefore the range is 2 to 8192. CODED_HEIGHT shall be less than or equal to MAX_CODED_HEIGHT.

Note: The coded height of the frames is also subject to the constraints specified in 5.5.1.

```
If (CODED_SIZE_FLAG) {
      CodedWidth = CODED_WIDTH;
      CodedHeight = CODED_HEIGHT;
}
else {
       CodedWidth = MAX_CODED_WIDTH;
       CodedHeight = MAX_CODED_HEIGHT;
}
```

Figure 12: Calculation of Frame Width and Height

6.2.14 Extended Differential Motion Vector Range Flag (EXTENDED_DMV)(1 bit)

EXTENDED_DMV is a 1-bit syntax element that shall be present if EXTENDED_MV == 1. If this bit is 1, extended differential motion vector range shall be signaled at the picture layer for the P and B pictures within the entry point segment. If this bit is 0, extended differential motion vector range shall not be signaled.

6.2.15 Range Mapping Luma Flag (RANGE_MAPY_FLAG)(1 bit)

RANGE_MAPY_FLAG is a 1-bit syntax element. If RANGE_MAPY_FLAG == 1, the syntax element RANGE_MAPY shall be present within the entry header. If RANGE_MAPY_FLAG == 0 the syntax element RANGE_MAPY shall not be present.

6.2.15.1 Range Mapping Luma (RANGE_MAPY)(3 bits)

RANGE_MAPY is a 3-bit syntax element that shall be present if RANGE_MAPY_FLAG == 1. RANGE_MAPY takes the value from 0 to 7. If this syntax element is present, the luma components of the decoded pictures within the entry point segment shall be scaled according to the formula:

$$Y[n] = \text{clip}((((Y[n] - 128) * (\text{RANGE\_MAPY} + 9) + 4) >> 3) + 128);$$

where Y[n] represents the pixel values of the luma component.

This scaling shall be performed after all other decoding stages (including loop-filter) have been performed.

If CLOSED_ENTRY == 0, the values of RANGE_MAPY_FLAG and RANGE_MAPY shall be set to the same values as those of the corresponding syntax elements in the previous entry-point segment.

6.2.16 Range Mapping Color-Difference Flag (RANGE_MAPUV_FLAG)(1 bit)

RANGE_MAPUV_FLAG is a 1-bit syntax element. If RANGE_MAPUV_FLAG == 1, the syntax element RANGE_MAPUV shall be present within the entry header. If RANGE_MAPUV_FLAG == 0, the syntax element RANGE_MAPUV shall not be present.

6.2.16.1 Range Mapping Color-Difference (RANGE_MAPUV)(3 bits)

RANGE_MAPUV is a 3-bit syntax element that shall be present if RANGE_MAPUV_FLAG == 1. RANGE_MAPUV takes the value from 0 to 7. If this syntax element is present, the color-difference components of the decoded pictures within the entry point segment shall be scaled according to the formula:

$$C_b[n] = \text{clip}((((C_b[n] - 128) * (\text{RANGE\_MAPUV} + 9) + 4) >> 3) + 128);$$

$$C_r[n] = \text{clip}((((C_r[n] - 128) * (\text{RANGE\_MAPUV} + 9) + 4) >> 3) + 128);$$

This scaling shall be performed after all other decoding stages (including loop-filter) have been performed.

If CLOSED_ENTRY == 0, the values of RANGE_MAPUV_FLAG and RANGE_MAPUV shall be set to the same values as those of the corresponding syntax elements in the previous entry-point segment.

7 Progressive Bitstream Syntax and Semantics

7.1 Picture-level Syntax and Semantics

This section defines the syntax and semantics of the picture layer, slice layer, macroblock layer, and block layer of the compressed stream when the picture is coded in progressive mode. The slice layer shall be present only in advanced profile bitstreams.

In the advanced profile, pictures and slices shall be byte-aligned and carried in a BDU (as defined in Annex E). Each new picture or a slice is detected via start-codes as defined in Annex E.

In the simple and main profiles, pictures shall be byte-aligned. For each coded picture, the pointer to the coded bitstream and its size shall be communicated to the decoder by the Transport Layer. In simple and main profiles, a picture whose coded size is less than or equal to one byte shall be considered to be a skipped picture.

Figure 13 through Figure 28 show the bitstream elements that make up each layer. Table 16 through Table 32 shall define the syntax elements of the picture-layer, slice-layer, macroblock-layer, block-layer bitstream.

Unless otherwise stated, the length of variable length fields shall be the number of bits in the defined binary value.

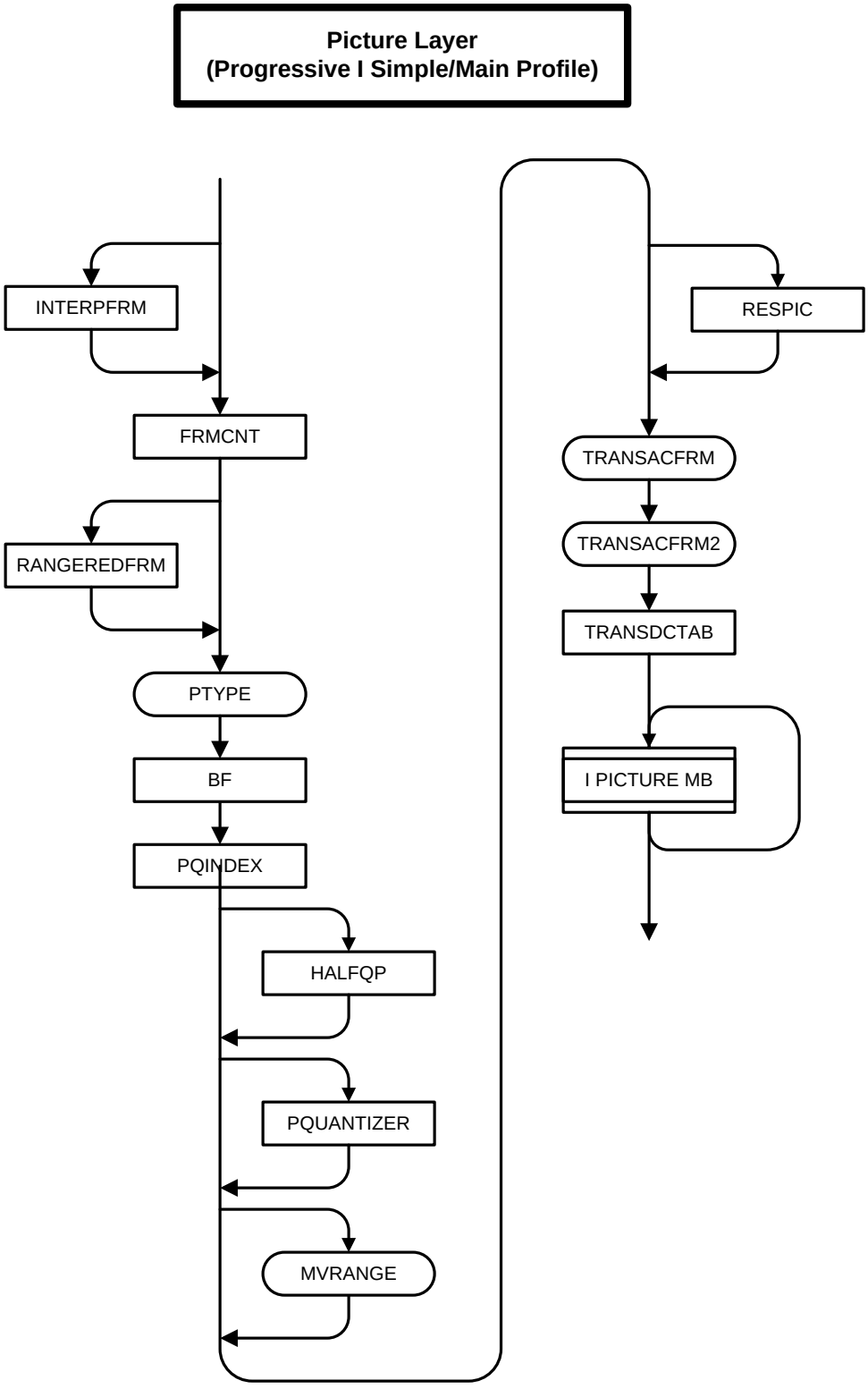


Figure 13: Syntax diagram for the Progressive I picture layer bitstream in simple/main profile

Table 16: Progressive I picture layer bitstream for Simple and Main Profile

I SIMPLE/MAIN PICTURE ( ) {	Number of bits	Descriptor	Reference
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
FRMCNT	2	uimsbf	7.1.1.2
if (RANGERED == 1) {			Annex J.1.17
RANGEREDFRM	1	uimsbf	7.1.1.3
}			
PTYPE	Var. size or 1	vlclbf	7.1.1.4 In this table, PTYPE is 1b if MAXBFRAMES == 0; PTYPE is 01b otherwise.
BF	7	uimsbf	7.1.1.5
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9 Only M.P.
}			
if (MULTIRES == 1) {			Annex J.1.10
RESPIC	2	uimsbf	7.1.1.10
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSACFRM2	Variable size	vlclbf	7.1.1.12
TRANSDCTAB	1	uimsbf	7.1.1.13
for ('all macroblocks') {			'all macroblocks' represents all macroblocks in the frame. Sync markers, if

			present, shall be handled as defined in 8.8
I AND BI SIMPLE/MAIN MB( )			Table 27
}			
}			

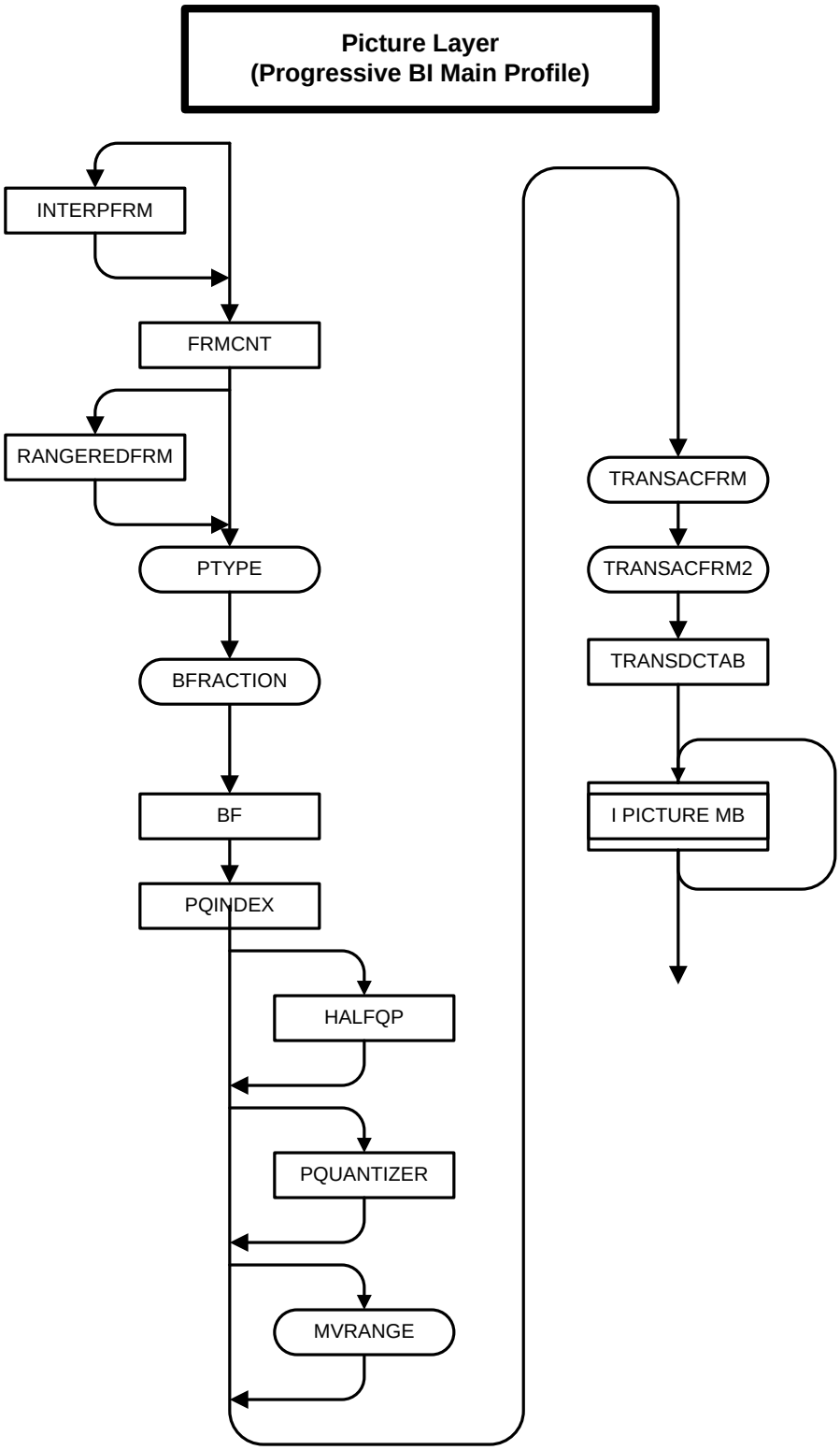


Figure 14: Syntax diagram for the Progressive BI picture layer bitstream in main profile

Table 17: Progressive BI picture layer bitstream for Main Profile

BI MAIN PICTURE ( ) {	Number of bits	Descriptor	Reference
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
FRMCNT	2	uimsbf	7.1.1.2
if (RANGERED == 1) {			Annex J.1.17
RANGEREDFRM	1	uimsbf	7.1.1.3
}			
PTYPE	Var. size or 1	vlclbf	7.1.1.4 In this table, PTYPE is 00b
BFRACTION	Var. size	vlclbf	7.1.1.14 In this table, BFRACTION is 1111111b
BF	7	uimsbf	7.1.1.5
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSACFRM2	Variable size	vlclbf	7.1.1.12
TRANSDCTAB	1	uimsbf	7.1.1.13
for (‘all macroblocks’) {			/‘all macroblocks’ represents all macroblocks in the frame. Sync markers, if present, shall be handled as defined in 8.8

I AND BI SIMPLE/MAIN MB ( )			Table 27
}			
}			

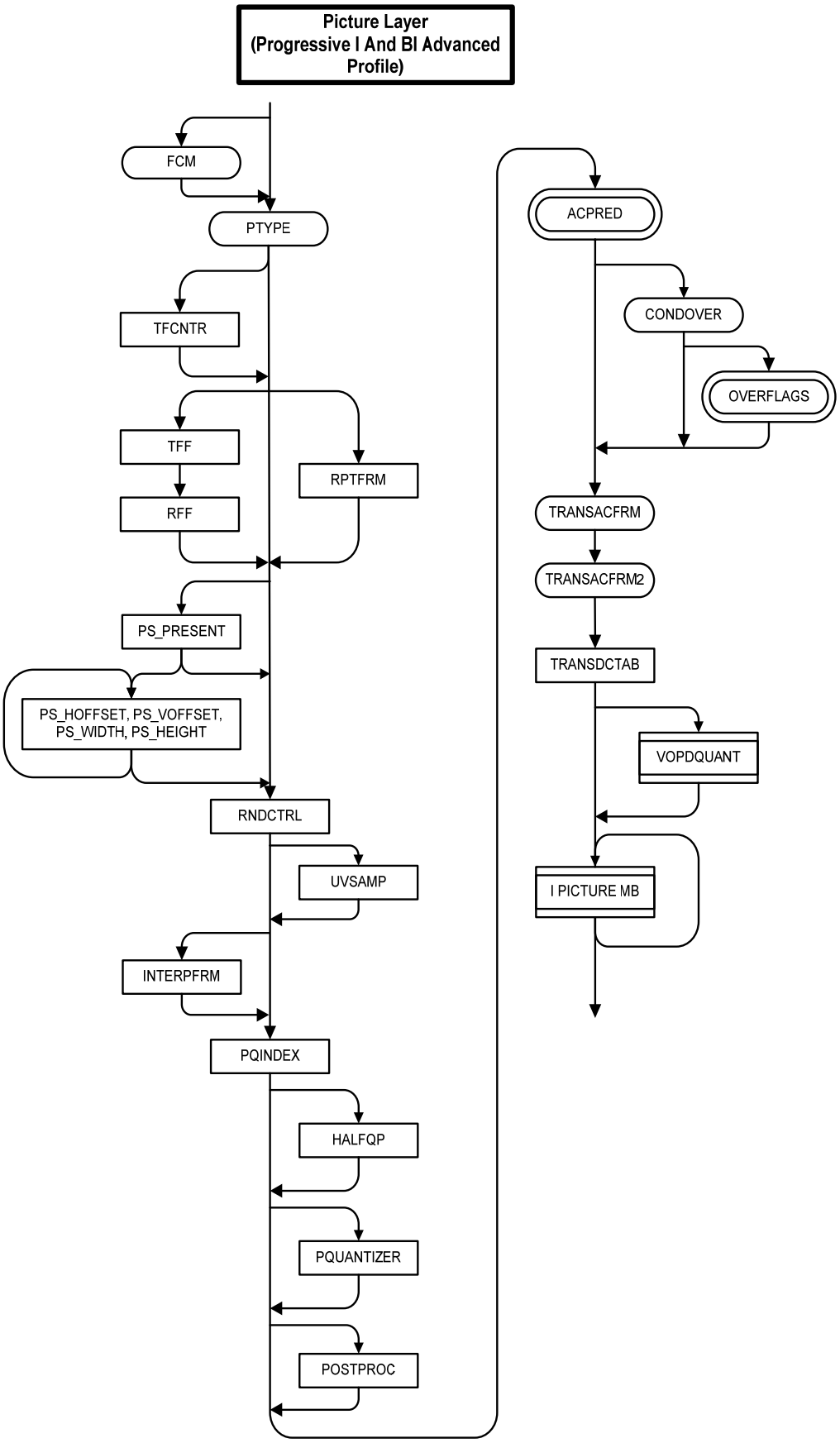


Figure 15: Syntax diagram for the Progressive I and BI picture layer bitstream in advanced profile.

Table 18: Progressive I and BI picture layer bitstream for Advanced Profile

I AND BI ADV PICTURE ( ) {	Number of bits	Descriptor	Reference
If (INTERLACE == 1)			6.1.9
FCM	Variable size	vlclbf	7.1.1.15 // FCM==0 in this table.
PTYPE	Variable size	vlclbf	7.1.1.4 In this table, PTYPE is 110b or PTYPE is 1110b
if (TFCNTRFLAG) {			6.1.10
TFCNTR	8	uimsbf	7.1.1.16
}			
if (PULLDOWN) {			6.1.8
if (INTERLACE == 0 || PSF == 1) {			6.1.9, 6.1.13
RPTFRM	2	uimsbf	7.1.1.19
}			
else {			
TFF	1	uimsbf	7.1.1.17
RFF	1	uimsbf	7.1.1.18
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	7.1.1.20
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	7.1.1.21
PS_VOFFSET	18	uimsbf	7.1.1.22
PS_WIDTH	14	uimsbf	7.1.1.23
PS_HEIGHT	14	uimsbf	7.1.1.24
}			
}			

}			
RNDCTRL	1	uimsbf	7.1.1.25
if (INTERLACE == 1)			6.1.9
UVSAMP	1	uimsbf	7.1.1.26
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	7.1.1.27
}			
ACPRED	Bitplane		7.1.1.28
if (OVERLAP == 1 && 'PQUANT <= 8') {			OVERLAP 6.2.10 PQUANT (7.1.1.6) computed from PQINDEX as shown in 7.1.1.6
CONDOVER	Variable size	vlclbf	7.1.1.29
if (CONDOVER == 11b) {			
OVERFLAGS	Bitplane		7.1.1.30
}			
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSACFRM2	Variable size	vlclbf	7.1.1.12
TRANSDCTAB	1	uimsbf	7.1.1.13
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 7.1.1.31
}			
for ('all macroblocks') {			'all macroblocks' represents all macroblocks in this BDU.

I AND BI ADV MB ( )			Table 28
}			
}			

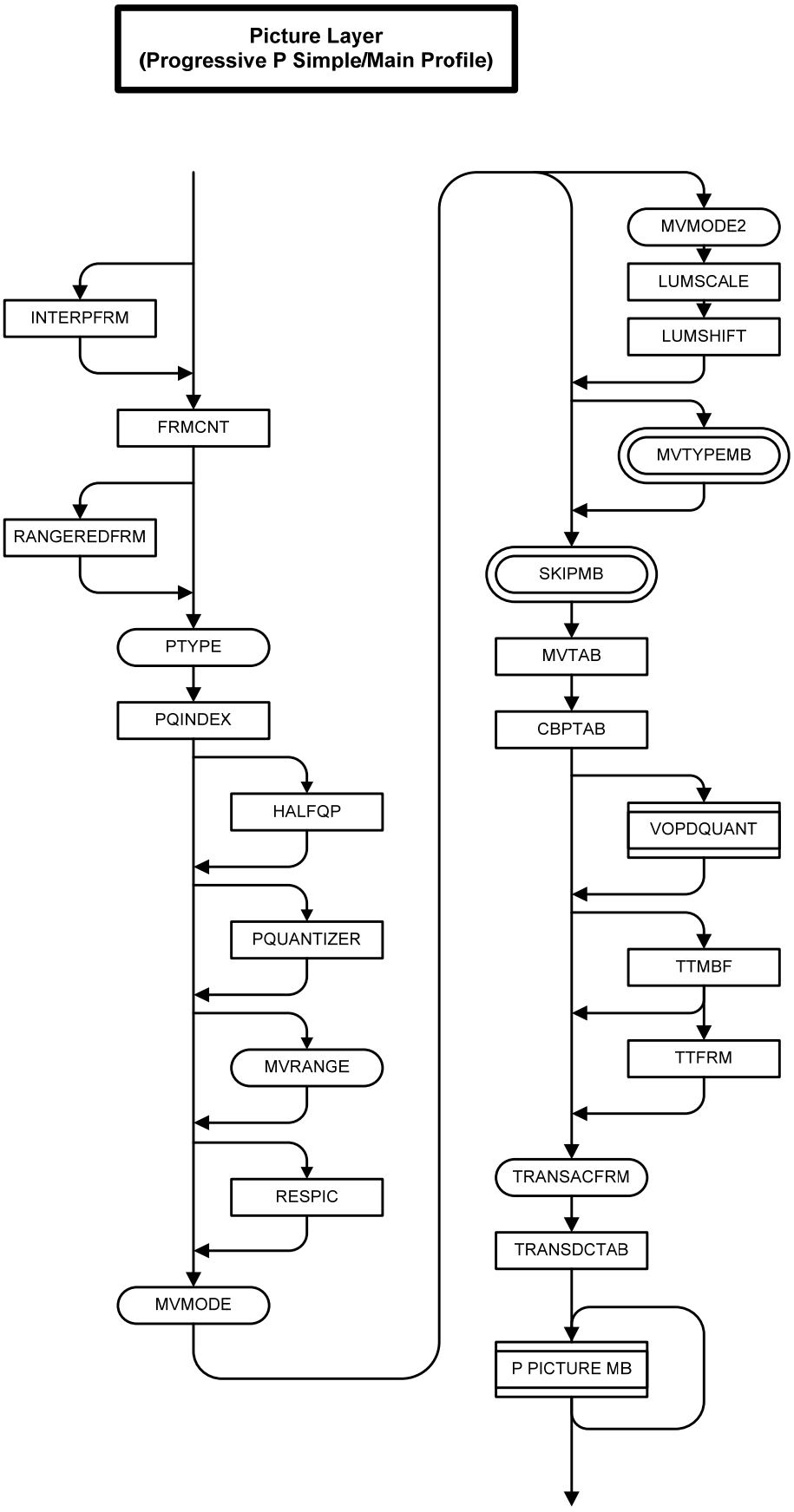


Figure 16: Syntax diagram for the Progressive P picture layer bitstream in Simple/Main Profile.

Table 19: Progressive P picture layer bitstream for Simple and Main Profile

P SIMPLE/MAIN PICTURE ( ) {	Number of bits	Descriptor	Reference
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
FRMCNT	2	uimsbf	7.1.1.2
if (RANGERED == 1) {			Annex J.1.17
RANGEREDFRM	1	uimsbf	7.1.1.3
}			
PTYPE	Var. size or 1	vlclbf	7.1.1.4 In this table, PTYPE is 1
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9 Not S.P.
}			
if (MULTIRES == 1) {			Annex J.1.10
RESPIC	2	uimsbf	7.1.1.10
}			
MVMODE	Variable size	vlclbf	7.1.1.32
if (MVMODE == ‘intensity compensation’) {			
MVMODE2	Variable size	vlclbf	7.1.1.33
LUMSCALE	6	uimsbf	7.1.1.34
LUMSHIFT	6	uimsbf	7.1.1.35
}			

if (MVMODE == 'Mixed-MV' || (MVMODE == 'Intensity Compensation' && MVMODE2 == 'Mixed-MV') {			
MVTYPEMB	Bitplane		7.1.1.36
}			
SKIPMB	Bitplane		7.1.1.37
MVTAB	2	uimsbf	7.1.1.38
CBPTAB	2	uimsbf	7.1.1.39
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 7.1.1.31
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	7.1.1.40
if (TTMBF == 1) {			
TTFRM	2	uimsbf	7.1.1.41
}			
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSDCTAB	1	uimsbf	7.1.1.13
for ('all macroblocks') {			'all macroblocks' represents all macroblocks in the frame. Sync markers, if present, shall be handled as defined in 8.8
P SIMPLE/MAIN/ADV MB ( )			Table 29
}			
}			

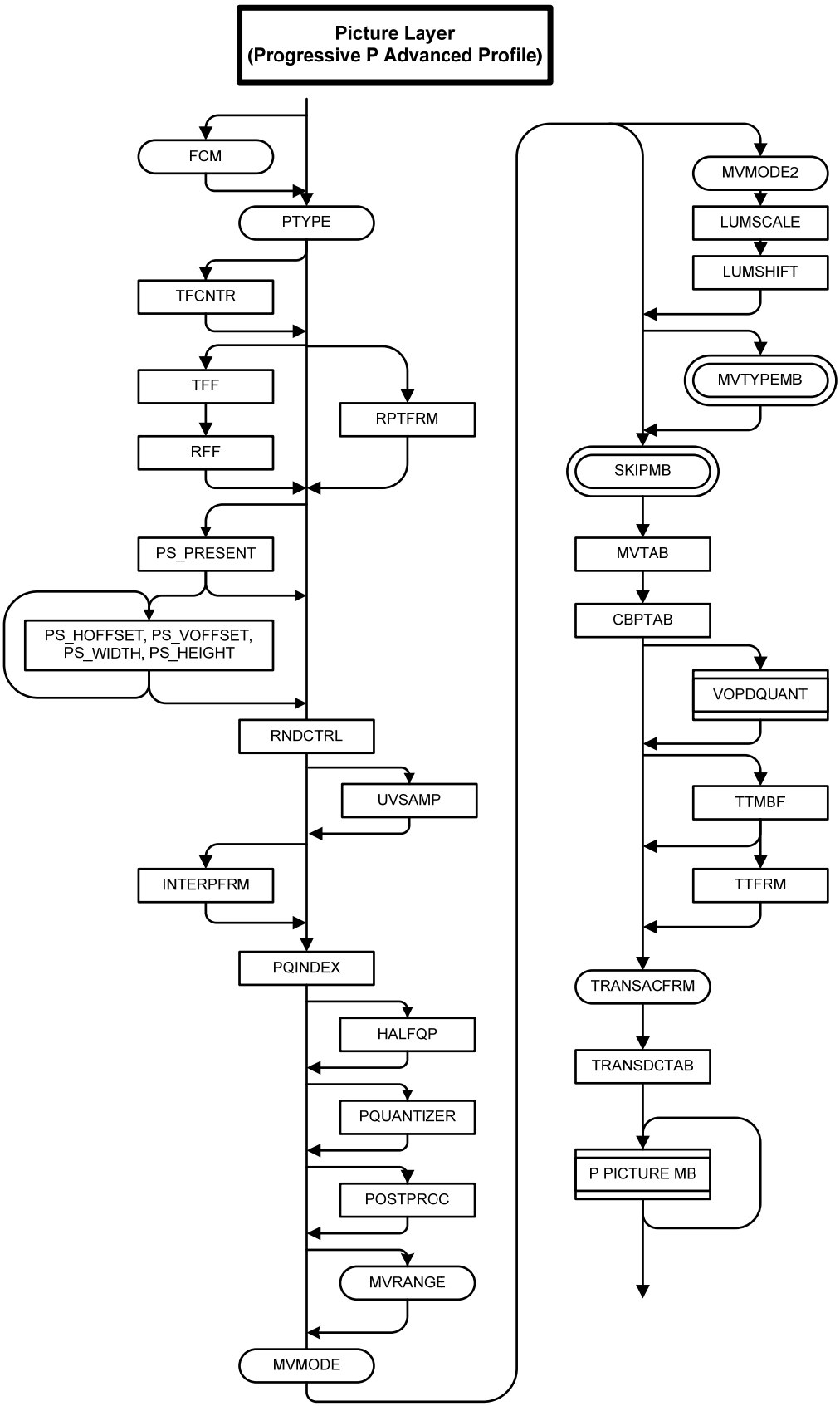


Figure 17: Syntax diagram for the Progressive P picture layer bitstream in Advanced Profile

Table 20: Progressive P picture layer bitstream for Advanced Profile

P ADV PICTURE ( ) {	Number of bits	Descriptor	Reference
If (INTERLACE == 1)			6.1.9
FCM	Variable size	vlclbf	7.1.1.15 // FCM==0 in this table.
PTYPE	Variable size	vlclbf	7.1.1.4 In this table, PTYPE is 0
if (TFCNTRFLAG) {			6.1.10
TFCNTR	8	uimsbf	7.1.1.16
}			
if (PULLDOWN) {			6.1.8
if (INTERLACE == 0 || PSF == 1) {			6.1.9, 6.1.13
RPTFRM	2	uimsbf	7.1.1.19
}			
else {			
TFF	1	uimsbf	7.1.1.17
RFF	1	uimsbf	7.1.1.18
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	7.1.1.20
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	7.1.1.21
PS_VOFFSET	18	uimsbf	7.1.1.22
PS_WIDTH	14	uimsbf	7.1.1.23
PS_HEIGHT	14	uimsbf	7.1.1.24
}			
}			
}			

RNDCTRL	1	uimsbf	7.1.1.25
if (INTERLACE == 1)			6.1.9
UVSAMP	1	uimsbf	7.1.1.26
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	7.1.1.27
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9
}			
MVMODE	Variable size	vlclbf	7.1.1.32
if (MVMODE == 'intensity compensation') {			
MVMODE2	Variable size	vlclbf	7.1.1.33
LUMSCALE	6	uimsbf	7.1.1.34
LUMSHIFT	6	uimsbf	7.1.1.35
}			
if (MVMODE == 'Mixed-MV' || (MVMODE == 'Intensity Compensation' && MVMODE2 == 'Mixed-MV') {			
MVTYPEMB	Bitplane		7.1.1.36
}			
SKIPMB	Bitplane		7.1.1.37
MVTAB	2	uimsbf	7.1.1.38
CBPTAB	2	uimsbf	7.1.1.39

if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 7.1.1.31
}			
if (VSTRANSFORM == 1) {			6.2.9
TTMBF	1	uimsbf	7.1.1.40
if (TTMBF == 1) {			
TTFRM	2	uimsbf	7.1.1.41
}			
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSDCTAB	1	uimsbf	7.1.1.13
for ('all macroblocks') {			'all macroblocks' represents all macroblocks in this BDU.
P SIMPLE/MAIN/ADV MB( )			Table 29
}			
}			

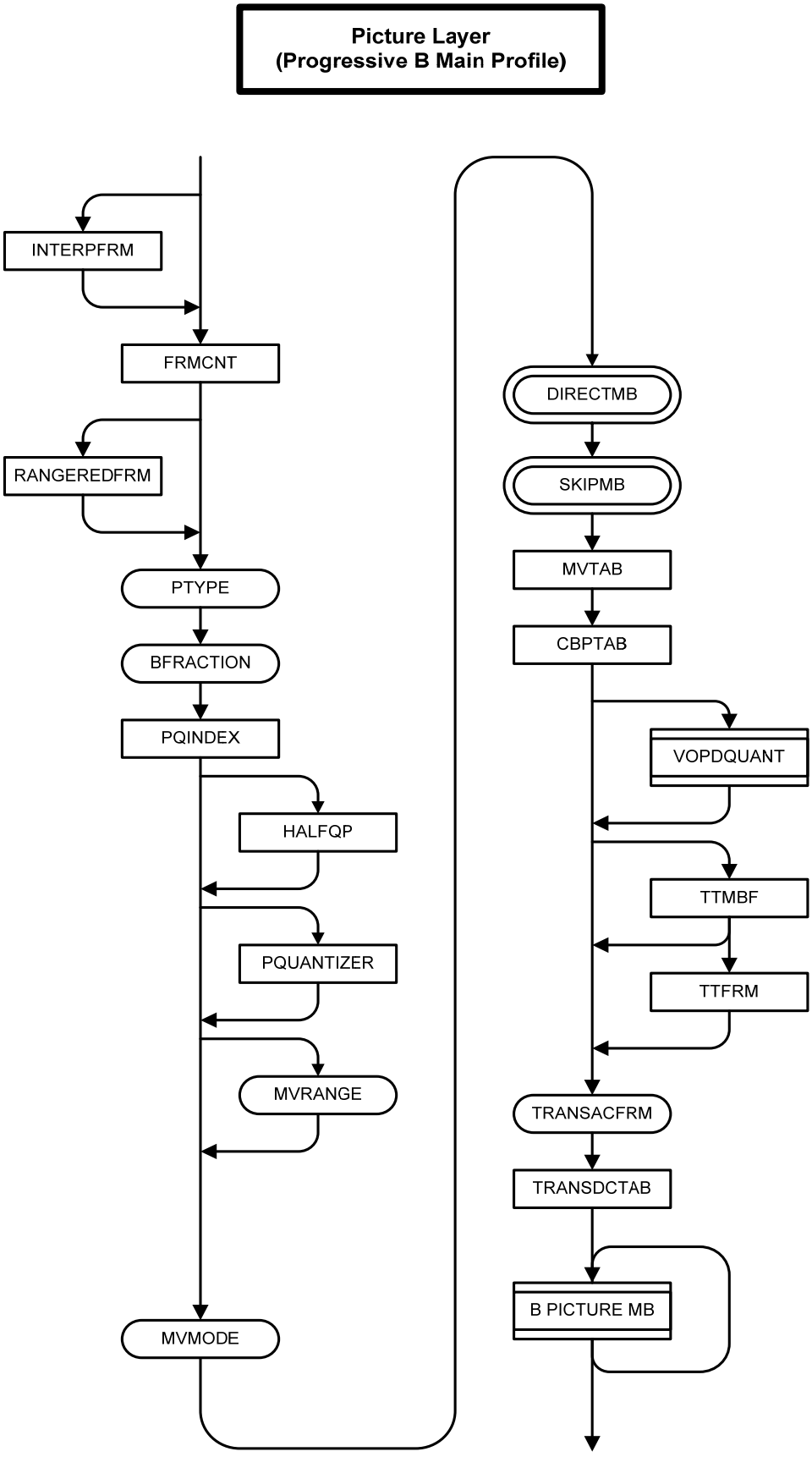


Figure 18: Syntax diagram for the Progressive B picture layer bitstream in Main Profile.

Table 21: Progressive B picture layer bitstream for Main Profile

B MAIN PICTURE ( ) {	Number of bits	Descriptor	Reference
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
FRMCNT	2	uimsbf	7.1.1.2
if (RANGERED == 1) {			Annex J.1.17
RANGEREDFRM	1	uimsbf	7.1.1.3
}			
PTYPE	Var. size or 1	vlclbf	7.1.1.4 In this table, PTYPE is 00b
BFRACTION	Variable size	vlclbf	7.1.1.14
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	vlclbf	7.1.1.8
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9 Note: Not Simple Profile
}			
MVMODE	1	uimsbf	7.1.1.32
DIRECTMB	Bitplane		7.1.1.42
SKIPMB	Bitplane		7.1.1.37
MVTAB	2	uimsbf	7.1.1.38
CBPTAB	2	uimsbf	7.1.1.39
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 7.1.1.31
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	7.1.1.40
if (TTMBF == 1) {			

TTFRM	2	uimsbf	7.1.1.41
}			
}			
TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSDCTAB	1	uimsbf	7.1.1.13
for ('all macroblocks') {			'all macroblocks' represents all macroblocks in the frame. Sync markers, if present, shall be handled as defined in **8.8**
B MAIN/ADV MB ( )			Table 30
}			
}			

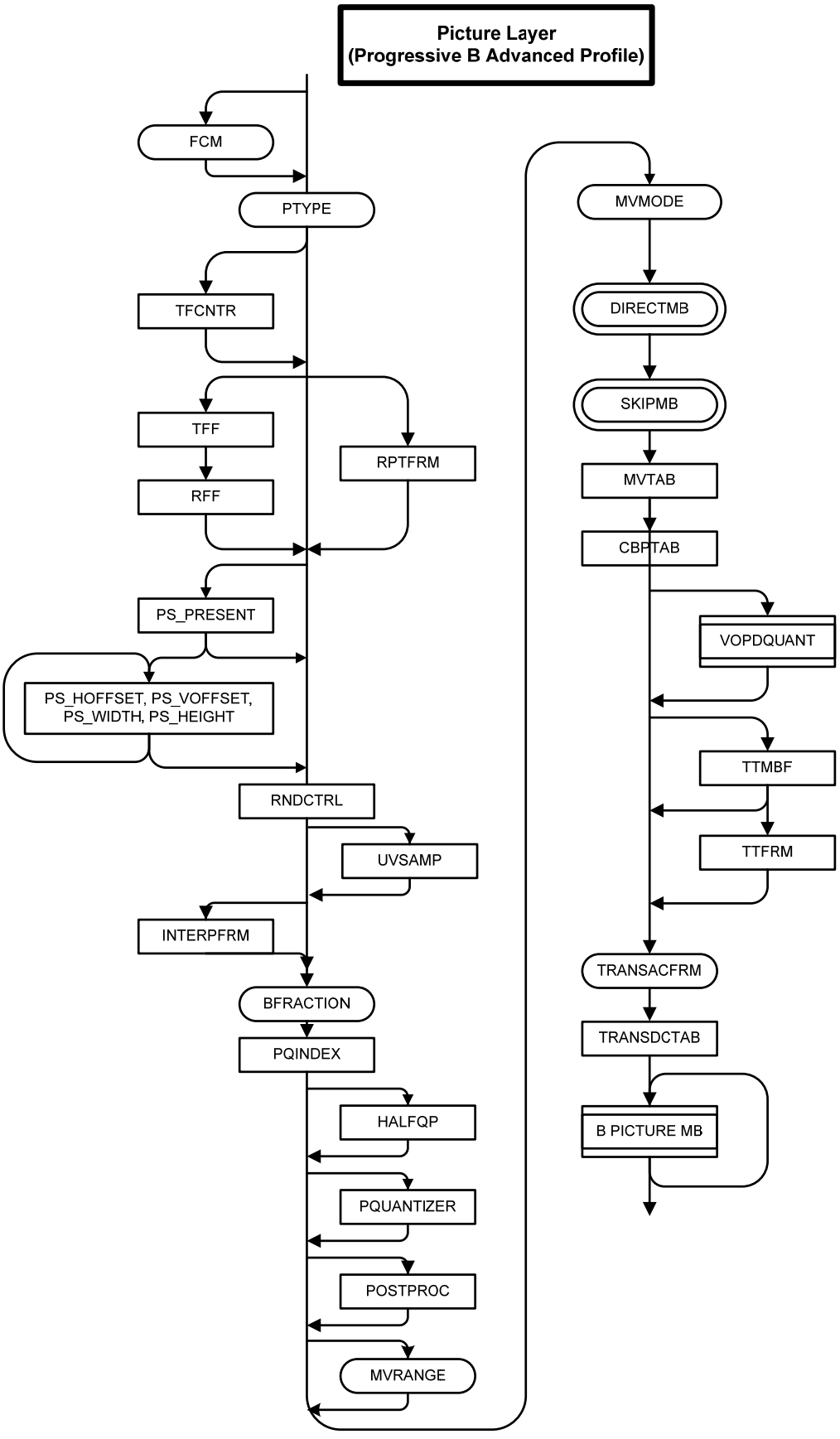


Figure 19: Syntax diagram for the Progressive B picture layer bitstream in Advanced Profile.

Table 22: Progressive B picture layer bitstream for Advanced Profile

B ADV PICTURE ( ) {	Number of bits	Descriptor	Reference
If (INTERLACE == 1)			6.1.9
FCM	Variable size	vlclbf	7.1.1.15 // FCM==0 in this table.
PTYPE	Variable size	vlclbf	7.1.1.4 In this table, PTYPE is 10b
if (TFCNTRFLAG) {			6.1.10
TFCNTR	8	uimsbf	7.1.1.16
}			
if (PULLDOWN) {			6.1.8
if (INTERLACE == 0 || PSF == 1) {			6.1.9, 6.1.13
RPTFRM	2	uimsbf	7.1.1.19
}			
else {			
TFF	1	uimsbf	7.1.1.17
RFF	1	uimsbf	7.1.1.18
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	7.1.1.20
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	7.1.1.21
PS_VOFFSET	18	uimsbf	7.1.1.22
PS_WIDTH	14	uimsbf	7.1.1.23
PS_HEIGHT	14	uimsbf	7.1.1.24
}			
}			
}			

RNDCTRL	1	uimsbf	7.1.1.25
if (INTERLACE == 1)			6.1.9
UVSAMP	1	uimsbf	7.1.1.26
if (FINTERPFLAG == 1) {			6.1.11
INTERPFRM	1	uimsbf	7.1.1.1
}			
BFRACTION	Variable size	vlclbf	7.1.1.14
PQINDEX	5	uimsbf	7.1.1.6
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	7.1.1.7
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	7.1.1.8
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	7.1.1.27
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	7.1.1.9
}			
MVMODE	1	uimsbf	7.1.1.32
DIRECTMB	Bitplane		7.1.1.42
SKIPMB	Bitplane		7.1.1.37
MVTAB	2	uimsbf	7.1.1.38
CBPTAB	2	uimsbf	7.1.1.39
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 7.1.1.31
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	7.1.1.40
if (TTMBF == 1) {			
TTFRM	2	uimsbf	7.1.1.41
}			
}			

TRANSACFRM	Variable size	vlclbf	7.1.1.11
TRANSDCTAB	1	uimsbf	7.1.1.13
for (‘all macroblocks’) {			‘all macroblocks’ represents all macroblocks in this BDU.
B MAIN/ADV MB( )			Table 30
}			
}			

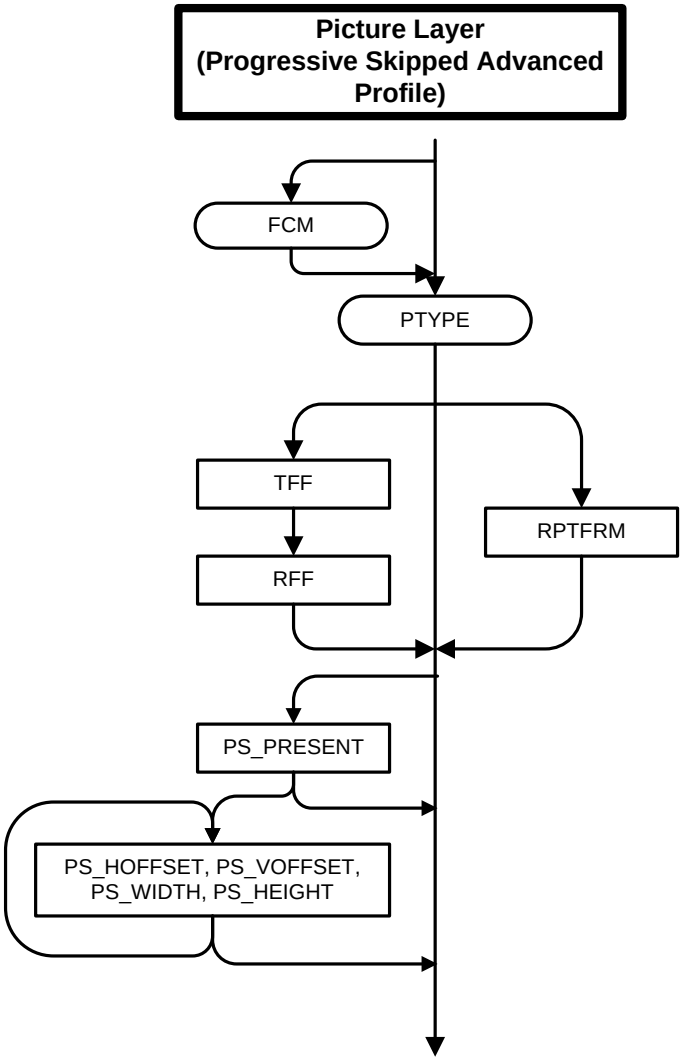


Figure 20: Syntax diagram for the Progressive Skipped picture layer bitstream in Advanced Profile.

Table 23: Progressive Skipped picture layer bitstream for Advanced Profile

SKIPPED ADV PICTURE ( ) {	Number of bits	Descriptor	Reference
If (INTERLACE == 1)			6.1.9
FCM	Variable size	vlclbf	7.1.1.15 // FCM==0b in this table.
PTYPE	Variable size	vlclbf	7.1.1.4 In this table PTYPE is 1111b
if (PULLDOWN) {			6.1.8
if (INTERLACE == 0 || PSF == 1) {			6.1.9, 6.1.13
RPTFRM	2	uimsbf	7.1.1.19
}			
else {			
TFF	1	uimsbf	7.1.1.17
RFF	1	uimsbf	7.1.1.18
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	7.1.1.20
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	7.1.1.21
PS_VOFFSET	18	uimsbf	7.1.1.22
PS_WIDTH	14	uimsbf	7.1.1.23
PS_HEIGHT	14	uimsbf	7.1.1.24
}			
}			
}			
}			

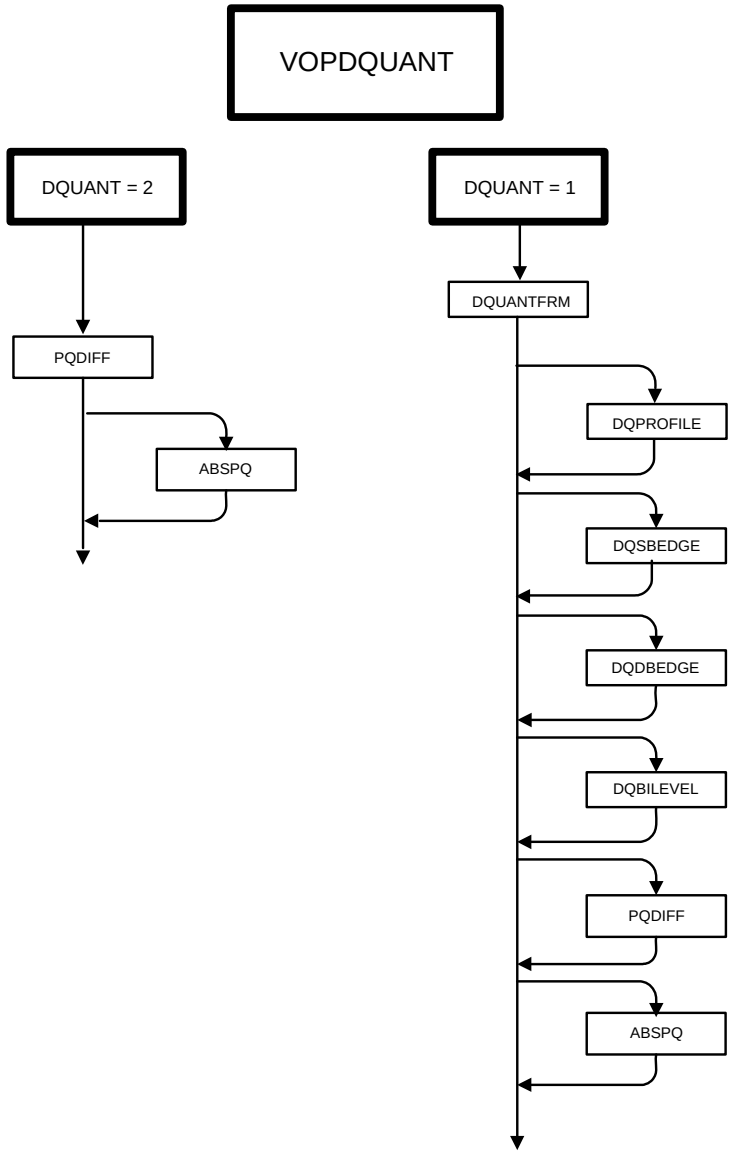


Figure 21: Syntax diagram for VOPDQUANT in picture header

Table 24: VOPDQUANT in picture header (Refer to 7.1.1.31)

VOPDQUANT( ) {	Number of bits	Descriptor	Reference
if (DQUANT == 2) {			6.2.8 (Annex J)
DQUANT_InFrame = TRUE // quantizer can vary in frame			See Note below.
PQDIFF	3	uimsbf	7.1.1.31.6
if (PQDIFF == 7) {			
ABSPQ	5	uimsbf	7.1.1.31.7
}			

}			
else {			
DQUANTFRM	1	uimsbf	7.1.1.31.1
if (DQUANTFRM == 1) {			
DQUANT_InFrame = TRUE //quantizer can vary in frame			See Note below.
DQPROFILE	2	uimsbf	7.1.1.31.2
if (DQPROFILE == 'Single Edge') {			
DQSBEDGE	2	uimsbf	7.1.1.31.3
}			
if (DQPROFILE == 'Double Edge') {			
DQDBEDGE	2	uimsbf	7.1.1.31.4
}			
if (DQPROFILE == 'All Macroblocks') {			
DQBILEVEL	1	uimsbf	7.1.1.31.5
}			
if (!(DQPROFILE == 'All macroblocks' && DQBILEVEL == 0)) {			
PQDIFF	3	uimsbf	7.1.1.31.6
if (PQDIFF == 7) {			
ABSPQ	5	uimsbf	7.1.1.31.7
}			
}			
}			
else { //DQUANTFRM is 0			
DQUANT_InFrame = FALSE; //same quantizer (PQUANT) is used for entire frame			See Note below.
}			
}			
}			
Note: Refer to section 7.1.1.31 for a definition of the syntax elements in VOPDQUANT. Note: DQuant_InFrame is TRUE if quantizer can vary between the macroblocks of a frame. DQuant_InFrame == FALSE if only PQUANT is used for all MBs in the frame.			

Table 25: Bitplane coding (Refer to 7.2)

BITPLANE( ) {	Number of bits	Descriptor	Reference
INVERT	1	uimsbf	7.2.1
IMODE	Variable size	vlclbf	7.2.2
DATABITS	Variable size	vlclbf	7.2.3
}			

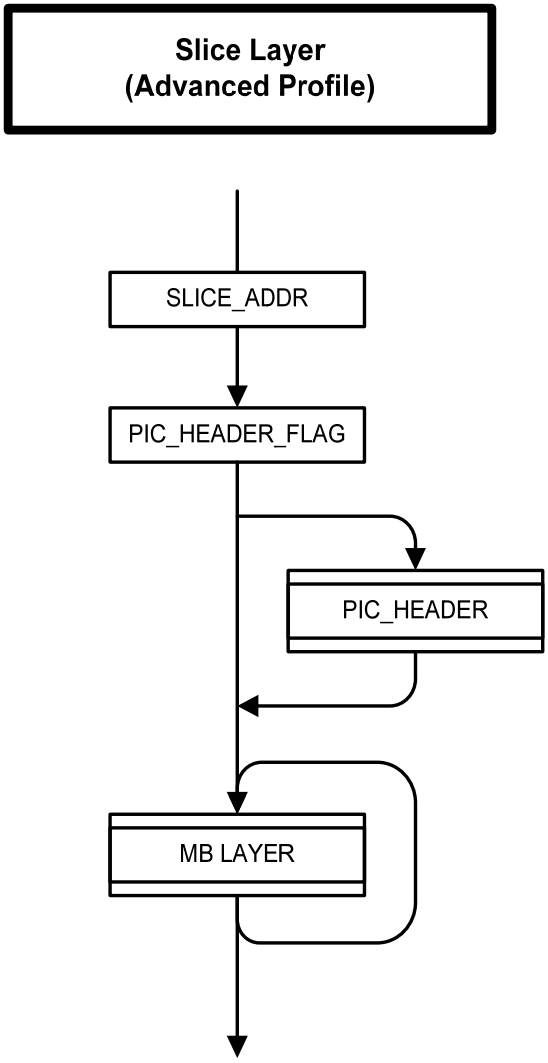


Figure 22: Syntax diagram for the Slice-Layer bitstream in the Advanced Profile

Table 26: Slice-Layer bitstream in Advanced Profile

SLICE ADV ( ) {	Number of bits	Descriptor	Reference
SLICE_ADDR	9	uimsbf	7.1.2.1
PIC_HEADER_FLAG	1	uimsbf	7.1.2.2
if (PIC_HEADER_FLAG == 1) {			
PICTURE_LAYER( )			
}			
for (‘all macroblocks’) {			‘all macroblocks’ represents all macroblocks in this slice layer BDU.
MB_LAYER ( )			Table 28, Table 29 or Table 30 as appropriate. See note below.
}			
}			
Note: The MB layer syntax may be I picture as defined as in Table 28, P picture as defined in Table 29, or B picture as defined in Table 30. The choice of table referenced by the MB layer syntax layer is defined by the kind of picture to which this slice belongs.			

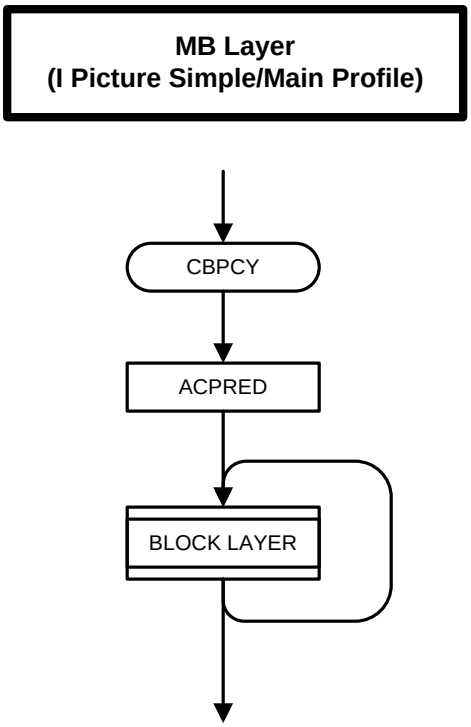


Figure 23: Syntax diagram for macroblock layer bitstream in Progressive I and BI picture for simple/main profile

Table 27: Macroblock layer bitstream in Progressive I and BI picture for Simple/Main Profile

I AND BI SIMPLE/MAIN MB( ) {	Number of bits	Descriptor	Reference
CBPCY	Variable size	vlclbf	7.1.3.1
ACPRED	1	uimsbf	7.1.3.2
for ('all blocks in MB') {			
INTRA_BLOCK( )			Table 31
}			
}			

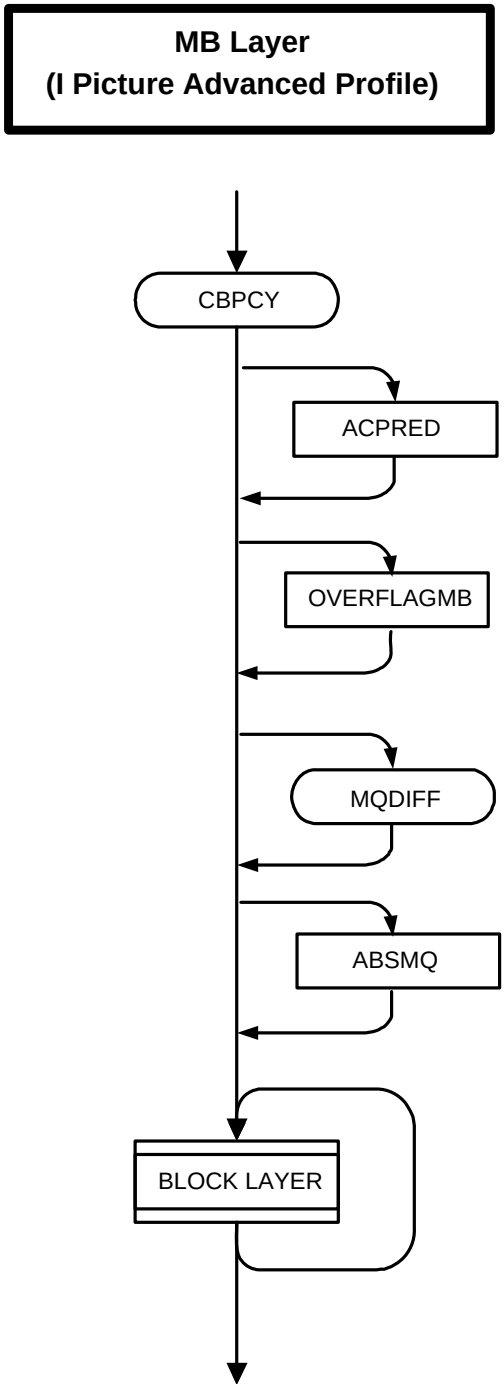


Figure 24: Syntax diagram for macroblock layer bitstream in progressive I picture for advanced profile

Table 28: Macroblock layer bitstream in Progressive I and BI picture for Advanced Profile

I AND BI ADV MB( ) {	Number of bits	Descriptor	Reference
CBPCY	Variable size	vlclbf	7.1.3.1
if (ACPRED Coding Mode == 'Raw') {			7.1.1.28, 7.2.2
ACPRED	1	uimsbf	7.1.3.2
}			
if (CONDOVER == 11b && OVERFLAGS Coding Mode == 'Raw') {			7.1.1.29, 7.1.1.30, 7.2.2
OVERFLAGMB	1	uimsbf	7.1.3.3
}			
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == 'All Macroblocks') {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	7.1.3.5
}			
}			
}			
}			
for ('all blocks in MB') {			
INTRA_BLOCK( )			Table 31
}			
}			

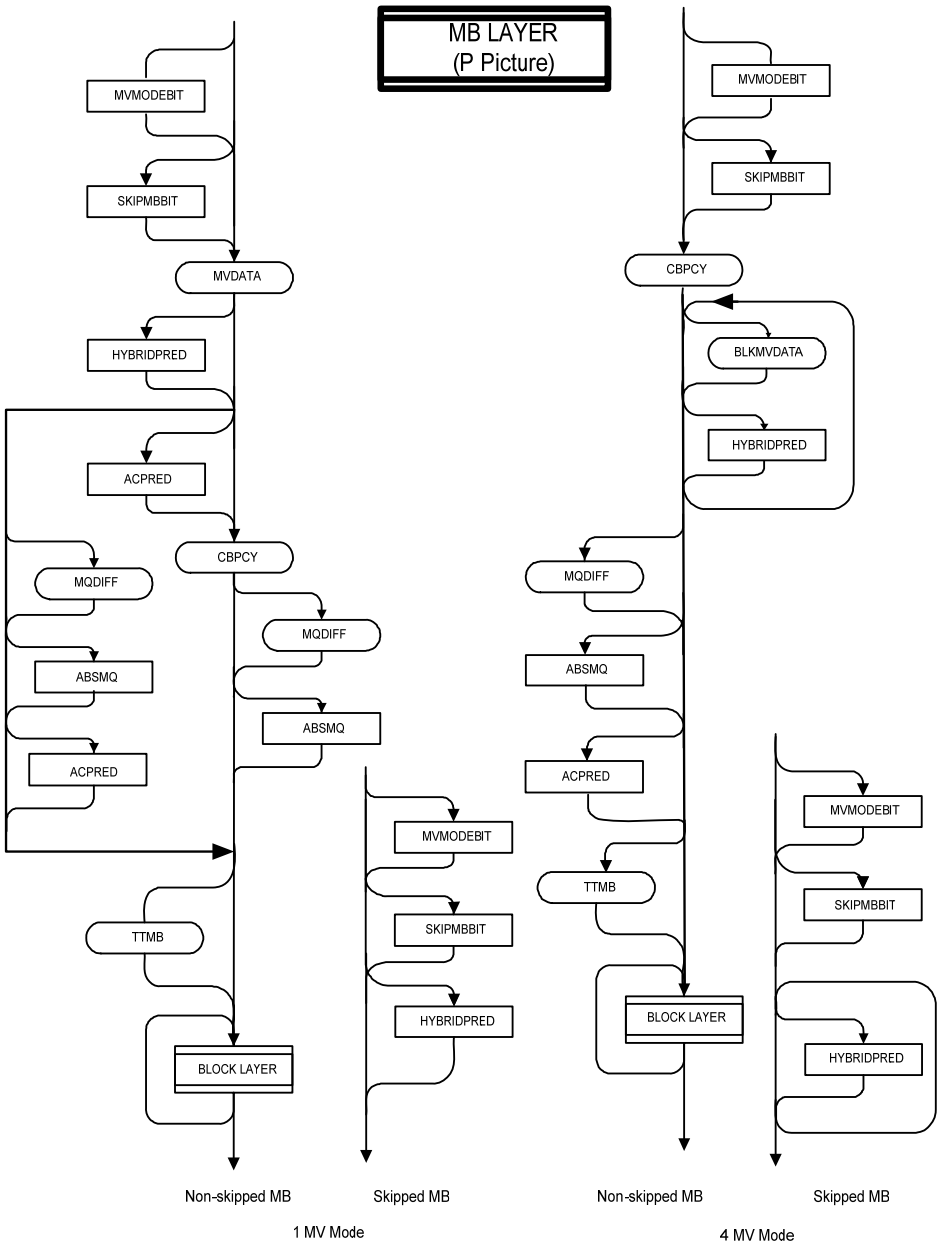


Figure 25: Syntax diagram for macroblock layer bitstream in Progressive-P picture for Simple/Main/Advanced Profiles

Table 29: Macroblock layer bitstream in Progressive P picture for Simple/Main/Advanced Profile

P SIMPLE/MAIN/ADV MB( ) {	Number of bits	Descriptor	Reference
if ((MVMODE == 'Mixed-MV' || (MVMODE == 'Intensity Compensation' && MVMODE2 == 'Mixed-MV') && MVTYPEMB Coding Mode == 'Raw') {			7.1.1.32, 7.1.1.33, 7.1.1.36
MVMODEBIT	1	uimsbf	7.1.3.6

}			
if (SKIPMB Coding Mode == 'Raw') {			7.1.1.37
SKIPMBBIT	1	uimsbf	7.1.3.7
}			
if (1-MV mode) {			
if (non-skipped MB) {			
MVDATA	Variable size	vlclbf	7.1.3.8
if ('hybridpred syntax element for MV prediction is present') {			Conditions for presence of hybridpred syntax element is defined in Figure 55 in 8.3.5.3.5
HYBRIDPRED	1	uimsbf	7.1.3.9
}			
if ('intra_flag' && 'more_present flag' == 0) {			For 'intra_flag' and 'more_present flag', see 8.3.5.2.1
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == 'All Macroblocks') {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	7.1.3.5
}			
}			
}			
}			
ACPRED	1	uimsbf	7.1.3.2
}			
else if ('more_present flag' == 1){			8.3.5.2.1
if ('intra_flag') {			8.3.5.2.1
ACPRED	1	uimsbf	7.1.3.2
}			

CBPCY	Variable size	vlclbf	7.1.3.1
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == 'all macroblocks') {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	7.1.3.5
}			
}			
}			
}			
}			
if (TTMBF == 0 && !('intra_flag') && 'at least 1 coded block') { /* the presence of 'at least 1 coded block' is inferred from more_present flag. Since DC coefficients are not treated separately, a coded inter-block has at least one non-zero coefficient */			For TTMBF, see 7.1.1.40 For intra_flag see 8.3.5.2.1
TTMB	Variable size	vlclbf	7.1.3.10
}			
for ('all blocks in MB') {			
if ('intra_flag' || 'coded block') /* coded block is inferred from CBPCY. */			For intra_flag see 8.3.5.2.1. For coded block, see 8.3.5.5.
BLOCK( )			Table 31 if intra block or Table 32 otherwise
}			
} /* non-skipped MB */			
else { /* skipped MB */			
if ('hybridpred syntax element for MV prediction is present') {			Conditions for presence of hybridpred syntax element is defined in Figure 55 in 8.3.5.3.5

HYBRIDPRED	1	uimsbf	7.1.3.9
}			
} /* skipped MB */			
} /* 1-MV mode */			
else { /* 4-MV mode */			
if (non-skipped MB) {			
CBPCY	Variable size	vlclbf	7.1.3.1
for (‘each of the 4 Y-blocks’) {			
if (‘CBPCY bit set for this block’) {			
BLKMVDATA	Variable size	vlclbf	7.1.3.11
}			
if (‘hybridpred syntax element for MV prediction is present’) {			Conditions for presence of hybridpred syntax element is defined in Figure 55 in 8.3.5.3.5
HYBRIDPRED	1	uimsbf	7.1.3.9
}			
}			
If (‘all blocks are inter’ && ‘all blocks have zero AC coefficients’)			//inferred via intra_flag and more_present_flag of BLKMVDATA as described in 8.3.5.2.1
goto End4MV;			
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == ‘all macroblocks‘) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	7.1.3.5
}			

}			
}			
}			
if (‘any block is intra’ && ‘non-zero prediction for that block’) {			//intra status inferred intra_flag of BLKMVDATA
ACPRED	1	uimsbf	7.1.3.2
}			
if (TTMBF == 0 && ‘at least 1 coded inter-block’) { /* The coded status of block is inferred from more_present flag of MVDATA. A coded inter-block has at least one non-zero coefficient. */			For TTMBF, see 7.1.1.40 For more_present_flag see 8.3.5.2.1
TTMB	Variable size	vlclbf	7.1.3.10
}			
for (‘all blocks in MB’) {			
If (‘intra block’ || ‘coded inter-block’) /* intra block is deduced from intra_flag, and ‘coded inter-block’ is deduced from more_present_flag of BLKMVDATA. A coded inter-block has at least one non-zero coefficient. */			For intra_flag and more_present_flag see 8.3.5.2.1
BLOCK( )			Table 31 if intra block or Table 32 otherwise
}			
End4MV: /* End of 4-MV */			
} /* non-skipped MB */			
else { /* skipped MB */			
for (‘all 4 Y-blocks’) {			
if (‘hybridpred syntax element for MV prediction is present’) {			Conditions for presence of hybridpred syntax element is defined in Figure 55 in 8.3.5.3.5
HYBRIDPRED	1	uimsbf	7.1.3.9
}			
}			
} /* skipped MB */			
} /* 4-MV mode */			

}			

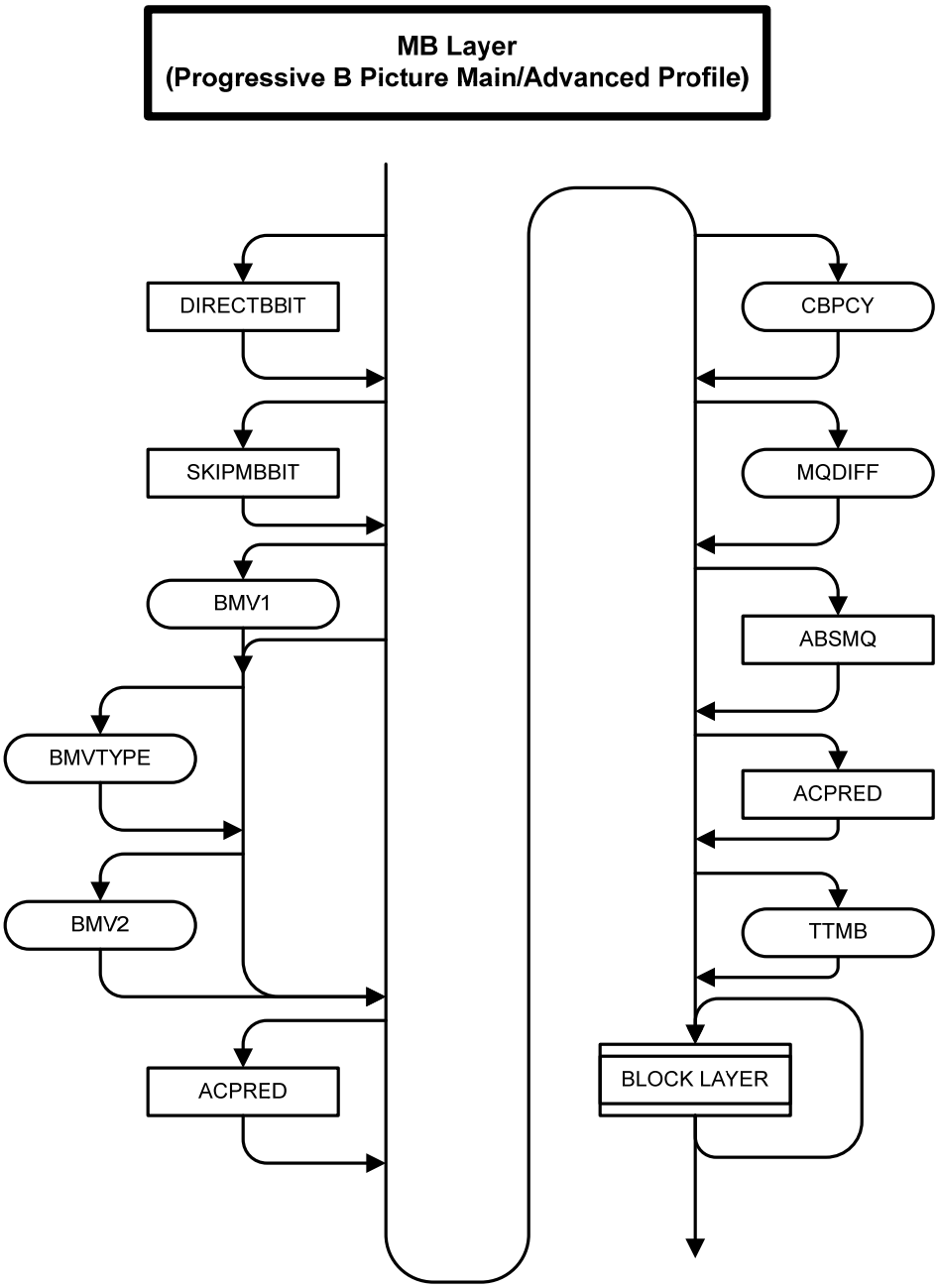


Figure 26: Syntax diagram for macroblock layer bitstream in Progressive B picture for Main/Advanced Profiles

Table 30: Macroblock layer bitstream in Progressive B picture for Main/Advanced Profile

B MAIN/ADV MB( ) {	Number of bits	Descriptor	Reference
if (DIRECTMB Coding Mode == 'Raw') {			7.1.1.42

DIRECTBBIT	1	uimsbf	7.1.3.12
}			
if (SKIPMB Coding Mode == 'Raw') {			
SKIPMBBIT	1	uimsbf	7.1.3.7
}			
if (!DIRECTBBIT) {			7.1.1.42, 7.1.3.12
if (!SKIPMBBIT) {			7.1.1.37, 7.1.3.7
BMV1	Variable size	vlclbf	7.1.3.13
}			
if (!('intra_flag of BMV1') || SKIPMBBIT) {			8.3.5.2.1, 7.1.1.37, 7.1.3.7
BMVTYPE	Variable size	vlclbf	7.1.3.14
}			
}			
if (SKIPMBBIT)			7.1.1.37, 7.1.3.7
goto End;			
if (DIRECTBBIT)			7.1.1.42, 7.1.3.12
goto DecodeCBPCY;			
If ('more_present flag' of BMV1 == 0 && 'Inter MB') /* 'Inter MB' is deduced if 'intra_flag' of BMV1 is 0 */			8.3.5.2.1
goto End:			
if ('intra_flag' && 'more_present flag' of BMV1 == 0) {			8.3.5.2.1,
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == 'all macroblocks') {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7)			
ABSMQ	5	uimsbf	7.1.3.5
}			
}			

}			
ACPRED	1	uimsbf	7.1.3.2
goto DecodeCoeff:			
}			
if (BMVTYPE == 'Interpolated') {			See 7.1.3.14. Also implies that the 'more_present flag' of BMV1 was 1
BMV2	Variable size	vlclbf	7.1.3.15 For interpolated MBs, BMV1 is backward MV and BMV2 is forward MV. Also note that BMV2 cannot indicate Intra.
if ('more_present flag' of BMV2 == 0)			See 8.3.5.2.1 for more_present flag
goto End;			
}			
if ('intra_flag')			8.3.5.2.1
ACPRED	1	uimsbf	7.1.1.28
DecodeCBPCY:			
CBPCY	Variable size	vlclbf	7.1.3.1
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == 'all macroblocks') {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	7.1.3.4
} else {			
MQDIFF	3	uimsbf	7.1.3.4
if (MQDIFF == 7)			
ABSMQ	5	uimsbf	7.1.3.5
}			
}			
}	Variable size	vlclbf	
if (TTMBF == 0 && 'Inter MB' && 'at least one coded block') { /* 'Inter MB' is deduced if 'intra_flag' of BMV1 is 0. The presence of 'at least 1 coded			For TTMBF, see 7.1.1.40 For more_present flag, see 8.3.5.2.1

block' is inferred from more_present flag. A coded block in inter-MB has at least one non-zero coefficient */			
TTMB	Variable size	vlclbf	7.1.3.10
}			
DecodeCoeff:			
for ('all blocks in MB') {			
if ('intra_flag' || 'coded block') /* The presence of 'coded block' is inferred from more_present flag. A coded block in inter-MB has at least one non-zero coefficient */			For more_present flag, see 8.3.5.2.1
BLOCK( )			Table 31 if intra block or Table 32 otherwise
}			
End:			
}			

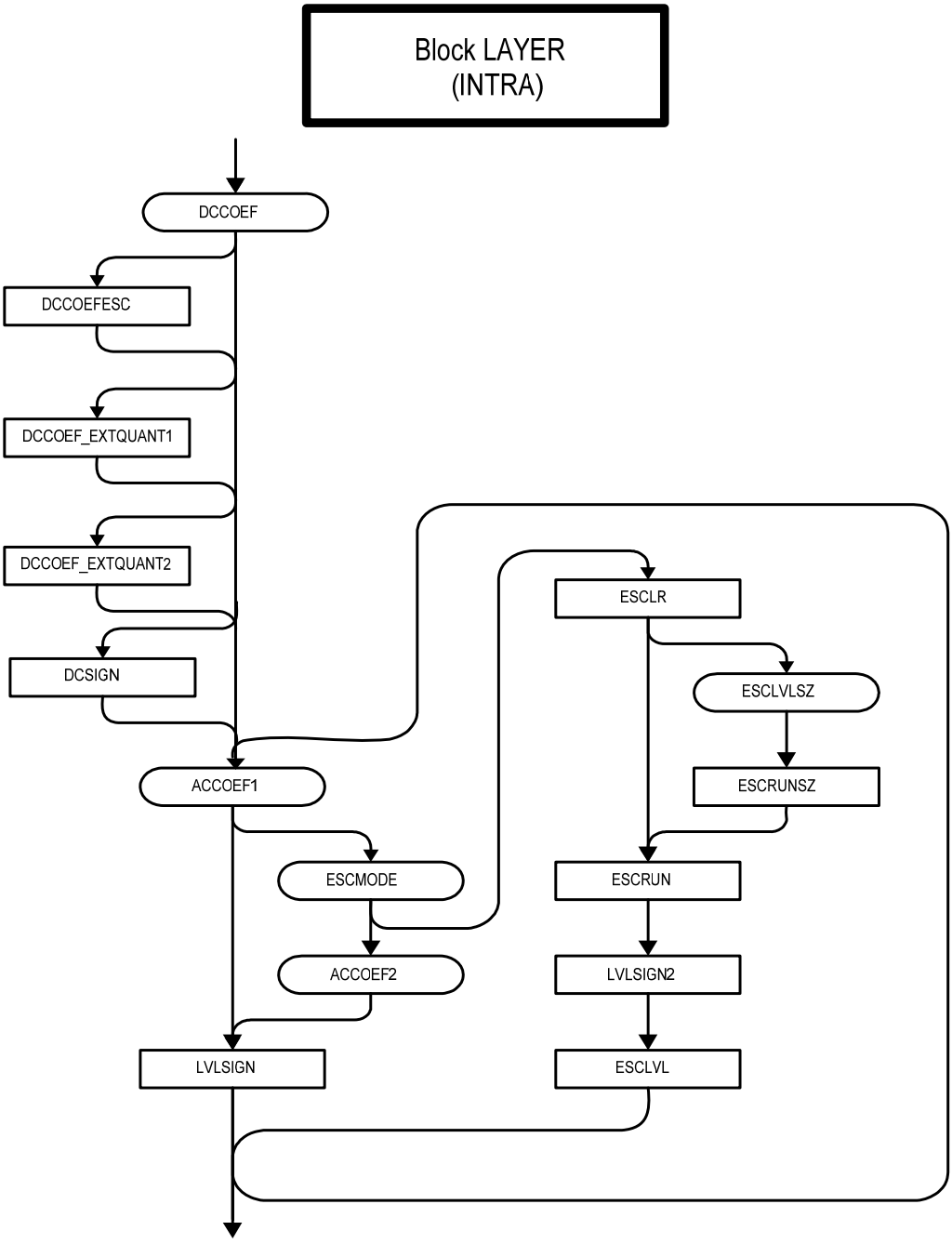


Figure 27: Syntax diagram for the Intra-coded block layer bitstream.

Table 31: Intra block layer bitstream

INTRA_BLOCK( ) {	Number of bits	Descriptor	Reference
DCCOEF	Variable size	vlclbf	7.1.4.1
if (DCCOEF != 0) {			
if (DCCOEF == 'ESCAPECODE') {			Figure 37 in 8.1.3.1

DCCOEFESC	Variable size	vlclbf	7.1.4.2
}			
else {			
if (QUANT == 1)			QUANT refers to MQUANT as described in 8.1.3.1
DCCOEF_EXTQUANT1	2	uimsbf	7.1.4.3
else if (QUANT == 2)			QUANT refers to MQUANT as described in 8.1.3.1
DCCOEF_EXTQUANT2	1	uimsbf	7.1.4.4
}			
DCSIGN	1	uimsbf	7.1.4.5
}			
if ('intra block has AC coefficient') {			presence of AC coefficient is inferred from CBPCY
while (!(last_flag)) { /* last_flag == 1 implies that all coefficients have been decoded in this block. last_flag == 0 implies that more coefficients are present. */			see 8.1.3.4 for decoding of last_flag
ACCOEF1	Variable size	vlclbf	7.1.4.6
if (ACCOEF1 == 'Escape Index') {			Escape Index has been described in Figure 41 in 8.1.3.4
ESCMODE	Variable size	vlclbf	7.1.4.7
if (ESCMODE == 'mode1' || ESCMODE == 'mode2') {			7.1.4.8, 8.1.3.4
ACCOEF2	Variable size	vlclbf	7.1.4.8
}			
else { /* 'escape mode 3' */			7.1.4.9, 8.1.3.4
ESCLR	1	uimsbf	7.1.4.9
if ('ESCMODE == mode3' && 'for first time') { /* This condition is triggered when mode3 is used (ESCMODE is equal to mode3) for the first time in a frame, field or slice as described in 8.1.3.4*/			8.1.3.4
ESCLVLSZ	Variable	vlclbf	7.1.4.10

	e size		
ESCRUNSZ	2	uimsbf	7.1.4.11
}			
ESCRUN	Variable size		7.1.4.12
LVLSGN2	1	uimsbf	7.1.4.13
ESCLVL	Variable size		7.1.4.14
}			
} /* 'escape mode' */			
if (ESCMODE != 'mode3') {			Figure 41 in 8.1.3.4
LVLSIGN	1	uimsbf	7.1.4.15
}			
} /* while ( ) */			
} /* if intra block has AC coefficient */			
}			

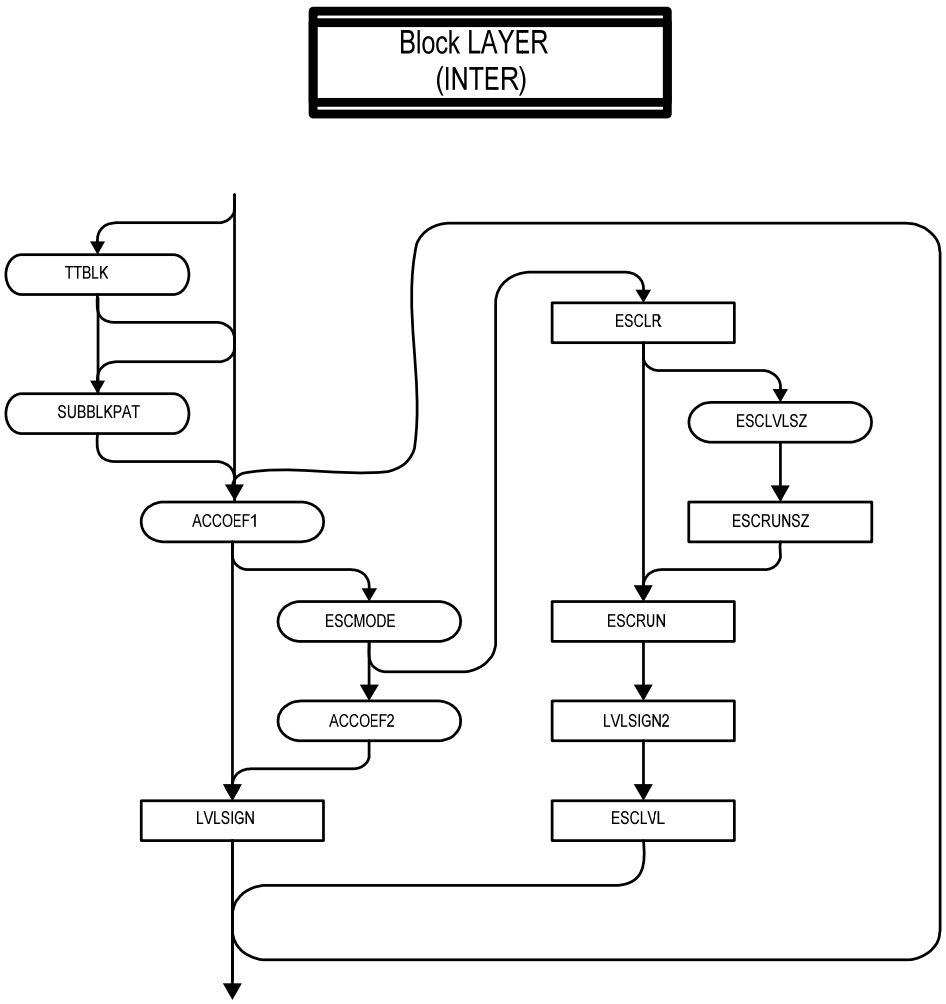


Figure 28: Syntax diagram for the Inter-coded block layer bitstream.

Table 32: Inter block layer bitstream

INTER BLOCK( ) {	Number of bits	Descriptor	Reference
if (TTMB == 'block' && !(' the first inter coded block')) {			7.1.3.10
TTBLK	Variable size	vlclbf	7.1.4.16
}			
if ('transform type is 4x4' || ('transform type is 8*4, 4*8 && (TTMBF == '1' || (TTMB == 'Macroblock' && !('the first inter coded block'))))) {			7.1.1.40, 7.1.3.10
SUBBLKPAT	Variable size	vlclbf	7.1.4.17

}			
for (i = 0; i < SUBBLKNUM; i++) {			SUBBLKNUM = 1, 2, 4 for Inter 8x8, 8x4/4x8, 4x4 blocks respectively
if (IsSubBlkCoded(i)){ /* IsSubBlkCoded(i) is 1 if the subblock i has at least one non-zero coefficient, 0 otherwise. */			The value of IsSubBlkCoded(i) is derived from either TTMB 7.1.3.10, or SUBBLKPAT 7.1.4.17
while (!(last_flag)) { /* last_flag == 1 implies that all coefficients have been decoded in this block. last_flag == 0 implies that more coefficients are present. */			see 8.1.3.4 for decoding of last_flag
ACCOEF1	Variable size	vlclbf	7.1.4.6
if (ACCOEF1 == 'Escape Index') {			Escape Index has been described in 8.1.3.4
ESCMODE	Variable size	vlclbf	7.1.4.7
if (ESCMODE == 'mode1' || ESCMODE == 'mode2') {			7.1.4.8, 8.1.3.4
ACCOEF2	Variable size	vlclbf	7.1.4.8
}			
else { /* 'escape mode 3' */			7.1.4.9, 8.1.3.4
ESCLR	1	uimsbf	7.1.4.9
if (ESCMODE == 'mode 3' for the first time') { /* This condition is triggered when mode3 is used (ESCMODE is equal to mode3) for the first time in a frame, field or slice as described in 8.1.3.4*/			8.1.3.4
ESCLVLSZ	Variable size	vlclbf	7.1.4.10
ESCRUNSZ	2	uimsbf	7.1.4.11
}			
ESCRUN	Variable size		7.1.4.12
LVLSGN2	1	uimsbf	7.1.4.13
ESCLVL	Variable size		7.1.4.14

}			
} /* 'escape mode' */			
if (ESCMODE != 'mode3') {			Figure 41 in 8.1.3.4
LVLSIGN	1	uimsbf	7.1.4.15
}			
} /* while ( ) */			
} // IsSubBlkCoded(i)			
}//for i= 0; i<SUBBLKNUM; i++			
}			
Note: 'the first inter coded block' is 1 if the block is the first inter-coded block in the macroblock, and 0 otherwise.			

7.1.1 Picture layer

Data for each picture shall consist of a picture header followed by data for the macroblock layer. Figure 13, Figure 14 and Figure 15 show the bitstream elements that make up the I and BI progressive picture layer in simple/main profile and advanced profile, respectively, and Figure 16 and Figure 17 show the bitstream elements that make up the P progressive picture layer in simple/main profile and advanced profile, respectively.

7.1.1.1 Frame Interpolation Hint (INTERPFRM) (1 bit)

INTERPFRM is a 1-bit syntax element that shall be present in all progressive frame types for all profiles, if the syntax element FINTERPFLAG == 1. This bit is not used in the decoding process. Its intended purpose is to provide a hint to the display process that the current temporal region is suitable for temporal interpolation.

Note: The need to perform frame interpolation or the method of doing so is outside the scope of this document.

If INTERPFRM == 0, then the current temporal region (the current frame and its surrounding frames) is considered unsuitable for frame interpolation. If INTERPFRM == 1, then the temporal region is considered suitable for frame interpolation. For example, the display process can use interpolation to increase the displayed frame-rate when INTERPFRM == 1. It is important to reemphasize that this interpolation is outside the decoding process.

7.1.1.2 Frame Count (FRMCNT) (2 bits)

FRMCNT is a 2-bit syntax element that shall be present in all picture headers for simple and main profiles. The value of this field should be used as a frame count and shall be incremented by one for each consecutive frame (in coded order) in the bitstream. FRMCNT has no effect on the decoding or display process.

Note: FRMCNT is a modulo 4 counter that can be used to check if any frames have been lost at the decoder input.

7.1.1.3 Range Reduction Frame (RANGEREDFRM) (1 bit)

RANGEREDFRM is a 1-bit syntax element present in all frame types, for the main profile only, if the sequence level flag RANGERED == 1. If RANGEREDFRM == 1, then range reduction shall be used for the frame. If RANGEREDFRM == 0, then range reduction shall not be used for the frame. For a B Frame or a BI Frame, the value of the RANGEREDFRM syntax element shall be identical to the value of the corresponding syntax element in the subsequent anchor frame (in display order). See sections 8.1.1.4 and 8.3.4.11 for a description of range reduction decoding.

7.1.1.4 Picture Type (PTYPE) (Variable size)

For simple and main profiles:

If the sequence level syntax element MAXBFRAMES == 0, then PTYPE shall be as defined in Table 33.

Table 33: Simple/Main Profile Picture Type FLC if MAXBFRAMES == 0

PTYPE FLC	Picture Type
0	I
1	P

If MAXBFRAMES > 0, then PTYPE shall be as defined in Table 34.

Table 34: Main Profile Picture Type VLC if MAXBFRAMES > 0

PTYPE VLC	Picture Type
1b	P
01b	I
00b	B or BI

In simple and main profiles, if the size of any coded picture is less than or equal to one byte, that picture shall be treated as a skipped frame.

Note: In simple and main profiles, the size of coded picture is passed to the decoder via the Transport Layer.

For advanced profile:

PTYPE shall be as defined in Table 35.

Table 35: Advanced Profile Picture Type VLC

PTYPE VLC	Picture Type
110b	I
0b	P
10b	B
1110b	BI
1111b	Skipped

If PTYPE indicates that the frame is skipped, then the frame shall be reconstructed as a P frame which is identical to its reference frame. The reconstruction of the skipped frame is equivalent conceptually to copying the reference frame. A skipped picture means that no further data is transmitted for this frame.

7.1.1.5 Buffer Fullness (BF) (7 bits)

BF is a 7-bit syntax element that shall be present in simple and main profile I picture headers. BF represents the value of buffer fullness (as a percentage of the buffer size) at the encoder, and shall be in the range of 0 to 100%, inclusive. A value of 100% (1111111b) shall indicate that the encoder buffer is full, and a value of 0 shall indicate that the encoder buffer is empty. This syntax element is not used in the decoding process.

Note: The knowledge of buffer fullness at an I Frame can be useful if decoding begins at this point, and thus facilitates implementation of trick modes.

7.1.1.6 Picture Quantizer Index (PQINDEX) (5 bits)

PQINDEX is a 5-bit syntax element that shall signal the quantizer scale index for the entire frame. It is present in all picture types. If the quantizer is signaled implicitly (this is signaled by syntax element QUANTIZER == 00b), then PQINDEX shall specify both the picture quantizer scale (PQUANT) and the quantizer (uniform or non-uniform) used for the frame. See section 8.1.3.8 for details on dequantization using uniform as well as non-uniform quantizers.

PQINDEX shall be translated to PQUANT for the case where QUANTIZER == 00b (implicit quantizer) as defined in Table 36.

Table 36: PQINDEX to PQUANT/Quantizer Translation (Implicit Quantizer)

PQINDEX	PQUANT	Quantizer	PQINDEX	PQUANT	Quantizer
0	NA	NA	16	13	Non-uniform
1	1	Uniform	17	14	Non-uniform
2	2	Uniform	18	15	Non-uniform
3	3	Uniform	19	16	Non-uniform
4	4	Uniform	20	17	Non-uniform
5	5	Uniform	21	18	Non-uniform
6	6	Uniform	22	19	Non-uniform
7	7	Uniform	23	20	Non-uniform
8	8	Uniform	24	21	Non-uniform
9	6	Non-uniform	25	22	Non-uniform
10	7	Non-uniform	26	23	Non-uniform
11	8	Non-uniform	27	24	Non-uniform
12	9	Non-uniform	28	25	Non-uniform
13	10	Non-uniform	29	27	Non-uniform
14	11	Non-uniform	30	29	Non-uniform
15	12	Non-uniform	31	31	Non-uniform

If the quantizer is signaled explicitly at the sequence or frame level (signaled by syntax element QUANTIZER == 01b, 10b or 11b), then PQUANT shall be equal to PQINDEX for all values of PQINDEX except when PQINDEX is equal to '0'. The case where PQINDEX is equal to 0 shall be SMPTE Reserved. The macroblock quantizer step size (MQUANT) shall be initialized to PQUANT. MQUANT may be modified as described in 7.1.1.31, 7.1.3.4 and 7.1.3.5.

7.1.1.7 Half QP Step (HALFQP) (1 bit)

HALFQP is a 1-bit syntax element that shall be present in all frame types if PQINDEX is less than or equal to 8. The HALFQP syntax element allows the picture quantizer to be expressed in half step increments over the low PQUANT range. If HALFQP == 1, then the picture quantizer step size shall be equal to PQUANT + ½. If HALFQP == 0, then the picture quantizer step size shall be equal to PQUANT. Therefore, if the uniform quantizer is used, then half step sizes are possible up to PQUANT == 9 (i.e., picture quantizer step size = 1, 1.5, 2, 2.5 … 8.5, 9), and then only integer step sizes are allowable above PQUANT == 9. For the non-uniform quantizer, half-step sizes are possible up to PQUANT == 7 (i.e., 1, 1.5, 2, 2.5 … 6.5, 7).

Note: HALFQP applies only if a macroblock is coded using PQUANT, and does not apply if the macroblock is coded with a quantizer value derived from VOPDQUANT syntax elements.

7.1.1.8 Picture Quantizer Type (PQUANTIZER) (1 bit)

PQUANTIZER is a 1 bit syntax element that shall be present in all frame types if the syntax element QUANTIZER == 01b. In this case, the quantizer used for the frame shall be specified by PQUANTIZER. If PQUANTIZER == 0, then the non-uniform quantizer shall be used for the frame. If PQUANTIZER == 1, then the uniform quantizer shall be used.

7.1.1.9 Extended MV Range Flag (MVRANGE) (Variable size)

MVRANGE is a variable-sized syntax element that shall be present for sequences coded using the main and advanced profiles when the sequence-layer EXTENDED_MV == 1. For the main profile, it shall be present in I, P and B pictures. The value of this syntax element shall be ignored in main profile I pictures as this syntax element is not used in I pictures. For the advanced profile, it shall be present in P, and B pictures.

For the simple profile, the default range shall be used. For the main profile and advanced profile, when EXTENDED_MV == 0, the default range shall be used. The default range of motion vectors is [-64 63.f] X [-32 31.f], where f is the fraction ¾ for ¼ pixel motion and ½ for ½ pixel motion resolution. In other words, the default range for quarter-pixel motion modes is [-64 63¾] along the horizontal (X) axis and [-32 31¾] along the vertical (Y) axis.

When EXTENDED_MV == 1, the MVRANGE meaning shall be as defined in Table 37. Section 8.3.5.2 details the decoding of differential motion vectors, at the macroblock layer, for different ranges specified by MVRANGE.

Table 37: Motion Vector Range Signaled by MVRANGE

MVRANGE Value	MV range in full pixel units (horizontal x vertical)
0b (also default)	[-64, 63.f] x [-32, 31.f]
10b	[-128, 127.f] x [-64, 63.f]
110b	[-512, 511.f] x [-128, 127.f]
111b	[-1024,1023.f] x [-256, 255.f]

The value of the MVRANGE syntax element of a B picture shall be greater or equal to the value of the MVRANGE syntax element of the subsequent (in display order) anchor P picture. The MVRANGE of a B picture may take any value defined in Table 37 if the subsequent anchor is an I picture or a skipped picture.

7.1.1.10 Picture Resolution Index (RESPIC) (2 bits)

RESPIC is a 2 bit syntax element that shall be present in progressive I and P pictures if MULTIRES ==1. This syntax element specifies the scaling factor of the current frame relative to the full resolution frame. The RESPIC syntax element shall be as defined in Table 38. Refer to section 8.1.1.3 for a description of variable resolution coding. The RESPIC syntax element of a P picture header shall carry the same value as the RESPIC syntax element of the closest preceding I frame.

Table 38: **Progressive picture resolution code-table**

RESPIC FLC	Horizontal Scale	Vertical Scale
00b	Full	Full
01b	Half	Full
10b	Full	Half
11b	Half	Half

7.1.1.11 Frame-level Transform AC Coding Set Index (TRANSACFRM) (Variable size)

TRANSACFRM is a variable-sized syntax element that shall be present in all frame types and shall be as defined in Table 39. See sections 8.1.1.1 and 8.3.4.9 for a description of the TRANSACFRM syntax element.

Table 39: **Transform AC coding set index code-table**

TRANSACFRM	Coding set index
0b	0
10b	1
11b	2

7.1.1.12 Frame-level Transform AC Table-2 Index (TRANSACFRM2) (Variable size)

TRANSACFRM2 is a variable-sized syntax element that shall be present in I frames, and shall be as defined in Table 39. See section 8.1.1.1 for a description of the Transform AC coding sets.

7.1.1.13 Intra Transform DC Table (TRANSDCTAB) (1 bit)

TRANSDCTAB is a 1 bit syntax element that shall be present in all frame types. See section 8.1.1.2 for a description.

7.1.1.14 B Picture Fraction (BFRACTION)(Variable size)

BFRACTION is a variable-sized syntax element that shall be present in B picture headers. It shall also be present in BI picture headers of main profile. BFRACTION signals a fraction that may take on a limited set of fractional values between 0 and 1, denoting the relative temporal position of the B frame within the interval formed by its anchors. This fraction shall be used to scale co-located motion vectors for deriving the 'Direct' motion vectors.

The mapping of BFRACTION is defined in Table 40.

Table 40: BFRACTION VLC Table

BFRACTION VLC	Fraction	BFRACTION VLC	Fraction
000b	1/2	1110101b	2/7
001b	1/3	1110110b	3/7
010b	2/3	1110111b	4/7
011b	1/4	1111000b	5/7
100b	3/4	1111001b	6/7
101b	1/5	1111010b	1/8
110b	2/5	1111011b	3/8
1110000b	3/5	1111100b	5/8
1110001b	4/5	1111101b	7/8
1110010b	1/6	1111110b	SMPTE Reserved
111001b1	5/6	1111111b	See below
1110100b	1/7		

For simple and main profiles the value 1111111b shall indicate that the frame is coded as a BI Frame. For advanced profile, this value is SMPTE Reserved, and BI Frames shall be signaled using the PTYPE syntax element.

7.1.1.15 Frame Coding Mode (FCM) (Variable size)

FCM is a variable-sized syntax element that shall be present only in advanced profile, and only if the sequence level syntax element INTERLACE == 1. It indicates whether the frame is coded as progressive, interlace-field or interlace-frame, and shall be as defined in Table 41. B frames shall be constrained to be of the same frame coding mode (i.e. progressive, field-interlace or frame-interlace) as the anchor frame that follows them.

Table 41: Frame Coding Mode VLC

FCM	Frame Coding Mode
0b	Progressive
10b	Frame-Interlace
11b	Field-Interlace

7.1.1.16 Temporal Reference Frame Counter (TFCNTR) (8 bits)

TFCNTR is an 8 bit syntax element that shall be present only in advanced profile, and only if the sequence level syntax element TFCNTRFLAG == 1. TFCNTR of each coded frame shall be incremented by one modulo 256 when examined in display order at the output of the decoding process, except when a sequence header occurs. TFCNTR of the first frame after a sequence header shall be set to zero.

In interlace field pictures the temporal reference coded in the frame header shall be associated with both field pictures in the frame.

7.1.1.17 Top Field First (TFF) (1 bit)

TFF is a 1 bit syntax element that shall be present in advanced profile picture headers if (PULLDOWN == 1 && INTERLACE == 1 && PSF == 0). TFF == 1 shall indicate that the Top Field is the first decoded field. TFF == 0 shall indicate that the bottom field is the first decoded field. If PULLDOWN == 0, TFF shall not be present in the picture header, and Top Field shall be the first decoded field.

7.1.1.18 Repeat First Field (RFF) (1 bit)

RFF is a one bit syntax element that shall be present in advanced profile picture headers if (PULLDOWN == 1 && INTERLACE == 1 && PSF == 0).

Note: RFF can be used during the display process. RFF == 1 implies that the first field can be repeated during display. RFF == 0 implies that no repetition is necessary.

7.1.1.19 Repeat Frame Count (RPTFRM) (2 bits)

RPTFRM is a 2 bit syntax element that shall be present in advanced profile picture headers if (PULLDOWN == 1 && (INTERLACE == 0 || PSF == 1)). RPTFRM shall be set to the number of frames to repeat (0-3).

Note: RPTFRM can be used during the display process. It represents the number of times the frame can be repeated during display.

7.1.1.20 Pan Scan Present Flag (PS_PRESENT) (1 bit)

PS_PRESENT is a 1 bit syntax element that shall be present in all advanced profile picture headers if the entry point header PANSCAN_FLAG == 1. There may be up to four pan scan windows for each frame. The number of pan scan windows in the frame is implicitly determined as defined in section 8.9.1. If PS_PRESENT == 1, then the syntax elements PS_HOFFSET, PS_VOFFSET, PS_WIDTH and PS_HEIGHT shall also be present for each pan scan window in the frame. If PS_PRESENT == 0, then the syntax elements PS_HOFFSET, PS_VOFFSET, PS_WIDTH and

PS_HEIGHT shall not be present. See section 8.9 for a description of pan scan and the definition of the four syntax elements below.

7.1.1.21 Pan Scan Window Horizontal Offset (PS_HOFFSET) (18 bits)

PS_HOFFSET is an 18 bit syntax element that shall be present in all advanced profile progressive picture headers if the picture header syntax element PS_PRESENT == 1. This syntax element is defined in 8.9.2.

7.1.1.22 Pan Scan Window Vertical Offset (PS_VOFFSET) (18 bits)

PS_VOFFSET is an 18 bit syntax element that shall be present in all advanced profile progressive picture headers if the picture header syntax element PS_PRESENT == 1. This syntax element is defined in 8.9.2.

7.1.1.23 Pan Scan Window Width (PS_WIDTH) (14 bits)

PS_WIDTH is a 14 bit syntax element present that shall be present in all advanced profile progressive picture headers if the picture header syntax element PS_PRESENT == 1. This syntax element is defined in 8.9.2.

7.1.1.24 Pan Scan Window Height (PS_HEIGHT) (14 bits)

PS_HEIGHT is a 14 bit syntax element that shall be present in all advanced profile progressive picture headers if the picture header syntax element PS_PRESENT == 1. This syntax element is defined in 8.9.2.

7.1.1.25 Rounding Control Bit (RNDCTRL)(1 bit)

RNDCTRL is a 1 bit syntax element that shall be present in all advanced profile picture headers. The flag is used to indicate the type of rounding used for the current frame. If RNDCTRL == 1, the parameter *RND* which controls rounding shall be set to 1. Otherwise, *RND* shall be set to 0. In I and BI pictures, RNDCTRL shall be equal to 0. See Section 8.3.7 for more details on the effect of *RND* on rounding.

7.1.1.26 UV Sampling Format (UVSAMP)(1 bit)

UVSAMP is a 1 bit syntax element that shall only be present in all advanced profile picture headers, when the sequence level field INTERLACE == 1. If UVSAMP == 1, then progressive subsampling of the color-difference is used. If UVSAMP == 0, interlace subsampling of the color-difference is used. This syntax element does not affect decoding of the bitstream.

7.1.1.27 Post Processing (POSTPROC)(2 bits)

POSTPROC is a 2-bit syntax element that is present in all pictures in advanced profile when the sequence level flag POSTPROCFLAG == 1. This element is not required for the decoding process, but may be used by the display process. The four post-processing mode indicators shall be as defined in Table 41.

Table 42: POSTPROC code table

POSTPROC	Post processing Indication
00b	No Post Processing
01b	De-blocking
10b	De-ringing
11b	De-blocking && De-ringing

Note: It is desirable that decoders perform the post processing steps as indicated by the values above, and when post processing is done, the algorithms of Annex H be used. If both post-processing steps are performed, it is preferred that de-blocking is performed before de-ringing.

7.1.1.28 AC Prediction (ACPRED)(Variable size)

ACPRED is a bitplane coded syntax element that shall be present in all advanced profile I and BI pictures. ACPRED is used to indicate the AC prediction status for each macroblock in the picture. See section 7.2 for a description of the bitplane coding. See section 8.1.3.7 for a description of AC prediction.

7.1.1.29 Conditional Overlap Flag (CONDOVER) (Variable size)

CONDOVER is a variable-sized syntax element that shall be present only in advanced profile I pictures, and only when OVERLAP == 1, and PQUANT is less than or equal to 8 (regardless of HALFQP). CONDOVER may take the values of 0b, or 10b, or 11b. For the meaning of these values, and how CONDOVER affects overlap smoothing in advanced profile, see section 8.5.2 (Rules 4c. 4d. and 4e).

7.1.1.30 Conditional Overlap Macroblock Pattern Flags (OVERFLAGS) (Variable size)

OVERFLAGS is a bitplane coded syntax element that shall be present only in advanced profile I pictures, and only when CONDOVER has the binary value 11b. See section 8.5.2 for a description of how OVERFLAGS affects over-loop smoothing.

7.1.1.31 Macroblock Quantization (VOPDQUANT) (Variable size)

The VOPDQUANT syntax element shall be made up of several bitstream syntax elements as shown in Figure 21. VOPDQUANT shall be present in Progressive P and B pictures, and in advanced profile I pictures, when the sequence header syntax element DQUANT is nonzero.

The syntax of VOPDQUANT is dependent on the value of DQUANT. The syntax of VOPDQUANT is defined in Table 24. The macroblock quantizer step size (MQUANT) of the individual macroblocks in the picture is modified as specified below:

Case 1: DQUANT == 1.

There are four possibilities in this case:

1. Those macroblocks located on the picture edge boundary shall be quantized with a second quantization step size (ALTPQUANT), while all other macroblocks shall be quantized with the frame quantization step size (PQUANT).
2. Two adjacent edges are signaled (see Table 45), and those macroblocks located on the two picture edges shall be quantized with ALTPQUANT, while the rest of the macroblocks shall be quantized with PQUANT.
3. One edge only is signaled (see Table 44) and those macroblocks located on the picture edge are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.
4. Every single macroblock may be quantized differently. In this case, it will be indicated whether each macroblock may be selected from only two quantization steps (PQUANT or ALTPQUANT), or whether each macroblock may be arbitrarily quantized using any step size.

Case 2: DQUANT == 2.

1. The macroblocks located on the picture edge boundary shall be quantized with ALTPQUANT while the rest of the macroblocks shall be quantized with PQUANT.

The VOPDQUANT syntax elements are defined as follows:

7.1.1.31.1 Differential Quantizer Frame (DQUANTFRM) (1 bit)

The DQUANTFRM is a 1-bit syntax element that shall be present only when DQUANT ==1. If DQUANTFRM == 0, then the current picture shall only be quantized with PQUANT. The decoder shall use the default value of zero for DQUANTFRM if DQUANT != 1.

7.1.1.31.2 Differential Quantizer Profile (DQPROFILE) (2 bits)

The DQPROFILE is a 2-bit syntax element that shall be present only when DQUANT == 1 and DQUANTFRM == 1. It shall specify where it is allowable to change quantization step sizes within the current picture as defined in Table 43.

Table 43: Macroblock Quantization Profile (DQPROFILE) Code Table

DQPROFILE FLC	Location
00b	All four Edges
01b	Double Edge
10b	Single Edge
11b	All Macroblocks

7.1.1.31.3 Differential Quantizer Single Boundary Edge (DQSBEDGE) (2 bits)

The DQSBEDGE is a 2-bit syntax element that shall be present when DQPROFILE == 'Single Edge'. It shall specify which edge will be quantized with ALTPQUANT as defined in Table 44.

Table 44: Single Boundary Edge Selection (DQSBEDGE) Code Table

DQSBEDGE FLC	Boundary Edge
00b	Left
01b	Top
10b	Right
11b	Bottom

7.1.1.31.4 Differential Quantizer Double Boundary Edge (DQDBEDGE) (2 bits)

The DQDBEDGE is a 2-bit syntax element that shall be present only when DQPROFILE == 'Double Edge'. It shall specify which two edges will be quantized with ALTPQUANT as defined in Table 45.

Table 45: Double Boundary Edges Selection (DQDBEDGE) Code Table

DQDBEDGE FLC	Boundary Edges
00b	Left and Top
01b	Top and Right
10b	Right and Bottom
11b	Bottom and Left

7.1.1.31.5 Differential Quantizer Binary Level (DQBILEVEL) (1 bit)

The DQBILEVEL is a 1-bit syntax element that shall be present only when DQPROFILE == 'All Macroblocks' (see Table 43). DQBILEVEL determines the number of possible quantization step sizes which can be used by each macroblock in the frame. See section 7.1.3.4.

7.1.1.31.6 Picture Quantizer Differential (PQDIFF) (3 bits)

PQDIFF is a 3 bit syntax element that signals either the PQUANT differential or an escape code. If PQDIFF != 7, then PQDIFF signals the differential, and the ABSPQ syntax element shall not be present in the bitstream. In this case:

ALTPQUANT = PQUANT + PQDIFF + 1

If PQDIFF equals 7, then PQDIFF signals the escape code and the ABSPQ syntax element shall be present in the bitstream, and ALTPQUANT shall be decoded as:

ALTPQUANT = ABSPQ

Note: The value of ALTPQUANT has to be in the range of 1 to 31 for the bitstream to be valid.

7.1.1.31.7 Absolute Picture Quantizer (ABSPQ) (5 bits)

ABSPQ is a 5-bit syntax element that shall be present in the bitstream only if PQDIFF equals 7. In this case, ABSPQ shall directly signal the value of ALTPQUANT as described above.

7.1.1.32 Motion Vector Mode (MVMODE) (Variable size)

MVMODE is a variable-sized syntax element that shall be present in P and B picture headers. For P Pictures, the MVMODE syntax element shall signal one of four motion vector coding modes, or the intensity compensation mode. If the bitstream corresponds to the simple profile, the MVMODE syntax element shall not take the value corresponding to intensity compensation mode.

For P pictures, depending on the value of PQUANT, MVMODE shall be as defined in either Table 46 or Table 47.

Table 46: P Picture Low rate (PQUANT > 12) MVMODE code table

MVMODE VLC	Mode
1b	1-MV Half-pel bilinear
01b	1-MV
001b	1-MV Half-pel
0000b	Mixed-MV
0001b	Intensity Compensation

Table 47: P Picture High rate (PQUANT <= 12) MVMODE code table

MVMODE VLC	Mode
1b	1-MV
01b	Mixed-MV
001b	1-MV Half-pel
0000b	1-MV Half-pel bilinear
0001b	Intensity Compensation

For B pictures, MVMODE shall be as defined in Table 48.

Table 48: B Picture MVMODE code table

MVMODE VLC	Mode
1b	1-MV
0b	1-MV Half-pel Bilinear

Note: Intensity compensation cannot be signaled for B Pictures, and only two motion modes are valid

7.1.1.33 Motion Vector Mode 2(MVMODE2) (Variable size)

MVMODE2 is a variable-sized syntax element that shall be present in P pictures and only if the picture header syntax element MVMODE == 0001b (intensity compensation, see Table 46 and Table 47). Refer to section 8.3.4.3 for a description of motion vector mode and intensity compensation. Depending on the value of PQUANT, MVMODE2 shall be as defined in either Table 49 or Table 50.

Table 49: P Picture Low rate (PQUANT > 12) MVMODE2 code table

MVMODE 2 VLC	Mode
1b	1-MV Half-pel bilinear
01b	1-MV
001b	1-MV Half-pel
000b	Mixed-MV

Table 50: P Picture High rate (PQUANT <= 12) MVMODE2 code table

MVMODE 2 VLC	Mode
1b	1-MV
01b	Mixed-MV
001b	1-MV Half-pel
000b	1-MV Half-pel bilinear

7.1.1.34 Luma Scale (LUMSCALE)(6 bits)

LUMSCALE is a 6-bit syntax element that shall be present in P pictures and only if the picture header syntax element MVMODE == 0001b (intensity compensation). LUMSCALE shall be as defined in section 8.3.8.

7.1.1.35 Luma Shift (LUMSHIFT)(6 bits)

LUMSHIFT is a 6-bit syntax element that shall be present in P pictures and only if the picture header syntax element MVMODE == 0001b (intensity compensation). LUMSHIFT shall be as defined in section 8.3.8.

7.1.1.36 Motion Vector Type Bitplane (MVTYPEMB)(Variable size)

MVTYPEMB is a variable-sized syntax element that shall be present in P pictures if MVMODE or MVMODE2 indicates that “Mixed-MV” motion vector mode is used. The MVTYPEMB syntax element uses bitplane coding to signal the motion vector type (1- or 4-MV) for each macroblock in the frame. MVTYPEMB shall be defined as in section 8.3.4.3. Refer to section 8.7 for a description of the bitplane coding method. Refer to section 8.3.5.2 for a description of the motion vector decoding process.

7.1.1.37 Skipped Macroblock Bit Syntax Element (SKIPMB)(Variable size)

SKIPMB is a variable-sized syntax element that shall be present in P or B pictures. The SKIPMB syntax element signals the skipped macroblocks using a bitplane coding method. SKIPMB shall be as defined in section 8.3.4.4. Refer to section 8.7 for a description of the bitplane coding method.

7.1.1.38 Motion Vector Table (MVTAB) (2 bits)

MVTAB is a 2-bit syntax element that shall be present only in P and B frames. The MVTAB syntax element shall specify which of four tables is used to decode the motion vector data as defined in Table 51. Refer to sections 8.3.5.2.1 and 8.4.5.1 for a description of the motion vector decoding process.

Table 51: **MVTAB code-table**

MVTAB FLC	Motion Vector Differential VLC Table
00b	Table 0 (Table 246)
01b	Table 1 (Table 247)
10b	Table 2 (Table 248)
11b	Table 3 (Table 249)

The motion vector tables are listed in section 11.10.

7.1.1.39 Coded Block Pattern Table (CBPTAB) (2 bits)

CBPTAB is a 2-bit syntax element that shall be present in P and B frames. CBPTAB shall specify the table used to decode the CBPCY syntax element (described in section 7.1.3.1) for each coded macroblock in P and B pictures, as defined in Table 52. The CBP tables are listed in section 11.6. See sections 8.3.5.2 and 8.4.4.5 for a description of how CBPCY is used.

Table 52: CBPTAB table

CBPTAB FLC	Table used to decode CBPCY (P & B pictures)
00b	CBP Table 0 (Table 169)
01b	CBP Table 1 (Table 170)
10b	CBP Table 2 (Table 171)
11b	CBP Table 3 (Table 172)

7.1.1.40 Macroblock-level Transform Type Flag (TTMBF) (1 bit)

TTMBF is 1-bit syntax element that shall be present in P and B picture headers, and only if the sequence-level syntax element VSTRANSFORM == 1. If TTMBF == 1, then the TTFRM syntax element shall also be present in the picture layer. See sections 8.3.4.7 and 8.4.4.6 for a description.

7.1.1.41 Frame-level Transform Type (TTFRM) (2 bits)

TTFRM is a 2-bit syntax element that shall be present in P and B picture headers only if VSTRANSFORM == 1 and TTMBF == 1. The TTFRM syntax element shall be as defined in Table 53. See sections 8.3.4.8 and 8.4.4.7 for a description.

Table 53: **Transform type select code-table**

TTFRM FLC	Transform type
00b	8x8 Transform
01b	8x4 Transform
10b	4x8 Transform
11b	4x4 Transform

7.1.1.42 B Frame Direct Mode Macroblock Bit syntax element (DIRECTMB)(Variable size)

DIRECTMB is a variable-sized syntax element that shall only be present in B pictures. The DIRECTMB syntax element uses bitplane coding to specify the macroblocks in the B picture that are coded in 'Direct' mode. The DIRECTMB syntax element may also specify that the 'Direct' mode is signaled in raw mode (see section 7.1.3.12). Refer to section 8.7 for a description of the bitplane coding method.

7.1.2 Slice Layer

A *slice* represents one or more contiguous rows of macroblocks. A slice-layer is present only in the advanced profile. Even in the advanced profile, the slice layer is optional, and may be skipped by coding a picture as a single bitstream data unit. When a picture is coded in multiple Bitstream Data Units (BDUs), slices are used. A slice shall always begin at the first macroblock in a row and shall always end at the last macroblock in the same or another row. Thus a slice shall contain an integer number of complete rows. A slice shall always be byte-aligned and each slice shall be contained in a different BDU.

The beginning of a new slice is detected through a search for start-codes as defined in Annex E. The first BDU in a frame shall be preceded by the frame start code, and shall contain the frame layer syntax elements. The first BDU in the second field of an inter-lace field coded picture shall be preceded by the field start code, and shall contain the field layer syntax elements. The other BDUs in the picture shall be preceded by the slice start code, and contain the slice layer syntax elements.

When a new slice begins, motion vector predictors, predictors for AC and DC coefficients, and the predictors for quantization parameters shall be reset. In other words, with respect to prediction, the first row of macroblocks in the slice shall be considered to be the first row of macroblocks in the picture. This ensures that there is no inter-slice dependency in predictors.

Further, when slices are used, all bitplane information shall be carried in raw coding mode which ensures that each macroblock carries its own local information. For the purposes of deblocking, each slice shall be treated independently. In other words, the top and bottom macroblock rows of each slice are treated as if they are the top and macroblocks rows of the picture in the deblocking process. Thus, there shall be no loop-filtering across slices. No overlap smoothing shall be allowed across a macroblock boundary if the adjacent macroblocks belong to different slices. Thus, there is no overlap smoothing across different slices.

The Slice Layer shall be as defined in Figure 22. The syntax elements that make up the slice layer are defined in the following sections.

7.1.2.1 Slice Address (SLICE_ADDR)(9 bits)

SLICE_ADDR is a 9-bit syntax element. The row address of the first macroblock row in the slice shall be binary encoded in this syntax element. This syntax element may take the value from 1 to 511 as a binary value. The value 0 for this syntax element is SMPTE Reserved.

Note: The maximum picture size of 8192 pixels corresponds to a maximum of 512 macroblock rows. Further, the first macroblock row in the frame/field cannot be preceded by a slice header, and therefore the SLICE_ADDR syntax element does not take the value 0.

7.1.2.2 Picture Header Present Flag (PIC_HEADER_FLAG)(1 bit)

PIC_HEADER_FLAG is a 1-bit syntax element that shall be present in the slice header. If PIC_HEADER_FLAG == 0, then the picture header information shall not be repeated in the slice header. If the PIC_HEADER_FLAG == 1, the picture header information shall be repeated in the slice header.

7.1.3 Macroblock Layer

Data for each macroblock shall consist of a macroblock header followed by the block layer. Figure 23 – Figure 26, and Table 27 - Table 30 show the macroblock layer structure for I picture, P picture and B picture macroblocks. The elements that make up the macroblock layer are described in the following sections. Specified in square brackets are the types (intra, inter or both) in which the block elements occur.

7.1.3.1 Coded Block Pattern (CBPCY) (Variable size)[I, P,B]

CBPCY is a variable-sized syntax element that shall be present in all I and BI picture macroblocks, and may be present in P and B picture macroblocks. In I and BI pictures, CBPCY shall be decoded using the VLC table of section 11.5. In P and B pictures, CBPCY shall be decoded using the VLC table specified by the CBPTAB syntax element as described in section 7.1.1.39. The CBPCY tables for P and B pictures are defined in section 11.6. Section 8.1.2.1 describes the CBPCY syntax element in I picture macroblocks and section 8.3.5.5 describes the CBPCY syntax element in P picture and B picture macroblocks.

7.1.3.2 AC Prediction Flag (ACPRED)(1 bit)[I, P,B]

ACPRED is a 1-bit syntax element that shall be present in all I and BI picture macroblocks and in intra macroblocks in P pictures and B Pictures. In advanced profile I and BI pictures, ACPRED shall be present at the macroblock layer only if the raw mode is used to code the ACPRED bitplane. See section 8.3.5.1 for a description of the macroblock types. ACPRED shall also be present in a 4-MV macroblock in P pictures, only if at least one of the blocks in that macroblock is intra-coded, and if that block(s) has a non-zero predictor. (See section 8.3.6.1.3 for details on determining if a block has a non-zero predictor). ACPRED == 0 shall indicate that AC prediction was not used. ACPRED == 1 shall indicate that AC prediction was used. See section 8.1.2.2 for a description of the ACPRED syntax element in I pictures and section 8.3.6.1 for a description of the ACPRED syntax element in P and B pictures.

7.1.3.3 Conditional Overlap Macroblock Pattern Flag (OVERFLAGMB) (1 bit) [I]

OVERFLAGMB is a 1 bit syntax element that shall be present only in advanced profile I pictures, only when CONDOVER has the value 11b and only when the raw mode is chosen to encode the OVERFLAGS plane. In this case, one bit shall be present in the macroblock header to indicate whether or not to perform overlap filtering to edge pixels within the block and neighboring blocks. See section 8.5.2 for a description.

7.1.3.4 Macroblock Quantizer Differential (MQDIFF)(Variable size)[I,P,B]

MQDIFF is a variable-sized syntax element that shall be present in P and B pictures, and in advanced profile I pictures only if the picture layer syntax element DQPROFILE == ‘All Macroblocks’. The syntax depends on the DQBILEVEL syntax element as defined below.

If DQBILEVEL == 1, then MQDIFF shall be a 1 bit syntax element and the ABSMQ syntax element shall not be present in the bitstream. If MQDIFF == 0, then MQUANT = PQUANT (meaning that PQUANT shall be used as the quantization step size for the current macroblock). If MQDIFF == 1, then MQUANT = ALTPQUANT.

If DQBILEVEL == 0, then MQDIFF shall be a 3 bit syntax element. In this case MQDIFF shall decode either to an MQUANT differential or to an escape code as follows:

> If MQDIFF does not equal 7, then MQDIFF shall decode to the differential and the ABSMQ syntax element shall not be present in the bitstream. In this case:
>
> MQUANT = PQUANT + MQDIFF
>
> MQUANT shall be in the range of 1 to 31 for the bitstream to be valid. If MQDIFF equals 7, then the ABSMQ syntax element shall be present in the bitstream and MQUANT shall be decoded as:
>
> MQUANT = ABSMQ

Figure 29 defines this computation of MQUANT.

```
if (DQPROFILE == 'all macroblocks') {
    if (DQBILEVEL) {
        //Decode 1 bit flag MQDIFF
        if (MQDIFF == 0)
            MQUANT = PQUANT
        else
            MQUANT = ALTPQUANT
    }
    else {
        //Decode 3 bit syntax element MQDIFF
        if (MQDIFF != 7) {
            MQUANT = PQUANT + MQDIFF; // Note MQUANT has be in the range 1 to 31
        }
        else {
            //Decode 5 bit syntax element ABSMQ
            MQUANT = ABSMQ;   // Note MQUANT has to be in the range 1 to 31
        }
    }
}
```

Figure 29: Calculation of MQUANT when DQPROFILE == 'all macroblocks'

7.1.3.5 Absolute Macroblock Quantizer Scale (ABSMQ)(5 bits)[I,P,B]

ABSMQ is a 5 bit syntax element that shall be present in the bitstream only if MQDIFF == 7. In this case, ABSMQ is decoded to directly derive the value of MQUANT as defined in Figure 29 above (i.e. MQUANT == ABSMQ).

7.1.3.6 MV Mode Bit (MVMODEBIT)(1 bit)[P]

MVMODEBIT is a 1-bit syntax element that shall be present in P frame macroblocks if the picture is coded in 'Mixed-MV' Mode, and only when the raw mode is chosen to code the MVTYPEMB bitplane (see section 7.1.1.36). The definition of raw mode is where the IMODE Coding Mode is set to 'Raw' as defined in section 7.2.2 (Table 69). If MVMODEBIT == 0, then the macroblock shall be coded in 1-MV mode, and if MVMODEBIT == 1, then the macroblock shall be coded in 4-MV mode.

7.1.3.7 Skip MB Bit (SKIPMBBIT)(1 bit)[P,B]

SKIPMBBIT is a 1-bit syntax element that shall be present in P and B frame macroblocks if the raw mode is used to code the SKIPMB bitplane (see section 7.1.1.32). For definition of raw mode, see section 7.2.2 (Table 69). If SKIPMBBIT == 1, then the macroblock shall be skipped. If SKIPMBBIT == 0, the macroblock shall not be skipped. See sections 8.3.4.4 and 8.4.4.3 for details on skipped macroblocks.

7.1.3.8 Motion Vector Data (MVDATA)(Variable size)[P]

MVDATA is a variable sized syntax element that may be present in P picture macroblocks. This syntax element decodes to the motion vector(s) for the macroblock. The table used to decode this syntax element is specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.38. See section 8.3.5.2.1 for a description of the motion vector decode process.

7.1.3.9 Hybrid Motion Vector Prediction (HYBRIDPRED)(1 bit)[P]

HYBRIDPRED is a 1-bit syntax element per motion vector that may be present in P picture macroblocks. Section 8.3.5.3.5 describes how HYBRIDPRED is used in the decoding process.

7.1.3.10 MB-level Transform Type (TTMB)(Variable size)[P,B]

The TTMB syntax element is a variable syntax element and shall be present in P and B picture inter-coded macroblocks, if the picture layer syntax element TTMBF == 0, and if at least one of the blocks has non-zero coefficients (i.e. at least one coded block in the macroblock). The TTMB syntax element shall specify the transform type, the signal level and the subblock pattern. If the signal level specifies macroblock mode, the transform type decoded from the TTMB syntax element shall be used to decode all coded blocks in the macroblock. If the signal level signals block mode, then the transform type decoded from the TTMB syntax element shall be used to decode the first coded block in the macroblock. The transform type of the remaining blocks shall be decoded at the block level. If the transform type is 8x4 or 4x8, then the subblock pattern shall indicate the subblock pattern of the first block. The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

The table used to decode the TTMB syntax element depends on the value of PQUANT. For PQUANT less than or equal to 4, Table 54 shall be used. For PQUANT greater than 4 and less than or equal to 12, Table 55 shall be used. For PQUANT greater than 12, Table 56 shall be used.

Table 54: High Rate (PQUANT < 5) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
11b	8x8	Block	NA
101110b	8x4	Block	Bottom
1011111b	8x4	Block	Top
00b	8x4	Block	Both
10110b	4x8	Block	Right
10101b	4x8	Block	Left
01b	4x8	Block	Both
100b	4x4	Block	NA
10100b	8x8	Macroblock	NA
1011110001b	8x4	Macroblock	Bottom
101111001b	8x4	Macroblock	Top
101111011b	8x4	Macroblock	Both
1011111000000b	4x8	Macroblock	Right
1011111000001b	4x8	Macroblock	Left
10111100001b	4x8	Macroblock	Both
101111010b	4x4	Macroblock	NA

Table 55: Medium Rate (5 <= PQUANT < 13) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
110b	8x8	Block	NA
0110b	8x4	Block	Bottom
0011b	8x4	Block	Top
0111b	8x4	Block	Both
1111b	4x8	Block	Right
1110b	4x8	Block	Left
000b	4x8	Block	Both
010b	4x4	Block	NA
10b	8x8	Macroblock	NA
0010100b	8x4	Macroblock	Bottom
0010001b	8x4	Macroblock	Top
001011b	8x4	Macroblock	Both
001001b	4x8	Macroblock	Right
00100001b	4x8	Macroblock	Left
0010101b	4x8	Macroblock	Both
00100000b	4x4	Macroblock	NA

Table 56: Low Rate (PQUANT >= 13) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
110b	8x8	Block	NA
000b	8x4	Block	Bottom
1110b	8x4	Block	Top
00101b	8x4	Block	Both
010b	4x8	Block	Right
011b	4x8	Block	Left
0011b	4x8	Block	Both
1111b	4x4	Block	NA
10b	8x8	Macroblock	NA
0010000001b	8x4	Macroblock	Bottom
00100001b	8x4	Macroblock	Top
001001b	8x4	Macroblock	Both
00100000001b	4x8	Macroblock	Right
001000001b	4x8	Macroblock	Left
0010001b	4x8	Macroblock	Both
00100000000b	4x4	Macroblock	NA

7.1.3.11 Block-level Motion Vector Data (BLKMVDATA)(Variable size)[P]

BLKMVDATA is a variable-sized syntax element that that may be present in P picture macroblocks that are coded in 4-MV mode. This syntax element decodes to the motion information for the block. The table used to decode this syntax element shall be specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.38. See section 8.3.5.2.1 for a description of when the BLKMVDATA syntax element is present and how it is used.

7.1.3.12 Direct B Frame Coding Mode (DIRECTBBIT)(1 bit)[B]

DIRECTBBIT is a 1-bit syntax element that shall be present in B frame macroblocks if the frame level syntax element DIRECTMB (see section 7.1.1.42) indicates that raw mode is used (see section 7.2.2, Table 69). If DIRECTBBIT == 1, then the macroblock shall be decoded using ‘Direct’ mode. See section 8.4.5.3 for details on ‘Direct’ mode.

7.1.3.13 B Macroblock Motion Vector 1 (BMV1)(Variable size)[B]

BMV1 is a variable sized syntax element that may be present in B picture macroblocks. This syntax element decodes to the first motion vector for the macroblock. The table used to decode this syntax element shall be specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.38. The decoding procedure for BMV1 is described in section 8.4.5.5.

7.1.3.14 B Macroblock Motion Prediction Type (BMVTYPE)(Variable size)[B]

BMVTYPE is a variable sized syntax element that may be present in B frame macroblocks and indicates whether the macroblock uses forward, backward or interpolated prediction. The value of BFRACTION (in the picture header, see section 7.1.1.14) along with BMVTYPE shall determine which type is used as defined in Table 57.

Table 57: B Frame Motion Prediction Type

BMVTYPE VLC	Motion Prediction Type	
	Fraction (derived from BFRACTION) >= 1/2	Fraction (derived from BFRACTION) < 1/2
0b	Backward	Forward
10b	Forward	Backward
11b	Interpolated	Interpolated

7.1.3.15 B Macroblock Motion Vector 2 (BMV2)(Variable size)[B]

BMV2 is a variable sized syntax element that shall be present in B picture macroblocks if the Interpolated mode is used. This syntax element decodes to the forward motion vector for the macroblock. In this case, the backward motion vector is decoded from BMV1. If BMV2 is present, it shall not indicate that the macroblock is intra-coded (i.e. decoding of BMV2 sets the corresponding intra_flag to 0). The table used to decode this syntax element shall be specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.38. The decoding procedure for BMV2 is described in section 8.4.5.5and 8.3.5.2.1

7.1.4 Block Layer

Figure 27 and Figure 28 show the block layer syntax elements for intra and inter-coded blocks respectively. The elements that make up the block layer are described in the following sections. Specified in square brackets are the types (intra, inter or both) in which the block elements occur.

7.1.4.1 Transform DC Coefficient (DCCOEF)(Variable size)[intra]

The DCCOEF is a variable-sized syntax element that shall only be present in intra-coded blocks and decodes to either the transform DC differential or the escape code. Refer to section 8.1.3.1 for a definition of the Transform DC decoding process. One of two code tables is used to decode the DCCOEF and the chosen table shall be specified by the TRANSDCTAB syntax element in the picture header as described in section 8.1.1.2. Section 11.7 lists the DC tables.

7.1.4.2 Transform DC Coefficient (DCCOEFESC)(variable size)[intra]

The DCCOEFESC syntax element shall only be present in intra-coded blocks and only if DCCOEF decodes to the escape code (refer to 7.1.4.7). The size of DCCOEFESC syntax element may be 8, 9 or 10 bits, depending on the quantization step size of the block. Refer to section 8.1.3.1 for a description of the Transform DC decoding process.

7.1.4.3 Transform DC Coefficient Extension for Quant1 (DCCOEF_EXTQUANT1)(2 bit)[intra]

The DCCOEF_EXTQUANT1 is a 2-bit syntax element that shall only be present in intra-coded blocks, and only if DCCOEF is decodes to a non-zero and non-escape code value, and if the quantizer step size for the block has the value 1. This syntax element shall be used in conjunction with DCCOEF to determine the value of the DC differential when the quantizer step size takes the value 1. Refer to section 8.1.3.1 for a description of the Transform DC decoding process.

7.1.4.4 Transform DC Coefficient Extension for Quant2 (DCCOEF_EXTQUANT2)(1 bit)[intra]

The DCCOEF_EXTQUANT2 is a 1-bit syntax element that shall only be present in intra-coded blocks, and only if DCCOEF decodes to a non-zero and non-escape code value, and if the quantizer step size for the block has the value 2. This syntax element shall be used in conjunction with DCCOEF to determine the value of the DC differential when the quantizer step size takes the value 2. Refer to section 8.1.3.1 for a description of the Transform DC decoding process.

7.1.4.5 Transform DC Sign (DCSIGN)(1 bit)[intra]

DCSIGN is a one-bit syntax element that indicates the sign of the DC differential. It shall only be present if DCCOEF decodes to a non-zero value. If DCSIGN == 0, then the DC differential is positive. If DCSIGN == 1, then the DC differential is negative.

7.1.4.6 Transform AC Coefficient 1 (ACCOEF1)(Variable size)[both]

ACCOEF1 is a variable-sized syntax element that may be present in both intra and inter blocks. This is a variable-length codeword that shall decode to the run, level and last_flag for each non-zero AC coefficient. Refer to section 8.1.3.4 for a description of the Transform AC decoding process. One of three code tables shall be used to decode ACCOEF1. The table is signaled in the picture header by the TRANSACFRM or TRANSACFRM2 as defined in sections 8.1.3.4 and 8.3.4.9. Section 11.8 lists the AC code tables.

7.1.4.7 Transform AC Escape Decoding Mode (ESCMODE)(Variable size)[both]

ESCMODE is a variable-sized syntax element that may be present in both intra and inter blocks. It shall only be present if ACCOEF1 decodes to the escape code. ESCMODE shall specify which of three escape decoding methods are used to decode the AC coefficient. Table 58 shows the code-table used to decode the escape modes.

Table 58: **AC escape decoding mode code-table**

ESCMODE VLC	AC Escape Decoding Mode
1b	Mode 1
01b	Mode 2
00b	Mode 3

If Mode 1 or Mode 2 decoding mode is specified, then the bitstream shall contain the ACCOEF2 element as described in section 7.1.4.8. If Mode 3 decoding mode is specified, then the bitstream shall contain the ESCLR, ESCRUN, ESCLVL and LVLSIGN2 syntax elements and may contain the ESCLVLSZ and ESCRUNSZ elements, as described in sections 7.1.4.9 - 7.1.4.11.

7.1.4.8 Transform AC Coefficient 2 (ACCOEF2)(Variable size)[both]

ACCOEF2 is a variable-sized syntax element that may be present in both intra and inter blocks. It shall only be present if ACCOEF1 decodes to the escape code and if the ESCMODE syntax element (described in section 7.1.4.7) specifies AC decoding Escape Mode 1 or 2 (refer to section 8.1.3.4 for a description of the Transform AC decoding process). One of three code tables shall be used to decode ACCOEF2. The table is signaled in the picture header by the TRANSACFRM or TRANSACFRM2 as described in sections 8.1.3.4 and 8.3.4.9. Section 11.8 lists the AC tables.

7.1.4.9 Escape Mode 3 Last Run (ESCLR)(1 bit)[both]

ESCLR is a 1-bit syntax element that may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding Escape Mode 3. ESCLR shall specify whether this coefficient is the last non-zero coefficient in the block. If ESCLR == 1, then this coefficient shall be the last non-zero coefficient. If ESCLR == 0, then this coefficient shall not be the last non-zero coefficient.

7.1.4.10 Escape Mode 3 Level Size (ESCLVLSZ)(Variable size)[both]

ESCLVLSZ is a variable-sized syntax element that may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding Escape Mode 3, and if this is the first time Mode 3 has been signaled within the current picture (in other words, all subsequent instances of Escape Mode 3 coding within this picture do not have this syntax element). ESCLVLSZ shall specify the codeword size for the Mode 3 escape-coded level values for the entire picture. If there are multiple slices in a picture, ESCLVLSZ shall only be present in a slice if ESCMODE specifies AC decoding Escape Mode 3 and if this is the first time Mode 3 has been signaled within that slice. In this case, ESCLVLSZ shall have the same value in all the slices of that picture.

Table 59 (conservative table) shall be used to decode ESCLVLSZ when PQUANT is between 1 and 7, both values inclusive, or if VOPDQUANT syntax element is present in that picture and the quantizer can vary in the picture as determined by the pseudo-code in Table 24. Table 60 (efficient table) shall be used when PQUANT is 8 and higher, and if VOPDQUANT syntax element is absent in that picture or if the same quantizer is used in the picture as determined by the pseudo-code in Table 24.

Note: The conservative table covers the widest range of possible transform values, whereas the efficient table covers a limited subset and is therefore used when the values may be guaranteed to be within the available range.

Table 59: Escape Mode 3 level codeword size conservative code-table (used for 1 <= PQUANT <= 7 or if VOPDQUANT is present and quantizer can vary in picture)

1 <= PQUANT <= 7	
ESCLVLSZ VLC	Level codeword size
001b	1
010b	2
011b	3
100b	4
101b	5
110b	6
111b	7
00000b	8
00001b	9
00010b	10
00011b	11

Table 60: **Escape Mode 3 level codeword size efficient code-table (used for 8 <= PQUANT <= 31, and if VOPDQUANT is absent or if the same quantizer is used in the picture)**

8 <= PQUANT <= 31	
ESCLVLSZ VLC	Level codeword size
1b	2
01b	3
001b	4
0001b	5
00001b	6
000001b	7
000000b	8

7.1.4.11 Escape Mode 3 Run Size (ESCRUNSZ)(2 bits)[both]

ESCRUNSZ is a 2-bit syntax element that may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding escape mode 3, and if this is the first time escape mode 3 is signaled within the frame. ESCRUNSZ shall specify the codeword size for the mode 3 escape-coded run values for the entire frame. If there are multiple slices in a frame, ESCRUNSZ shall be present in a slice if this is the first time mode 3 has been signaled within that slice, and ESCRUNSZ shall have the same value in all the slices of that frame. The run codeword size shall be decoded according to Table 61:

Table 61: **Escape Mode 3 run codeword size code-table**

ESCRUNSZ FLC	Run codeword size
00b	3
01b	4
10b	5
11b	6

7.1.4.12 Escape Mode 3 Run (ESCRUN)(Variable size)[both]

ESCRUN may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding Escape Mode 3. The size of the ESCRUN codeword is fixed throughout the picture, with the size being specified in the ESCRUNSZ syntax element described in section 7.1.4.11. ESCRUN shall directly decode to the run value for the coefficient. For example, if the size (from ESCRUNSZ) is 4 bits and the value is 0101b, then the run is decoded as 5.

7.1.4.13 Escape Mode 3 Level Sign (LVLSGN2)(1 bit)[both]

LVLSGN2 is a 1-bit syntax element that may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding Escape Mode 3. LVLSGN2 specifies the sign of the decoded level value (ESCLVL). If LVLSGN2 == 0, then the level shall be positive. If LVLSGN2 == 1, then the level shall be negative.

7.1.4.14 Escape Mode 3 Level (ESCLVL)(Variable size)[both]

ESCLVL may be present in both intra and inter blocks. It shall only be present if ESCMODE specifies AC decoding Escape Mode 3. The size of the ESCLVL codeword is fixed throughout the frame, with the size being specified in the ESCLVLSZ syntax element described in section 7.1.4.10. ESCLVL shall directly decode to the level value for the coefficient. For example, if the size (from ESCLVLSZ) is 3 bits and the value is 110b, then the run is decoded as 6.

7.1.4.15 Transform AC Level Sign (LVLSIGN)(1 bit)[both]

LVLSIGN may be present in both intra and inter blocks. It shall always be present unless ESCMODE specifies AC decoding Escape Mode 3. LVLSIGN is a one-bit value that shall specify the sign of the AC level. Refer to section 8.1.3.4 for a description of the Transform AC decoding process. If LVLSIGN == 0, then the level shall be positive. If LVLSIGN == 1, then the level shall be negative.

7.1.4.16 Block-level Transform Type (TTBLK)(Variable size)[inter]

The TTBLK syntax element shall be present only in inter-coded blocks and only if the macroblock level syntax element TTMB (see section 7.1.3.10) indicates that the signaling level is 'Block.' The 8x8 error blocks may be transformed using an 8x8 Transform, two 8x4 Transforms, two 4x8 Transforms or four 4x4 Transforms. The TTBLK syntax element shall decode to the transform type for the inter-coded block as well as the subblock pattern if the transform type is 8x4 or 4x8. The table used to decode the TTBLK syntax element depends on the value of PQUANT.

If PQUANT <= 4, then Table 62 shall be used.

If PQUANT > 4 && PQUANT <= 12, then Table 63 shall be used.

If PQUANT > 12, then Table 64 shall be used.

The TTBLK syntax element shall not be present for the first inter-coded block in each macroblock since the transform type and subblock pattern decoded in TTMB is used for the first inter-coded block. TTBLK shall be present for each inter-coded block after the first. The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

Table 62: High Rate (PQUANT < 5) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
00b	8x4	Both
01b	4x8	Both
11b	8x8	NA
101b	4x4	NA
10000b	8x4	Top
10001b	8x4	Bottom
10010b	4x8	Right
10011b	4x8	Left

Table 63: Medium Rate (5 =< PQUANT < 13) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
11b	8x8	NA
000b	4x8	Right
001b	4x8	Left
010b	4x4	NA
011b	8x4	Both
101b	4x8	Both
1000b	8x4	Bottom
1001b	8x4	Top

Table 64: Low Rate (PQUANT >= 13) TTBLK VLC Table

TTBLK VLC	Transform Type	Subblock Pattern
01b	8x8	NA
000b	4x8	Both
001b	4x4	NA
100b	8x4	Bottom
110b	4x8	Right
111b	4x8	Left
1010b	8x4	Both
1011b	8x4	Top

7.1.4.17 Transform sub-block pattern (SUBBLKPAT)(Variable size)[inter]

The SUBBLKPAT syntax element shall only be present in inter-coded blocks and only if the transform type for the block is 4x4, or if the transform type for the block is 8x4, 4x8 and the two conditions described below are satisfied:

a) if the transform type is specified at the frame level,
b) if the transform type is specified at the macroblock level and the block is not the first coded inter block in the macroblock.

For 4x4 transform types, the SUBBLKPAT syntax element shall always be present, and it shall specify which of the 4 4x4 subblocks have at least one non-zero coefficient.

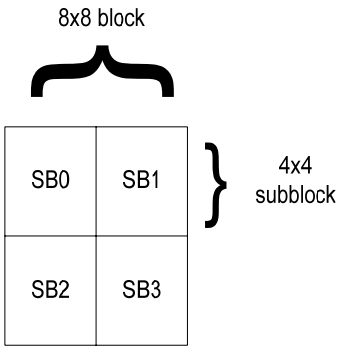


Figure 30: 4x4 Subblocks

The subblock pattern shall be decoded as a 4 bit field where each bit indicates whether the corresponding subblock contains at least one non-zero coefficient. Figure 30 shows the labeling of the 4 subblocks that make up an 8x8 block. The subblock pattern shall be decoded as follows:

Subblock pattern = 8 * SB0 + 4 * SB1 + 2 * SB2 + SB3

Where:

SBx = 0 if the corresponding subblock does not contain any non-zero coefficients and

SBx = 1 if the corresponding subblock contains at least one non-zero coefficient.

The following tables show the VLC codewords used to decode the subblock pattern. The table used depends on the value of PQUANT.

If PQUANT <= 4, then Table 65 shall be used.

If PQUANT > 4 && PQUANT <= 12, then Table 66 shall be used.

If PQUANT > 12, then Table 67 shall be used.

Table 65: High Rate (PQUANT < 5) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
1b	15	01010	8
0000b	11	01011	4
0001b	13	01100	2
0010b	7	01110	1
00110b	12	01111	14
00111b	3	011010	6
01000b	10	011011	9
01001b	5		

Table 66: Medium Rate (5 =< PQUANT < 13) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
01b	15	1111	4
000b	2	00100	6
0011b	12	00101	9
1000b	3	10110	14
1001b	10	10111	7
1010b	5	11000	13
1101b	8	11001	11
1110b	1		

Table 67: Low Rate (PQUANT >= 13) SUBBLKPAT VLC Table

SUBBLKPAT VLC	Subblock Pattern	SUBBLKPAT VLC	Subblock Pattern
010b	4	1111	15
011b	8	00000	6
101b	1	00001	9
110b	2	10010	14
0001b	12	10011	13
0010b	3	11100	7
0011b	10	11101	11
1000b	5		

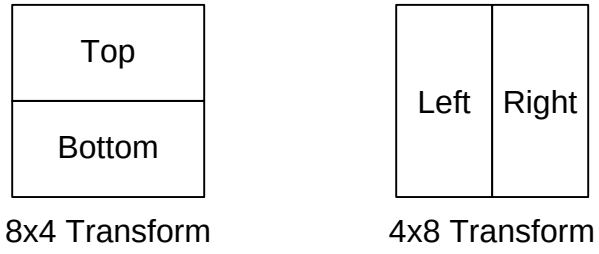


Figure 31: 8x4 and 4x8 Subblocks

For 8x4 or 4x8 transform types (shown in Figure 31), the SUBBLKPAT syntax element specifies which of the two sub-blocks have at least one non-zero coefficient. For 8x4 and 4x8 transform types, SUBBLKPAT shall be present only if either a) the transform type is specified at the frame level (TTMBF == 1), or b) if the transform type is specified at the macroblock level and if the block is not the first coded inter block in the macroblock. If the transform type is specified at the macroblock level, the coded block pattern for the first coded block in the macroblock is decoded from the TTMB syntax element. SUBBLKPAT shall be decoded according to Table 68 (an X indicates that the sub-block contains at least one non-zero coefficient):

Table 68: **8x4 and 4x8 Transform sub-block pattern code-table for Progressive pictures**

SUBBLKPAT VLC	8x4 Sub-block pattern		4x8 Sub-block pattern	
	Top	Bottom	Left	Right
10b		X		X
0b	X	X	X	X
11b	X		X	

7.2 Bitplane Coding Syntax

Various frame-level syntax elements use a bitplane coding scheme to indicate the status of the macroblocks that make up the frame. For example, in P and B frames, the presence of skipped macroblocks is signaled with a bit set to 1 and the presence of a non-skipped macroblock is signaled with a bit set to 0. These bits are coded as a frame-level bitplane. The following diagram shows the elements that make up the bitplane. Refer to section 8.7 for a definition of bitplane coding.

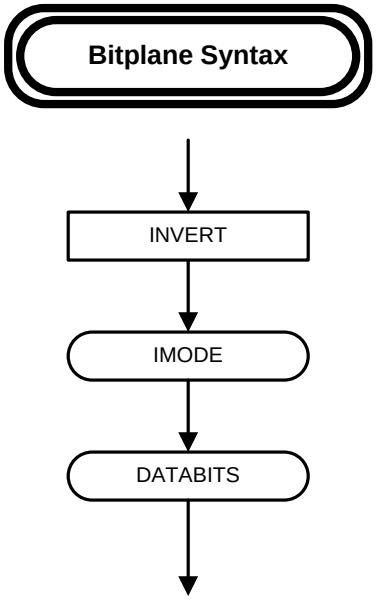


Figure 32: Syntax diagram for the bitplane coding

7.2.1 Invert Flag (INVERT) (1-bit)

INVERT is a 1-bit syntax element. Refer to section 8.7.1 for a description of how the INVERT value is used in decoding the bitplane.

7.2.2 Coding Mode (IMODE) (variable)

IMODE is a variable length syntax element that specifies the coding mode used to decode the bitplane. The IMODE syntax element shall be as defined in Table 69. Refer to section 8.7.2 for a description of how the IMODE value is used in decoding the bitplane.

Table 69: **IMODE VLC Code table**

IMODE VLC	Coding Mode
10b	Norm-2
11b	Norm-6
010b	Rowskip
011b	Colskip
001b	Diff-2
0001b	Diff-6
0000b	Raw

7.2.3 Bitplane Coding Bits (DATABITS) (variable)

DATABITS is a variable sized syntax element that decodes the bitplane. The method used to decode the bitplane shall be determined by the value of IMODE. Refer to section 8.7.3 for a description the different coding methods.

8 Progressive Decoding Process

This section describes the decoding processes required for decoding progressive I pictures (section 8.1), progressive BI pictures (section 8.2), progressive P pictures (section 8.3) and progressive B pictures (section 8.4) for all profiles.

This section also defines the processes that are common to progressive coded picture types including the overlapped transform decoding process (section 8.5), the in-loop deblock filtering process (section 8.6), the bitplane coding process (section 8.7), the sync markers for simple/main profiles (section 8.8), and the pan-scan decoding process for advanced profile (section 8.9).

8.1 Progressive I Frame Picture Decoding

The following sections describe the processes for decoding progressive I frame pictures.

Section 8.1.1 defines the picture layer decoding, and section 8.1.2 defines the macroblock and section 8.1.3 defines block layer decoding.

8.1.1 Progressive I Frame Picture Layer Decode

Figure 13 defines the elements that make up the I Picture layer header for simple and main profiles. Figure 15 defines the elements that make up the I Frame Picture layer header for the advanced profile. The following sections provide extra detail for those elements that are not self-explanatory.

8.1.1.1 Frame-level Transform AC Table Index (TRANSACFRM and TRANSACFRM2)

TRANSACFRM (**7.1.1.11**) and TRANSACFRM2 (**7.1.1.12**) are variable-length syntax elements that are present in the picture layer. The TRANSACFRM syntax element shall provide the index that selects the coding set used to decode the Transform AC coefficients for the C_b, C_r blocks. The TRANSACFRM2 syntax element shall provide the index that selects the coding set used to decode the Transform AC coefficients for the Y blocks. Table 39 defines the Transform AC Coding Set Index code table that shall be used to decode the TRANSACFRM and TRANSACFRM2 syntax elements. Refer to section 8.1.3.4 for a description of AC coefficient decoding.

8.1.1.2 Intra Transform DC Table (TRANSDCTAB)

TRANSDCTAB (**7.1.1.13**) is a one-bit syntax element that shall specify which of two tables is used to decode the Transform DC coefficients in intra-coded blocks. If TRANSDCTAB == 0, then the low motion tables of Section 11.7.1 shall be used. If TRANSDCTAB == 1, then the high motion tables of Section 11.7.2 shall be used.

8.1.1.3 Picture Resolution Index (RESPIC)

The RESPIC syntax element in I pictures shall specify the scaling factor of the decoded I picture relative to a full resolution frame. The decoded picture may be full resolution or half the original resolution in either the horizontal or vertical dimensions or half resolution in both dimensions. The scaling factor shall be decoded from the RESPIC syntax element according to Table 38.

Note: RESPIC is not used in advanced profile.

The resolution decoded in the I picture RESPIC syntax element shall apply to all subsequent P pictures until the next I picture. In other words, all P pictures are coded at the same resolution as the closest preceding I picture. The RESPIC syntax element that is present in a P picture header shall carry the same value as the RESPIC syntax element of the closest temporally preceding I frame.

The pseudo-code of Figure 12 specifies how the new frame dimensions shall be calculated if a down-sampled resolution is indicated.

```
X = 16 * ((CodedWidth + 15) / 16)
Y = 16 * ((CodedHeight + 15) / 16)
x = new horizontal resolution
y = new vertical resolution
hscale = horizontal scaling factor (0 = full resolution, 1= half resolution)
vscale = vertical scaling factor (0 = full resolution, 1= half resolution)

x = X
y = Y
if (hscale == 1)
{
      x = X / 2
      if ((x & 15) != 0)
            x = x + 16 – (x & 15)
}
if (vscale == 1)
{
      y = Y / 2
      if ((y & 15) != 0)
            y = y + 16 – (y & 15)
}
```

Figure 33: Calculation of Frame Dimensions in Multires Down-sampling Pseudo-code

If the decoded picture is one of the sub-sampled resolutions, then it may be up-sampled to full resolution prior to display according to Annex B.

8.1.1.4 Range Reduction Frame - I Frame (RANGEREDFRM)

The RANGEREDFRM shall only be present when RANGERED == 1 at the sequence level.

When RANGEREDFRM == 1 for the current I picture, the current decoded picture shall be scaled up while keeping the original reconstructed picture for use in future motion compensation. Each pixel of the picture shall be scaled up according to the following formula:

$Y[n] = clip((Y[n] – 128) * 2 + 128);$

$C_b[n] = clip((C_b[n] – 128) * 2 + 128);$

$C_r[n] = clip((C_r[n] – 128) * 2 + 128);$

8.1.2 Macroblock Layer Decode

As shown in Figure 8, the picture is composed of macroblocks and blocks. Figure 23 shows the elements that make up the I Picture macroblock layer.

8.1.2.1 Coded Block Pattern (CBPCY)

The coded block pattern specifies which of the six blocks that make up the macroblock have AC coefficient information coded within the bitstream. The coded block pattern is derived from the six-bit value obtained from decoding the variable-length CBPCY syntax element in the macroblock header. The table of section 11.5 shall be used. The coded block pattern (*cbpcy*) shall be derived from the six-bit value decoded from the CBPCY syntax element (*decoded_cbpcy*) as follows:

```
cbpcy = (predicted_Y0 << 5)| (predicted_Y1 << 4) | (predicted_Y2 << 3) | (predicted_Y3 << 2)| (decoded_cbpcy & 0x03)
where predicted_Y0 .. predicted_Y3 are each one-bit values calculated as follows:
if (LT3 == T2)
  predicted_Y0 = L1,
else
    predicted_Y0 = T2
predicted_Y0 ^= ((decoded_cbpcy >> 5) & 0x01);
if (T2 == T3)
  predicted_Y1 = predicted_Y0
else
    predicted_Y1 = T3
predicted_Y1 ^= ((decoded_cbpcy >> 4) & 0x01);
if (L1 == predicted_Y0)
    predicted_Y2 = L3
else
    predicted_Y2 = predicted_Y0
predicted_Y2 ^= ((decoded_cbpcy >> 3) & 0x01);
if (predicted_Y0 == predicted_Y1)
    predicted_Y3 = predicted_Y2
else
    predicted_Y3 = predicted_Y1
predicted_Y3 ^= ((decoded_cbpcy >> 2) & 0x01);
```

Figure 34: Calculation of cbpcy

L0, L1, L2, L3, LT3, T0, T1, T2 and T3 are one-bit values representing the coded status of the neighboring luma blocks as illustrated in Figure 35. If the neighboring block is outside the frame boundaries, or if the neighboring block belongs to a different slice, its coded status shall be set to zero. The figure shows the four luma blocks which make up the current macroblock outlined in a heavy border along with blocks from the neighboring macroblocks. The values of T0,

T1, etc define whether the corresponding block was coded or not. For example, if L0 == 1, then block Y0 in the macroblock to the immediate left of the current macroblock was coded. If L0 == 0, then the block was not coded.

Note: The cbpcy bits corresponding to Cb and Cr are not affected by this calculation.

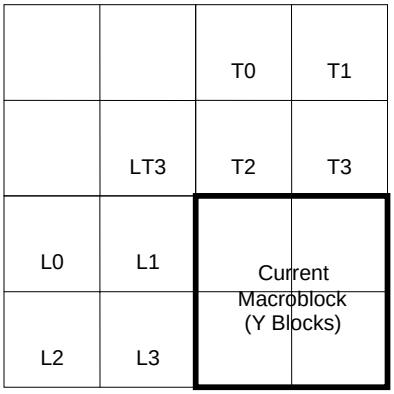


Figure 35: CBP encoding using neighboring blocks

The six-bit coded block pattern (*cbpcy*) shall specify which of the six blocks that make up the macroblock have at least one non-zero AC coefficient coded in the block layer bitstream. The bit positions in the six-bit coded block pattern syntax element shall correspond to the six blocks as defined in Table 70 (bit position 0 is the rightmost bit):

Table 70: **Coded block pattern bit position**

	Coded Block Pattern Bit Position					
	5	4	3	2	1	0
Block	Y0	Y1	Y2	Y3	Cb	Cr

A bit value of 1 in the coded block pattern shall indicate that the corresponding block has at least one non-zero AC coefficient coded in the block layer bitstream. A value of zero shall indicate that there are no AC coefficients coded in the block layer bitstream.

8.1.2.2 AC Prediction Flag (ACPRED)

The ACPRED (**7.1.3.2**) syntax element in the macroblock header is a one-bit syntax element that shall specify whether AC prediction is used to decode the AC coefficients for all the blocks in the macroblock. Section 8.1.3.7 describes the AC prediction process. If ACPRED == 1, then AC prediction shall be used. If ACPRED == 0, it shall not be used.

8.1.3 Block Layer Decode

Figure 8 illustrates how each macroblock is made up of 6 blocks. As the figure shows, the 4 blocks that make up the Y component of the macroblock are decoded first followed by the C_b and C_r blocks.

Figure 36 shows the process used to reconstruct the 8x8 blocks.

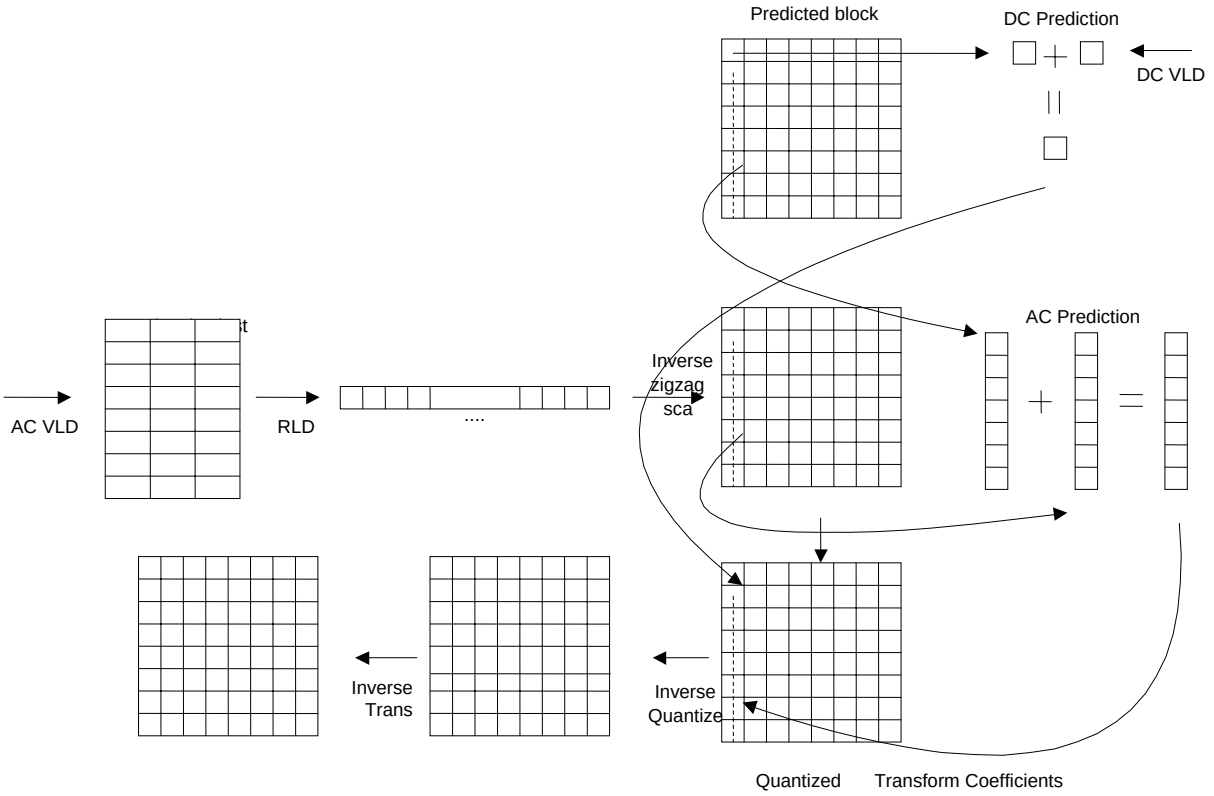


Figure 36: Intra block reconstruction

As Figure 36 shows, the DC and AC Transform coefficients are decoded using separate techniques. The DC coefficient is decoded differentially. An optional differential coding of the left or top AC coefficients may be used. The following sections describe the process for reconstructing intra blocks.

8.1.3.1 DC Differential Bitstream Decode

The DC coefficient shall be decoded differentially with respect to an already-decoded DC coefficient neighbor. This section describes the process used to decode the bitstream to obtain the DC differential.

Figure 27 shows the bitstream elements used to decode the DC differential. DCCOEF (**7.1.4.1**) shall be decoded using one of two VLC code tables. The table shall be specified by the TRANSDCTAB syntax element in the picture header (see section 8.1.1.2). Based on the value of TRANSDCTAB, one of the two tables listed in section 11.7 shall be used to decode DC Differential. If TRANSDCTAB == 0, then the low motion luma DC differential and the low motion Color-Differential DC differential tables of Section 11.7.1 shall be used to decode the DC differential for luma and color-difference components respectively. If TRANSDCTAB == 1, then the corresponding high motion counterparts of Section 11.7.2 shall be used.

The pseudo-code of Figure 37, where QUANT refers to MQUANT, shall specify the DC differential decoding process and the decoding of the syntax elements DCCOEF, DCCOEF_ESC, DCCOEF_EXTQUANT1, DCCOEF_EXTQUANT2, and DCSIGN:

```
DCDifferential = vlc_decode(); // DCCOEF
if(DCDifferential != 0) {
    if(DCDifferential == ESCAPECODE) {
        if(QUANT == 1)
            DCDifferential = flc_decode(10); // DCCOEF_ESC
        else if(QUANT == 2)
            DCDifferential = flc_decode(9); // DCCOEF_ESC
```

```
            else // QUANT is > 2
                DCDifferential = flc_decode(8); // DCCOEF_ESC
            }
    else {   // DCDifferential is not ESCAPECODE
        if(QUANT == 1)
            DCDifferential = DCDifferential*4 + flc_decode(2) – 3; // DCCOEF_EXTQUANT1
        else if(QUANT == 2)
            DCDifferential = DCDifferential*2 + flc_decode(1) – 1; // DCCOEF_EXTQUANT2
        }
     DCSign = flc_decode(1);      // DCSIGN
     if (DCSign == 1)
    DCDifferential = -DCDifferential
}
```

Figure 37: DC Differential Decoding Pseudo-code

8.1.3.2 DC Predictor

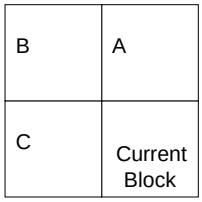


Figure 38: DC predictor candidates

The quantized DC value for the current block shall be obtained by adding the DC predictor to the DC differential obtained as described in section 8.1.3.1. In the simple and main profiles, the DC predictor shall be obtained directly from the DC coefficients of one of the previously decoded adjacent blocks. In the advanced profile, there may be an additional coefficient scaling step if the macroblocks quantizers of the neighboring blocks are different than that of the current block. The DC coefficient scaling (along with AC coefficient scaling) for prediction in advanced profile is described in section 8.1.3.9 and these scaled DC coefficients shall be used for computing the prediction direction, as well as the actual prediction. Figure 38 shows the current block and the candidate predictors from the adjacent blocks. The values A, B and C represent the quantized DC values for the top, top-left and left adjacent blocks respectively.

In the following cases there are no adjacent blocks:

1) The current block is in the first block row of the frame. In this case there are no A or B (and possibly C) blocks
2) The current block is in the first block column in the frame. In this case there are no B and C (and possibly A) blocks.

In the case of I and BI pictures in simple/main profile, the procedure specified in Figure 39 shall be used for calculating the DC predictor and the prediction direction. In the case of I and BI pictures in advanced profile, the procedure specified in Figure 40 shall be used for calculating the DC predictor and the prediction direction. In the advanced profile, if the macroblock quantizer is different from those of the predictor blocks, the DC components shall be scaled as specified in section 8.1.3.9 before selection the DC predictor and the prediction direction.

```
// If a neighboring block is unavailable for prediction, its DC coefficient is assumed to be the default_predictor
if (OVERLAP == 1 && PQUANT >= 9)
    default_predictor = 0; //overlap filtering is on in this case
else
    default_predictor =   (1024 + (DCStepSize >> 1)) / DCStepSize; // overlap filtering is off
```

```
if (predictor A is unavailable)
    predictor A's DC coefficient = default_predictor;
if (predictor B is unavailable)
    predictor B's DC coefficient = default_predictor;
if (predictor C is unavailable)
    predictor C's DC coefficient = default_predictor;
if (|predictor B's DC coefficient – predictor A's DC coefficient| <= |predictor B's DC coefficient – predictor C's DC coefficient|)
{
     prediction_direction = LEFT;
     DCPredictor = predictor C's DC coefficient;
}
else
{
     prediction_direction = TOP;
     DCPredictor = predictor A's DC coefficient;
}
```

Figure 39: Calculation of DC Predictor and Prediction Direction for Simple/Main Profile I and BI pictures: pseudo-code

```
if ((predictorA is available) && (predictorC is available))
{

     if (abs(predictorB's DC coefficient – predictorA's DC coefficient) <=
          abs(predictorB's DC coefficient – predictorC's DC coefficient)) {
          prediction_direction = LEFT;
                    DCPredictor = predictorC's DC coefficient;
          }
     else {
          prediction_direction = TOP;
                    DCPredictor = predictorA's DC coefficient;
               }
}
else if (predictorA is available) {
          prediction_direction = TOP;
          DCPredictor = predictorA's DC coefficient;
           }
else if (predictorC is available) {
          prediction_direction = LEFT;
          DCPredictor = predictorC's DC coefficient;
           }
else {
        prediction_direction = LEFT;
        DCPredictor = 0;
}
```

Figure 40: Calculating of DC Predictor and Prediction Direction for all other cases: pseudo-Code

The quantized DC coefficient shall be calculated by adding the DC differential and the DC predictor as follows:

DCCoeffQ = DCPredictor + DCDifferential

8.1.3.3 DC Inverse-quantization

The quantized DC coefficient shall be reconstructed by performing the following de-quantization operation:

DCCoefficient = DCCoeffQ * DCStepSize

The value of DCStepSize shall be derived from the value of MQUANT (obtained from PQUANT and the VOPDQUANT syntax elements) as follows:

For MQUANT equal to 1 or 2:

DCStepSize = 2 * MQUANT

For MQUANT equal to 3 or 4:

DCStepSize = 8

For MQUANT greater than or equal to 5:

DCStepSize = MQUANT / 2 + 6

8.1.3.4 AC Coefficient Bitstream Decode

The non-zero quantized AC coefficients shall be coded using a 3D run-level method. A set of tables and constants are used to decode the *run*, *level* and *last_flag* values. For descriptive purposes, the set of tables and constants is called an **AC coding set**. Following is a description of the tables and constants that make up an AC coding set.

Tables

The first step in reconstructing the AC Transform coefficients is to decode the bitstream to obtain the *run*, *level* and *last_flag* triplets that represent the location and quantized level for each non-zero AC coefficient.

Code table (CodeTable): The code table used to decode the ACCOEF1 and ACCOEF2 variable-length encoded syntax elements.

Run table (RunTable): The table of *run* values indexed by the value decoded in the ACCOEF1 or ACCOEF2 syntax elements

Level table (LevelTable): The table of level values indexed by the value decoded in the ACCOEF1 or ACCOEF2 syntax elements.

Not-last delta run table (NotLastDeltaRunTable): The table of delta *run* values indexed by the *level* value as illustrated in pseudo-code of Figure 41. It is used in escape coding Mode 2 (see Table 58).

Last delta run table (LastDeltaRunTable): The table of delta *run* values indexed by the *level* value as illustrated in pseudo-code of Figure 41. It is used in escape coding Mode 2 (see Table 58).

Not-last delta level table (NotLastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 41. It is used in escape coding Mode 1 (see Table 58).

Last delta level table (LastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 41. It is used in escape coding Mode 1 (see Table 58).

Presence of Fixed Length Codes – Mode3 (first_mode3): This is used in escape coding Mode 3 (see Table 58) where symbols are coded by fixed length codes. It shall be set to one at the beginning of a frame, field or a slice. It shall be set to zero, whenever Mode 3 is used for the first time.

Constants

Start index of last coefficient (StartIndexOfLast): The VLC encodes index values from 0 to N. The index values are used to obtain the run and level values from RunTable and LevelTable respectively. The first (StartIndexOfLast-1) of these index values corresponds to run, level pairs that are not the last pair in the block. The next StartIndexOfLast to N-1 index values correspond to run, level pairs that are the last pair in the block. The last value, N, is the Escape Index (see next).

Escape Index (EscapeIndex): The last in the set of indices encoded by the VLC. See the description above and the pseudo-code of Figure 41 for a description of how this constant is used.

The pseudo-code of Figure 41 shall specify how the tables and constants are used to decode a run, level and last_flag triplet.

```
decode_symbol(run, level, last_flag) {
    last_flag = 0;
    int index = vlc_decode(); // Use CodeTable to decode VLC codeword (ACCOEF1)
    if (index != EscapeIndex)
    {
     run = RunTable[index];
     level = LevelTable[index];
     sign = get_bits(1);
     if (sign == 1)
          level = -level;
     if (index >= StartIndexOfLast)
          last_flag = 1;
    }
    else
    {
     escape_mode = vlc_decode();     // Use Table 58   to decode ESCMODE syntax element
     if (escape_mode == mode1)
     {
          index = vlc_decode();    // Decode VLC codeword (ACCOEF2)
          run = RunTable[index];
          level = LevelTable[index];
          if (index >= StartIndexOfLast)
               last_flag = 1;
          if (last_flag == 0)
               level = level + NotLastDeltaLevelTable[run];
          else
               level = level + LastDeltaLevelTable[run];
          sign = get_bits(1);
          if (sign == 1)
               level = -level;

     }
     else if (escape_mode == mode2)
     {
          index = vlc_decode();    // Use Decode VLC codeword (ACCOEF2)
          run = RunTable[index];
          level = LevelTable[index];
          if (index >= StartIndexOfLast)
               last_flag = 1;
          if (last_flag == 0)
               run = run + NotLastDeltaRunTable[level] + 1;
          else
```

```
                run = run + LastDeltaRunTable[level] + 1;
            sign = get_bits(1);
            if (sign == 1)
                level = -level;
        }
        else if escape_mode == mode3 // (fixed-length encoding)
        {
                                last_flag = get_bits(1);
            if (first_mode3 == 1)
            {
                first_mode3 = 0;
                level_code_size = vlc_decode(); // Use Table 59 or Table 60 to decode
                run_code_size = 3 + get_bits(2);
            }
            run = get_bits(run_code_size);
            sign = get_bits(1);
            level = get_bits(level_code_size);
            if (sign == 1)
                level = -level;
        }
       }
}
```

Figure 41: Coefficient decode pseudo-code

The process illustrated in Figure 41 above for decoding the non-zero AC shall be repeated until *last_flag* == 1. This flag indicates the last non-zero coefficient in the block.

To improve coding efficiency, there are eight AC coding sets. The eight coding sets are divided into two groups of four, nominally called intra and inter coding sets. For Y blocks, one of the four intra coding sets shall be used. For C_b and C_r blocks, one of the four inter coding sets shall be used. Section 11.8 defines the tables that make up each coding set that shall be used. The particular set used to decode a block shall be signaled by an index value in the picture header. The following two tables define how the index corresponds to the coding set for Y and C_b/C_r blocks. As the tables show, if the value of PQINDEX (see section 7.1.1.6) is <= 8, then the high rate coding set shall be used for index 0. If PQINDEX is > 8, then the low motion coding set shall be used for index 0.

Table 71: Coding Set Correspondence for PQINDEX <= 8

Y blocks		C_b and C_r blocks	
Index	Table	Index	Table
0	High Rate Intra (Table 219-Table 225)	0	HighRate Inter (Table 226- Table 232)
1	High Motion Intra (Table 177-Table 183)	1	High Motion Inter (Table 184-Table 190)
2	Mid Rate Intra (Table 205-Table 211)	2	Mid Rate Inter (Table 212-Table 218)

Table 72: Coding Set Correspondence for PQINDEX > 8

Y blocks		C_b and C_r blocks	
Index	Table	Index	Table
0	Low Motion Intra (Table 191-Table 197)	0	Low Motion Inter (Table 198-Table 204)
1	High Motion Intra (Table 177-Table 183)	1	High Motion Inter (Table 184-Table 190)
2	Mid Rate Intra (Table 205-Table 211)	2	Mid Rate Inter (Table 212-Table 218)

The value decoded from the TRANSACFRM2 syntax element shall be used as the coding set index for Y blocks and the value decoded from the TRANSACFRM syntax element shall be used as the coding set index for C_b and C_r blocks.

8.1.3.5 AC Run-level Decode

The ordered run and level pairs obtained as described in section 8.1.3.4 shall be used to form 63 elements (representing AC coefficients) in a 64 element array by employing a run-level decode process as illustrated in the pseudo-code of Figure 42.

```
array[64] = {0};       // 64 element array initialized to zero.
array[0] = DCCoefficient;
curr_position = 1;
do {
    decode_symbol(run, level, last_flag);// decode the bitstream as described in Figure 41 to
                                         // obtain run, level and last_flag values for coefficient
    array[curr_position + run] = level;
    curr_postion = curr_postion + run + 1;
} while (last_flag != 1)
```

Figure 42: Run-level decode pseudo-code

8.1.3.6 Zigzag Scan of AC Coefficients

Decoding the run-level pairs as described in section 8.1.3.5 produces a one-dimensional array of quantized transform coefficients. The elements in the array shall be scanned to form an 8x8 two-dimensional array in preparation for the Inverse Transform. Figure 44 shows the elements in an 8x8 array labeled from 0 to 63. A mapping array is used to scan out the coefficients in the one-dimensional array to the 8x8 array, where the DC coefficient always maps to position (0,0). The process of scanning out coefficients for an NxM Inverse Transform block is defined in Figure 43.

```
// NxM Inverse Transform Block has M rows and N columns
// array [N * M ] is represents coefficients after run-level decode
// Coeff[M][N] represents Transform Coefficient block
// zigzagscan [N * M] represents mapping used to scan out coefficients
for (i = 0; i < N * M; i++) {
   index = zigzagscan[i];
```

```
    row = index / N;
    col = index % N;
    Coeff[row][col] = array [i];
}
```

Figure 43: Zigzag Scan Pseudo-code for NxM Block

As an example,

Figure 46 shows the mapping array used to produce the one-dimensional to two-dimensional scan out pattern shown in Figure 45.

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	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	52	53	54	55
56	57	58	59	60	61	62	63

Figure 44: 8x8 array with positions labeled

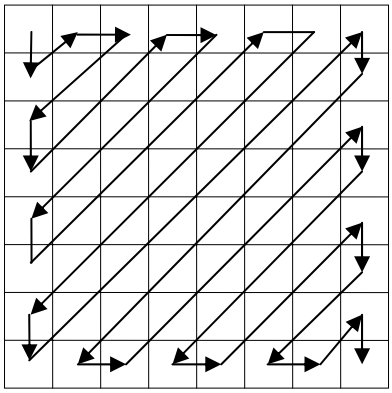


Figure 45: Example zigzag scanning pattern

0	8	1	2	9	16	24	17	10	3	4	11	18	25	32	40	33	26	19	12	5	6	13	20	27	34	41	48
56	49	42	35	28	21	14	7	15	22	29	36	43	50	57	58	51	44	37	30	23	31	38	45	52	59	60	53
46	39	47	54	61	62	55	63																				

Figure 46: Zigzag scan mapping array

One of three scan arrays shall be used to scan out the one-dimensional array depending on the AC prediction status for the block (see section 8.1.3.7 for description of AC prediction). The AC prediction status shall determine which scan array is used according to Table 73.

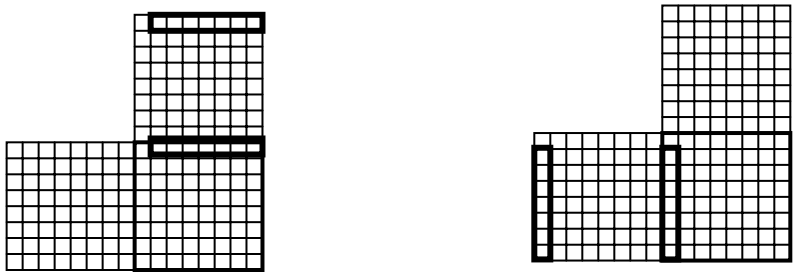
Table 73: **Scan Array Selection**

AC Prediction	AC Scan Array
ACPRED == 0	Normal scan
ACPRED == 1 and prediction_direction == Top	Horizontal scan
ACPRED == 1 and prediction_direction == Left	Vertical scan

The tables for the normal, horizontal, and vertical scan arrays as defined in section 11.9.1 shall be used.

8.1.3.7 AC Prediction

If the ACPRED (**7.1.1.28**, **7.1.3.2**) syntax element specifies that AC prediction is used for the blocks, then the top row or left column of AC coefficients in the decoded block shall be treated as differential values from the coefficients in the corresponding row or column in a predicted block. The predictor block shall be either the block immediately above or to the left of the current block. For each block, the direction chosen for the DC predictor shall be used for the AC predictor (see section 8.1.3.2). If there is no intra block in this direction, the AC predictor shall be set to zero. Figure 47 shows that for top prediction the first row of AC coefficients in the block immediately above shall be used as the predictor for the first row of AC coefficients in the current block. For left prediction the first column of AC coefficients in the block to the immediate left shall be used as the predictor for the first column of AC coefficients in the current block. There is an additional coefficient scaling step if the macroblock quantizers of the neighboring blocks are different than that of the current block. The AC coefficient scaling (along with DC coefficient scaling) for prediction in advanced profile is described in section 8.1.3.9, and these scaled AC coefficients shall be used for prediction.



Top Prediction Left Prediction

Figure 47: AC prediction candidates

If a block does not exist in the predicted direction then the predicted values for all 7 coefficients shall be set to zero. For example, if the prediction is ‘TOP’ but the block is in the top row, then there is no adjacent block in the ‘TOP’ direction.

When the ACPRED syntax element specifies that AC prediction is not used for the blocks, the predictors for the 7 AC coefficients in the first row or first column shall be set to zero.

The AC coefficients in the predicted row or column shall be added to the corresponding decoded AC coefficients in the current block to produce the fully reconstructed quantized Transform coefficient block as defined in Figure 48.

```
// Coeff[8][8] represents Transform Coefficient block
// PredTop[7] represents first row of AC coefficients (after scaling if needed) in block above
// PredLeft[7] represents first column of AC coefficients (after scaling if needed) in block to the left
if (ACPRED == 1 && prediction_direction == TOP) {
```

```
        for (col = 1; col < 8; col++)
            Coeff[0][col] = Coeff[0][col] + PredTop[col-1];
}
else if (ACPRED == 1 && prediction_direction == LEFT) {
        for (row = 1; row < 8; row++)
            Coeff[row][0] = Coeff[row][0] + PredLeft[row-1];
}
```

Figure 48: AC Prediction pseudo-code

8.1.3.8 Inverse AC Coefficient Quantization

Depending on whether the uniform or non-uniform quantizer is used (see section 7.1.1.6 and 7.1.1.8), the non-zero quantized AC coefficients reconstructed as described in the sections above shall be inverse quantized according to the following formula:

(if uniform quantizer)

dequant_coeff = *quant_coeff* * *double_quant*,

or (if nonuniform quantizer)

dequant_coeff = *quant_coeff* * *double_quant* + sign(*quant_coeff*) * *quant_scale*

where:

quant_coeff is the quantized coefficient

dequant_coeff is the inverse quantized coefficient

if the block is coded with PQUANT

double_quant = 2 * MQUANT + HALFQP

or if the block is coded with quantizer derived from VOPDQUANT syntax elements (i.e. derived from ALTPQUANT, or PQDIFF/MQDIFF, or ABSPQ/ABSMQ)

double_quant = 2 * MQUANT

quant_scale = MQUANT

MQUANT may be decoded in the macroblock layer in the VOPDQUANT syntax elements, or may be derived from the picture layer PQUANT, as defined in sections 7.1.1.6, 7.1.1.31, 7.1.3.4 and 7.1.3.5. HALFQP is decoded in the picture layer as defined in section 7.1.1.7.

8.1.3.9 Coefficient Scaling

For DC and AC prediction, the coefficients in the predicted blocks shall be scaled if the macroblocks quantizers are different than that of the current block. The scaling process is described below.

$$\overline{DC}_p = (DC_p * DCSTEP_p * DQScale[DCSTEP_c] + 0x20000) >> 18,$$

$$\overline{AC}_p = (AC_p * STEP_p * DQScale[STEP_c] + 0x20000) >> 18$$

where

$\overline{DC}_p$ is the scaled DC coefficient in the predictor block

DC_p is the original DC coefficient in the predictor block

$DCSTEP_p$ is the DCStepSize of the predictor block

$DCSTEP_c$ is the DCStepSize in the current block

$\overline{AC}_p$ is the scaled AC coefficient in the predictor block

AC_p is the original AC coefficient in the predictor block

$STEP_p$ is the double_quant-1 in the predictor block

$STEP_c$ is the double_quant-1 in the current block

$DQScale$ is a 63 element integer array as shown in Table 74.

Table 74: DQScale

Index	DQScale[Index]
1	262144
2	131072
3	87381
4	65536
5	52429
6	43691
7	37449
8	32768
9	29127
10	26214
11	23831
12	21845
13	20165
14	18725
15	17476
16	16384
17	15420
18	14564
19	13797
20	13107
21	12483
22	11916
23	11398
24	10923
25	10486

26	10082
27	9709
28	9362
29	9039
30	8738
31	8456
32	8192
33	7944
34	7710
35	7490
36	7282
37	7085
38	6899
39	6722
40	6554
41	6394
42	6242
43	6096
44	5958
45	5825
46	5699
47	5578
48	5461
49	5350
50	5243
51	5140
52	5041
53	4946
54	4855
55	4766
56	4681
57	4599
58	4520
59	4443
60	4369

61	4297
62	4228
63	4161

The product $DC_p * DCSTEP_p$, and the product $\overline{AC}_p * STEP_p$ product shall not exceed the signed 12 bit range, i.e., these product values shall be limited to ≥ -2048 && ≤ 2047.

8.1.3.10 Inverse Transform

After reconstruction of the transform coefficients, the resulting 8 × 8 blocks shall be processed by a separable two-dimensional inverse transform of size 8 by 8 that is in accordance with Annex A. The inverse transform output shall have a dynamic range of 10 bits, as a signed integer.

Subsequent to the inverse transform, the process of overlap smoothing shall be carried out if signaled, as described in section 8.5. Finally, the constant value of 128 shall be added to the reconstructed and possibly overlap smoothed intra block. This result shall be clamped to the range [0 255] and shall form the reconstruction prior to loop filtering.

For simple and main profile I frames, if overlap filtering is not used, the constant 128 shall not be added prior to clamping. The conditions for when overlap filtering is applied in simple and main profile I frames are described section 8.5.1. For advanced profile I frames, or if overlap filtering is used in simple and main profile I frames, the constant 128 shall be added prior to clamping.

8.2 Progressive BI Frame Picture Decoding

A BI frame picture is a special type of frame that is a hybrid of I and B pictures and shall not be used as an anchor or reference frame to predict other frames. BI frame pictures are permitted in main and advanced profiles only.

The syntax of BI frame pictures is almost identical to that of I frame pictures. This section describes only the differences between decoding of BI frame pictures and decoding of I frame pictures as described in section 8.1.

8.2.1 BFRACTION following picture type (main profile only)

The BFRACTION syntax element (section 7.1.1.14) immediately follows the picture type in a BI frame sent in main profile. BFRACTION == “1111111b” (Table 40) indicates that this B frame shall be re-interpreted as BI.

In advanced profile, the PTYPE syntax element is used to signal the BI frame as defined in Table 35.

8.2.2 No picture resolution index (RESPIC)

RESPIC (**7.1.1.10**) shall not be present in a BI frame. BI frames, along with B frames are constrained to operate at the same resolution as their corresponding anchor frames.

8.3 Progressive P Frame Picture Decoding

Figure 59 shows the steps required to decode and reconstruct blocks in progressive P frame pictures. The following sections describe the process for decoding progressive P frame pictures. Section 8.3.1 defines frame skipping, section 8.3.2 defines the handling of out-of-bounds reference pixels, section 8.3.3 defines the P picture types, section 8.3.4 defines the picture layer decoding, section 8.3.5 defines macroblock layer decoding, section 8.3.6 defines block layer decoding, section 8.3.7 defines rounding control, and section 8.3.8 defines intensity compensation.

8.3.1 Skipped Frame Pictures

In main and simple profiles, frame skipping is signaled through additional means. As a coded frame always contains more than 8 bits of data, if the total length of the data comprising a compressed frame is 8 bits, this signals that the frame was coded as a P frame with no motion or residual error information present (a non-coded frame). The decoder shall not decode the 8 bits used for frame skipping as they do not represent an actual codeword. In main profile, if the sequence level syntax element MAXBFRAMES > 0 (Annex J.1.18), a frame shall not be coded as a skipped frame.

In advanced profile, a skipped frame is signaled by the PTYPE syntax element (**7.1.1.4**) in the picture header. If a frame is signaled as skipped then it represents a P frame which is identical to the reference frame. Therefore, the reconstruction of the skipped frame pixel data shall be a copy of the reference frame pixel data.

8.3.2 Out-of-bounds Reference Pixels

The previously decoded anchor frame shall be used as the reference for motion-compensated predictive coding of the current P frame. The motion vectors used to locate the predicted blocks in the reference frame may include pixel locations that are outside the boundary of the reference frame. In these cases, the out-of-bounds pixel values shall be the replicated values of the edge pixel. Figure 49 illustrates pixel replication for the upper-left corner of the frame. In the advanced profile, "frame edge", "frame corner" and "outside the boundary" shall refer to the true frame dimensions, not the dimensions right or bottom justified to the edge of the macroblock. In other words, the right and bottom pixels that are repeated to infinity for a 300 x 200 image shall begin at column 304 and row 208 for the simple and main profiles. However, for the advanced profile, these shall begin respectively at column 300 and row 200.

The reference frames in advanced profile shall have the vertical boundary pixels replicated in a manner which is dependent on the reference frame coding type. If the reference frame was coded as progressive, then the top row of pixels shall be replicated upward as shown in the middle diagram of Figure 49. Similarly, the bottom row of pixels shall be replicated downward.

If the reference frame was coded as interlaced, i.e. either frames coded as interlace frame picture or as interlace field picture, then the top line shall be replicated upward in every other line of the frame and the second-to-top line shall be replicated upward in every other line of the frame as shown in the right diagram of Figure 49. The bottom row of pixels shall be replicated downward in a similar manner.

Reference pixels that are out-of-bounds along both axes shall be determined by horizontal pixel replication to an appropriate width, followed by vertical pixel replication to an appropriate height. For progressive frames, such out-of-bound pixels shall be identical to the closest frame corner pixel. For interlace frames (frames coded as interlace frame picture or as interlace field picture), such out-of-bound pixels shall be either the frame corner pixel, or the in-bounds pixel to the immediate top or bottom of the frame corner pixel, depending on the polarity of the field line.

See Annex K.2 for additional information on internal representation of a frame.

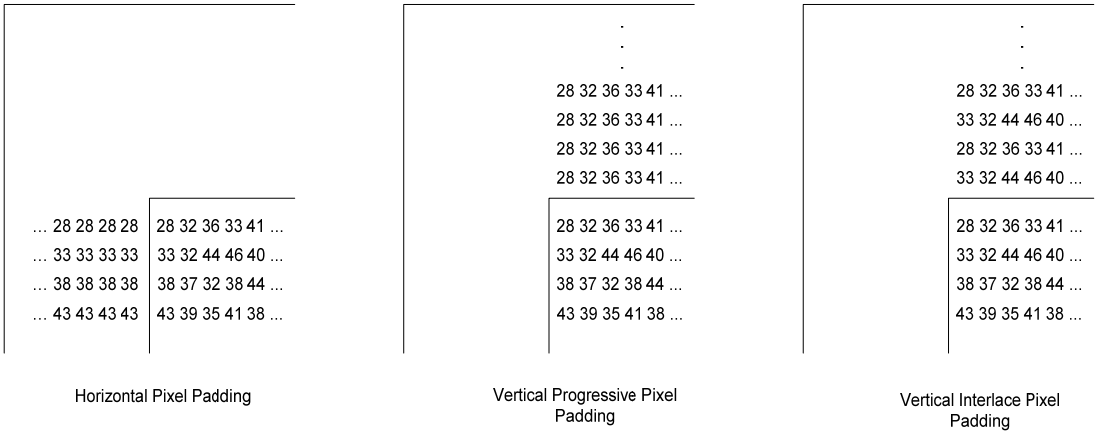


Figure 49: Horizontal and vertical pixel replication for out-of-bounds reference

8.3.3 P Picture Types

P pictures shall be one of 2 types: 1-MV and Mixed-MV. The following sections describe each P picture type.

8.3.3.1 1-MV P Picture

In 1-MV P pictures, a single motion vector shall be used to indicate the displacement of the predicted blocks for all 6 blocks in the macroblock. The 1-MV mode is signaled by the MVMODE and MVMODE2 picture layer syntax elements as described in section 8.3.4.3.

8.3.3.2 Mixed-MV P Picture

In Mixed-MV P pictures, each macroblock may be decoded as a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luma blocks shall have a motion vector associated with it. The 1-MV or 4-MV mode for each macroblock is indicated by the MVTYPEMB bitplane syntax element in the picture layer as described in section 8.3.4.3. The Mixed-MV mode is signaled by the MVMODE and MVMODE2 picture layer syntax elements as described in section 8.3.4.3.

8.3.4 P Picture Layer Decode

Figure 16 shows the elements that make up the progressive P picture layer header. The following sections provide extra detail for those elements that are not self-explanatory.

8.3.4.1 Picture-level Quantizer Scale (PQUANT)

The picture level quantizer scale PQUANT is decoded from the 5-bit picture layer syntax element PQINDEX as described in section 7.1.1.6. PQUANT shall specify the picture level quantizer scale (a value between 1 and 31) for the macroblocks in the current picture. When the sequence header DQUANT == 0, then PQUANT shall be used as the quantization step size for every macroblock in the current picture. When DQUANT != 0, then PQUANT shall be used as described in section 7.1.1.31. The PQINDEX syntax element shall also specify whether the uniform or non-uniform quantizer is used for all macroblocks in the frame.

8.3.4.2 Picture Resolution Index (RESPIC)

The RESPIC syntax element (**7.1.1.10**) in P pictures shall carry the same resolution as the RESPIC syntax element of the closest temporally preceding I frame. In other words, the resolution of an I picture determines the resolution of all subsequent P pictures until the next I picture. For example, if an I picture specifies a resolution index of 1 (full vertical resolution, half horizontal resolution), then all subsequent P pictures specify the same resolution until the next I picture.

All P pictures that are coded at less than full resolution may be up-sampled to full resolution prior to display. This up-sampling process is outside the reconstruction loop. The spatial alignment of video samples of the down-sampled frame with respect to the video samples of the original frame is described in Annex B.

8.3.4.3 Picture Layer Motion Compensation and Intensity Compensation Decoding

The P picture layer contains syntax elements that control the motion compensation mode and intensity compensation for the frame. The MVMODE syntax element (**7.1.1.32**) is a variable sized value that signals either:

1) one of four motion vector modes for the frame or

2) that intensity compensation is used in the frame.

If intensity compensation is signaled, then the MVMODE2 (**7.1.1.33**), LUMSCALE (**7.1.1.34**) and LUMSHIFT (**7.1.1.35**) syntax elements shall follow in the picture layer. In this case, MVMODE2 shall signal the motion vector mode and LUMSCALE and LUMSHIFT are 6-bit values which shall specify parameters used in the intensity compensation process. Refer to section 8.3.8 for a definition of intensity compensation decode.

Table 46 and Table 47 define the code tables used to decode the MVMODE syntax element. Table 46 shall be used if PQUANT is > 12 and Table 47 shall be used if PQUANT is <= 12. In a similar fashion, Table 49 and Table 50 shall be used to decode the MVMODE2 syntax element. Either MVODE or MVMODE2 signal one of four motion vector modes.

If the motion vector mode is ‘Mixed-MV’ mode, then the MVTYPEMB syntax element (**7.1.1.36**) is present in the picture layer. MVTYPEMB is a bitplane coded syntax element that indicates the 1-MV/4-MV motion vector status for each macroblock in the picture. The decoded bitplane represents the motion vector status for each macroblock as a syntax element of 1-bit values from upper left to lower right. Refer to section 8.7 for a definition of the bitplane coding. A value of 0 shall indicate that the macroblock is coded in 1-MV mode. A value of 1 shall indicate that the macroblock is coded in 4-MV mode. Refer to section 8.3.5.2.1 for a description of the motion vector decoding process.

8.3.4.4 Skipped Macroblock Decoding (SKIPMB)

The P picture layer contains the SKIPMB syntax element (**7.1.1.37**) which is a bitplane coded syntax element that defines the skipped/not-skipped status of each macroblock in the picture. The decoded bitplane represents the

skipped/not-skipped status for each macroblock as a syntax element of 1-bit values from upper left to lower right. Refer to section 8.7 for a definition of the bitplane coding. A value of 0 shall indicate that the macroblock is not skipped. A value of 1 shall indicate that the macroblock is coded as skipped.

A skipped macroblock shall not contain any prediction error information. A skipped status for a macroblock means that the macroblock may only contain the HYBRIDPRED syntax element as a qualifier to the predicted motion vector(s). Refer to section 8.3.5.3.5 for a description of how the HYBRIDPRED syntax element (**7.1.3.9**) is used in the decoding process.

8.3.4.5 Motion Vector Table (MVTAB)

The table used to decode the motion vector differentials is indicated by the MVTAB syntax element as described in section 7.1.1.38. Refer to section 8.3.5.2.1 for a description of the motion vector decoding process.

8.3.4.6 Macroblock-level Quantizer Mode

See section **7.1.1.31**.

8.3.4.7 Macroblock-level Transform Type Flag (TTMBF)

TTMBF (**7.1.1.40**) is a one-bit syntax element that signals whether transform type coding is enabled at the frame or macroblock level. If TTMBF == 1, then the same transform type shall be used for all blocks in the frame. In this case, the transform type shall be signaled in the TTFRM syntax element (**7.1.1.41**) that follows. If TTMBF == 0, then the transform type may vary throughout the frame and shall be signaled at either the macroblock or block levels.

8.3.4.8 Frame-level Transform Type (TTFRM)

TTFRM (**7.1.1.41**) is a variable-length syntax element that shall be present in the picture layer if TTMBF == 1. TTFRM shall be decoded using Table 53 and shall signal the Transform type used to transform the 8x8 pixel error signal in predicted blocks. The 8x8 error blocks may be transformed using an 8x8 Transform, two 8x4 Transforms, two 4x8 Transforms or four 4x4 Transforms.

8.3.4.9 Frame-level Transform AC Coding Set Index (TRANSACFRM)

TRANSACFRM (**7.1.1.11**) is a variable-length syntax element that shall be present in the picture layer. This syntax element indexes the coding set used to decode the Transform AC coefficients for the intra- and inter-coded blocks. Table 39 is used to decode the TRANSACFRM syntax element.

8.3.4.10 Intra Transform DC Table (TRANSDCTAB)

The TRANSDCTAB syntax element (**7.1.1.13**) shall have the same meaning as the TRANSDCTAB syntax element in I pictures. See section 8.1.1.2 for its definition.

8.3.4.11 Range Reduction Frame - P Frame (RANGEREDFRM)

The RANGEREDFRM is only signaled when RANGERED == 1.

When RANGEREDFRM == 1 for the current P frame, the pixels of the current decoded frame shall be scaled up similar to the scaling of the pixels of the I frame (described in section 8.1.1.4), while keeping the current reconstructed frame intact. The pixels shall be scaled up according to the following formula:

$$Y[n] = \text{clip}((Y[n] - 128) * 2 + 128);$$
$$C_b[n] = \text{clip}((C_b[n] - 128) * 2 + 128);$$
$$C_r[n] = \text{clip}((C_r[n] - 128) * 2 + 128);$$

In addition, the pixels of the previously reconstructed anchor frame shall be scaled prior to using them for motion compensation if the current frame and previous frame are operating at different range. The process shall be applied to the pixels of the reconstructed frame as the first stage of decoding prior to Intensity compensation, motion compensation, and macroblock level decoding.

More specifically, there are two cases that require scaling of the pixels of the previous reconstructed frame.

- The current frame's RANGEREDFRM is signaled and the previous frame's RANGEREDFRM is not signaled. In this case, the pixels of the previously reconstructed frame shall be scaled down as follows:

```
Y[n] = ((Y[n] – 128)   >> 1) + 128;
Cb[n] = ((Cb[n] – 128)   >> 1) + 128;
Cr[n] = ((Cr[n] – 128)   >> 1) + 128;
```

- The current frame's RANGEREDFRM is not signaled and the previous frame's RANGEREDFRM is signaled. In this case, the pixels of the previous reconstructed frame shall be scaled as follows:

```
Y[n] = clip((Y[n] – 128)   * 2   + 128);
Cb[n] = clip((Cb[n] – 128)   * 2   + 128);
Cr[n] = clip((Cr[n] – 128)   * 2   + 128);
```

8.3.5 Macroblock Layer Decoding

This section defines the Macroblock layer decoding for progressive P-frame pictures for simple, main and advanced profiles.

8.3.5.1 Macroblock Types

Macroblocks in P pictures shall be one of 3 types: 1-MV, 4-MV, or Skipped. The macroblock type is indicated by a combination of picture and macroblock layer syntax elements. The following sub-sections describe each type and how they are signaled.

8.3.5.1.1 1-MV Macroblocks

1-MV macroblocks may occur in 1-MV and Mixed-MV P pictures. In a 1-MV macroblock, a single MVDATA syntax element (**7.1.3.8**) is associated with all blocks in the macroblock. The MVDATA syntax element signals whether the blocks are coded as Intra or Inter type. If they are coded as Inter, then the MVDATA syntax element shall also indicate the motion vector differential. See section 8.3.6.1 for a description of how to decode Intra blocks in P pictures and see section 8.3.6.2 for a description of how to decode Inter blocks in P pictures.

If the P picture is of type 1-MV, then all the macroblocks in the picture are of type 1-MV and the macroblock type is not signaled.

If the P picture is of type Mixed-MV, then the macroblocks in the picture may be of type 1-MV or 4-MV. In this case the macroblock type (1-MV or 4-MV) is signaled in the MVTYPEMB syntax element (**7.1.1.36**) in the picture layer. See section 8.3.4.3 for a definition of how the MVTYPEMB syntax element signals the 1-MV/4-MV macroblock type.

8.3.5.1.2 4-MV Macroblocks

4-MV macroblocks shall only occur in Mixed-MV P pictures. A 4-MV macroblock shall be indicated by signaling that the macroblock is 4-MV in the MVTYPEMB (**7.1.1.36**) picture layer syntax element, or in the MVMODEBIT (**7.1.3.6**) macroblock layer syntax element. Individual blocks within a 4-MV macroblock may be coded as Intra blocks. For the 4 luma blocks, the Intra/Inter state shall be signaled by the BLKMVDATA syntax element (**7.1.3.11**) associated with that block. The CBPCY syntax element (**7.1.3.1**) shall indicate which blocks have BLKMVDATA syntax elements present in the bitstream. See section 8.3.5.5.2 for a definition of how the CBPCY syntax element is used in 4-MV macroblocks.

The Inter/Intra state for the color-difference blocks shall be derived from the luma Inter/Intra states. If 3 or 4 of the luma blocks are coded as Intra, then the color-difference blocks shall also be coded as Intra. Otherwise, the color-difference shall be coded as Inter.

8.3.5.1.3 Skipped Macroblocks

Skipped macroblocks can occur in both 1-MV and Mixed-MV P pictures. In all cases, a skipped macroblock shall be signaled by the SKIPMB (**7.1.1.37**) bitplane syntax element in the picture layer, or by the SKIPMBBIT macroblock layer syntax (**7.1.3.7**). See section 8.3.4.4 for a definition of the SKIPMB syntax element.

8.3.5.2 Macroblock Decoding Process

The following sub-sections describe the macroblock layer decoding process for P picture macroblocks.

Refer to section 8.3.6.3 for a description of the inverse quantization process.

8.3.5.2.1 Decoding Motion Vector Differential

The MVDATA (**7.1.3.8**) or BLKMVDATA (**7.1.3.11**) syntax elements are used to decode motion information for the blocks in the macroblock. 1-MV macroblocks have a single MVDATA syntax element, and 4-MV macroblocks may have between zero and four BLKMVDATA syntax elements (see section 8.3.5.5.2 for a definition of how the CBPCY syntax element is used to decode the number of MVDATA syntax elements in 4-MV macroblocks).

Each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly codes three parameters:

1) the horizontal motion vector differential component,

2) the vertical motion vector differential component and

3) a binary flag indicating whether any Transform coefficients are present.

Whether the macroblock (or block for 4-MV) is Intra or Inter-coded is coded as one of the horizontal/vertical motion vector possibilities, i.e., one of the VLC entries for differential MV values indicates that the block is actually intra-coded.

The MVDATA and BLKMVDATA syntax elements are each a variable length codeword followed by a fixed length codeword. The value of the variable length codeword determines the size of the fixed length codeword. The MVTAB syntax element in the picture layer specifies the code table used to decode the variable sized codeword.

The pseudo-code of Figure 50 shall define how the motion vector differential, Inter/Intra type and ‘more_present flag’ information shall be decoded.

Note: The motion vector differentials decoded in this pseudo-code are modulo differentials. The computation of motion vectors from these differentials is shown in section 8.3.5.4.

The values: **‘more_present flag’, intra_flag, dmv_x** and **dmv_y** are computed in the pseudo-code of Figure 50. The values are defined as follows:

‘more_present flag’: a binary flag indicating whether any Transform coefficients are present (1 = coefficients present, 0 = coefficients not present)

intra_flag: a binary flag indicating whether the block or macroblock is intra-coded (0 = inter-coded, 1 = intra-coded)

dmv_x: the differential horizontal motion vector component

dmv_y: the differential vertical motion vector component

k_x, k_y: fixed length for long motion vectors

k_x and **k_y** depend on the motion vector range as defined by the MVRANGE symbol (section 7.1.1.9) and shall be set according to Table 75, where range_x and range_y are the motion vector ranges, in quarter pixel units, in horizontal and vertical directions respectively.

Table 75: k_x and k_y specified by MVRANGE

MVRANGE	k_x	k_y	range_x	range_y
0 (default)	9	8	256	128
10	10	9	512	256
110	12	10	2048	512
111	13	11	4096	1024

The value **halfpel_flag** is a binary value indicating whether half-pel or quarter-pel precision is used for the picture. The value of **halfpel_flag** is determined by the picture layer syntax element MVMODE (see section 8.3.4.3). If MVMODE specifies the mode as 1-MV or Mixed-MV, then **halfpel_flag** = 0 and quarter-pel precision shall be used. If MVODE

specifies the mode as 1-MV Half-pel or 1-MV Half-pel Bilinear, then **halfpel_flag** = 1 and half-pel precision shall be used.

size_table and **offset_table** are arrays defined as follows:

size_table[6] = {0, 2, 3, 4, 5, 8}
offset_table[6] = {0, 1, 3, 7, 15, 31}

```
index = vlc_decode()     // Use the table indicated by MVTAB in the picture layer
index = index + 1
if (index >= 37)
{
      'more_present flag' = 1
      index = index - 37
}
else
      'more_present flag' = 0

intra_flag = 0
if (index == 0)
{
      dmv_x = 0
      dmv_y = 0
}
else if (index == 35)
{
      dmv_x = get_bits(k_x – halfpel_flag)
      dmv_y = get_bits(k_y – halfpel_flag)
}
else if (index == 36)
{
      intra_flag = 1
      dmv_x = 0
      dmv_y = 0
}
else
{
      index1 = index % 6
      if (halfpel_flag == 1 && index1 == 5)
            hpel = 1
      else
            hpel = 0
        val = get_bits (size_table[index1] - hpel)
        sign = 0 - (val & 1)
        dmv_x = sign ^ ((val >> 1) + offset_table[index1])
        dmv_x = dmv_x - sign

      index1 = index / 6
```

```
    if (halfpel_flag == 1 && index1 == 5)
        hpel = 1
    else
        hpel = 0
      val = get_bits (size_table[index1] - hpel)
      sign = 0 - (val & 1)
      dmv_y = sign ^ ((val >> 1) + offset_table[index1])
      dmv_y = dmv_y - sign
}
```

Figure 50: Decoding MV Differential in Progressive Pictures: Pseudo-code

8.3.5.3 Motion Vector Predictors

Motion vectors shall be computed by adding the motion vector differential computed in section 8.3.5.2.1 to a motion vector predictor. All of these computations shall be performed in quarter-pixel units even if the actual motion vector was transmitted in half-pel mode. Motion vectors expressed in half-pel mode are converted to quarter-pel units by multiplying them by 2. The predictor shall be computed from three neighboring motion vectors. If a neighboring block is intra-coded, its motion vector shall be set to be zero for the purpose of prediction. The following sections describe how the predictors are calculated for macroblocks in 1-MV P pictures, and Mixed-MV P pictures.

Note: Predictor candidate MVs can be out of bounds as defined in 8.3.5.3.3.

8.3.5.3.1 Motion Vector Predictors In 1-MV P Pictures

Figure 51 shows the three motion vectors used to predict the motion vectors for the current macroblock within a picture. As the figure shows, the predictor shall be taken from the left, top and top-right macroblocks, except in the case where the macroblock is the last macroblock in the row. In this case, Predictor B shall be taken from the top-left macroblock instead of the top-right.

For the special case where the frame is one macroblock wide, the predictor shall always be Predictor A (the top predictor).

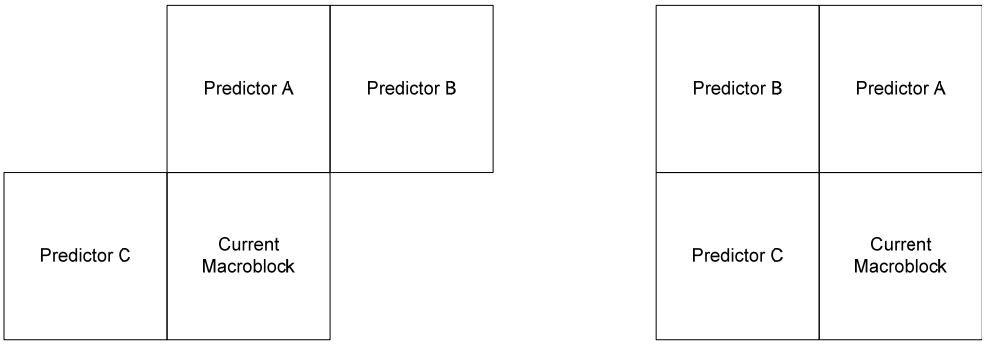


Figure 51: Candidate Motion Vector Predictors in 1-MV P Pictures

8.3.5.3.2 Motion Vector Predictors In Mixed-MV P Pictures

Figure 52 and Figure 53 show the 3 candidate motion vectors for 1-MV and 4-MV macroblocks in Mixed-MV P pictures. In the following figures, the larger rectangles are macroblock boundaries and the smaller rectangles are block boundaries.

For the special case where the frame is one macroblock wide, the predictor shall always be Predictor A (the top predictor).

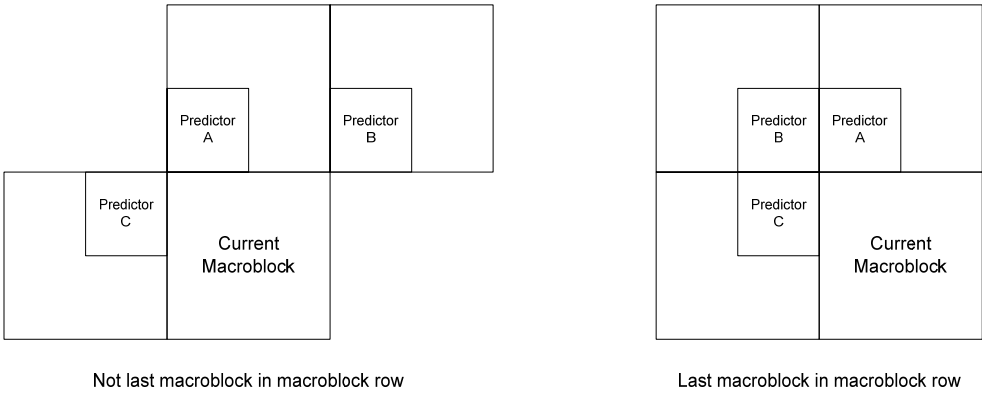


Figure 52: Candidate Motion Vectors for 1-MV Macroblocks in Mixed-MV P Pictures

Figure 52 shows the candidate motion vectors for 1-MV macroblocks. The neighboring macroblocks are either 1-MV or 4-MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4-MV (i.e., predictor A is the motion vector for block 2 in the macroblock above the current and predictor C is the motion vector for block 1 in the macroblock immediately to the left of the current). If any of the neighbors are 1-MV macroblocks, then the motion vector predictors shown in Figure 52 shall be taken to be the vectors for the entire macroblock. As the figure shows, if the macroblock is the last macroblock in the row, then Predictor B shall be from block 3 of the top-left macroblock instead of from block 2 in the top-right macroblock as is the case otherwise.

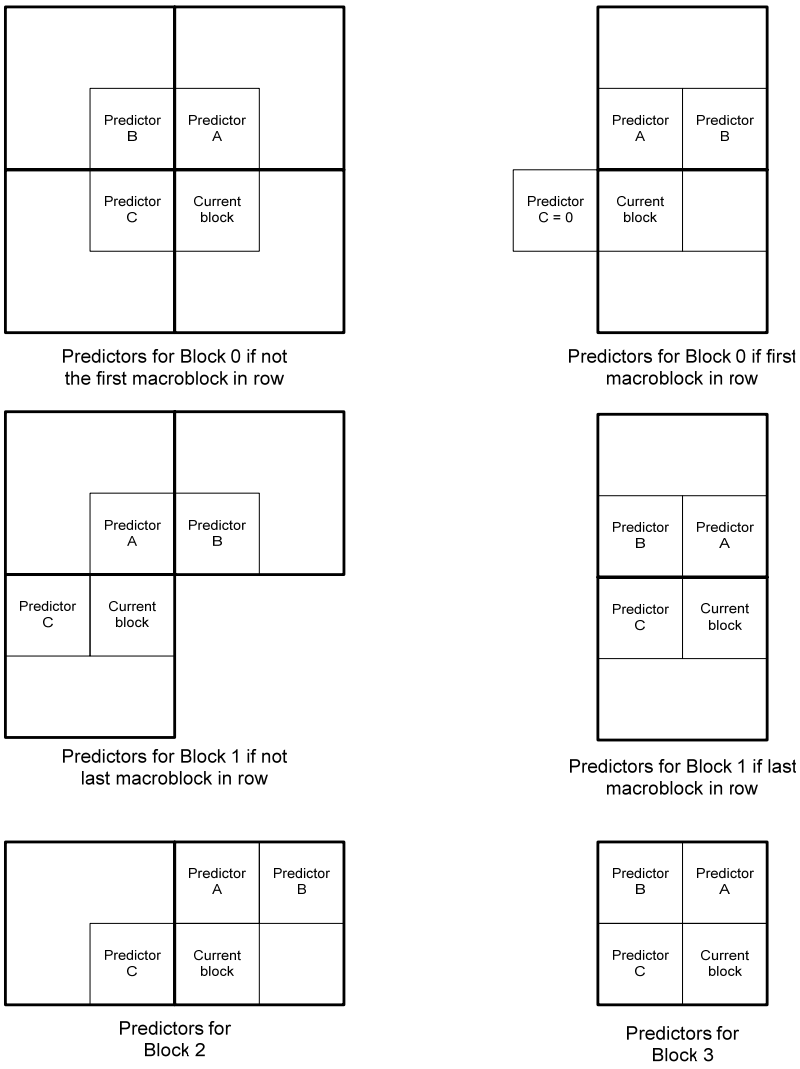


Figure 53: Candidate Motion Vectors for 4-MV Macroblocks in Mixed-MV P Pictures

Figure 53 shows the predictors for each of the 4 luma blocks in a 4-MV macroblock. For the case where the macroblock is the first macroblock in the row, Predictor B for block 0 shall be handled differently than the remaining blocks in the row. In this case, Predictor B shall be taken from block 3 in the macroblock immediately above the current macroblock instead of from block 3 in the macroblock above and to the left of current macroblock as is the case otherwise. Similarly, for the case where the macroblock is the last macroblock in the row, Predictor B for block 1 is handled differently. In this case, the predictor shall be taken from block 2 in the macroblock immediately above the current macroblock instead of from block 2 in the macroblock above and to the right of the current macroblock as is the case otherwise. If the macroblock is in the first macroblock column, then Predictor C for blocks 0 and 2 shall be set to 0.

8.3.5.3.3 Calculating the Preliminary Motion Vector Predictor

Given the 3 motion vector predictor candidates, the pseudo-code of Figure 54 shall specify the process for calculating the preliminary motion vector predictors. These preliminary motion vector predictors are used in the next section to compute the actual motion vector predictors.

In the pseudo-code, a motion vector predictor candidate shall be considered to be out of bounds, if either the corresponding block is outside the frame boundary, or if the corresponding block is part of a different slice. A predictor candidate that is out of bounds shall be set to zero.

```
// The preliminary predicted motion vector is (predictor_pre_x, predictor_pre_y)
```

```
if (predictorC is out of bound || predictorC is intra) {
        predictorC_x = predictorC_y = 0;
}
if (predictorA is out of bound || predictorA is intra) {
        predictorA_x = predictorA_y = 0;
}
if (predictorB is out of bound || predictorB is intra) {
        predictorB_x = predictorB_y = 0;
}
if (predictorA is not out of bound) {
        if (predictorC is out of bound && predictorB is out of bound) {
                // This condition is true only if the CodedWidth is less than or equal to 16.
                predictor_pre_x = predictorA_x;
                predictor_pre_y = predictorA_y;
      } else {
                // calculate predictor from A, B and C predictor candidates
               predictor_pre_x = median3(predictorA_x, predictorB_x, predictorC_x);
               predictor_pre_y = median3(predictorA_y, predictorB_y, predictorC_y);
        }
} else if (predictorC is not out of bound) {
          predictor_pre_x = predictorC_x;
          predictor_pre_y = predictorC_y;
} else {
          predictor_pre_x = predictor_pre_y = 0;
}
```

Figure 54: Calculating Preliminary MV Predictor: Pseudo-code

8.3.5.3.4 Pullback of the Preliminary Motion Vector Predictor

After the preliminary predicted motion vectors (predictor_pre_x, predictor_pre_y) are computed, a "pull-back' operation is performed, if necessary, on its values. The pull-back operation can be understood as follows: The preliminary predicted motion vector is applied as if it is the motion vector of the current macroblock or block. The preliminary predicted motion vector is checked to see if the block/macroblock referenced by it lies entirely outside of the reference frame. If the referenced region is entirely outside of the frame boundaries, the preliminary predictor motion vectors are clipped such that at least one pixel of the reference frame is inside the block/macroblock referenced by the predictor.

The pull-back operation shall be implemented as follows:

Let X = 16 * ((CodedWidth + 15) / 16) * 4 – 4, and

Y = 16 * ((CodedHeight + 15) / 16) * 4 – 4.

Pull-back of Motion Vector Predictor of Macroblock: Let (MBx, MBy) be the current macroblock in the frame with preliminary predicted motion vector (predictor_pre_x, predictor_pre_y). Let the values qx and qy represent the quarter pixel coordinates of the top left corner of the macroblock as qx = MBx * 16 * 4, qy = MBy * 16 * 4.

- If qx + predictor_pre_x < –60, then predictor_pre_x shall be set to the value (–60 – qx).
- If qx + predictor_pre_x > X, then predictor_pre_x shall be set to the value X – qx.
- If qy + predictor_pre_y < –60, then predictor_pre_y shall be set to the value (–60 – qy).
- If qy + predictor_pre_y >Y, then predictor_pre_y shall be set to the value Y – qy.

Pull-back of Motion Vector Predictor of block: Let (Bx, By) be the coordinates of the top left corner of the current block in the frame with preliminary predicted motion vector (predictor_pre_x, predictor_pre_y). Let the values qx and qy represent the quarter pixel coordinates of the top left corner of the block as qx = Bx * 8 * 4, qy = By * 8 * 4.

- If qx + predictor_pre_x < –28, then predictor_pre_x shall be set to the value (–28 – qx).
- If qx + predictor_pre_x > X, then predictor_pre_x shall be set to the value X – qx.
- If qy + predictor_pre_y < –28, then predictor_pre_y shall be set to the value (–28 – qy).
- If qy + predictor_pre_y > Y, then predictor_pre_y shall be set to the value Y – qy.

8.3.5.3.5 Hybrid Motion Vector Prediction (HYBRIDPRED)

In the P picture, the motion predictor calculated in the previous section shall be tested relative to the A and C predictors to see if the predictor is explicitly coded in the bitstream. If so, then the HYBRIDPRED syntax element (**7.1.3.9**) shall be decoded, and this syntax element shall specify whether to use predictor A or predictor C as the motion vector predictor. Hybrid motion vectors may exist even for skipped MBs, i.e. macroblocks which have zero differential motion vectors. The pseudo-code of Figure 55 shall specify the decoding of HYBRIDPRED using get_bits(), and shall also specify hybrid motion vector prediction.

The variables are defined as follows:

predictor_pre_x: The horizontal motion vector predictor as calculated in the above section
predictor_pre_y: The vertical motion vector predictor as calculated in the above section
predictor_post_x: The horizontal motion vector predictor after checking for hybrid motion vector prediction
predictor_post_y: The vertical motion vector predictor after checking for hybrid motion vector prediction

```
if ((predictorA is out of bounds) || (predictorC is out of bounds))
{
    predictor_post_x = predictor_pre_x
    predictor_post_y = predictor_pre_y
}
else
{
    if (predictorA is intra)
        sum = abs(predictor_pre_x) + abs(predictor_pre_y)
    else
        sum = abs(predictor_pre_x – predictorA_x) + abs(predictor_pre_y – predictorA_y)
    if (sum > 32)
    {
        // read next bit to see which predictor candidate to use
        if (get_bits(1) == 1)       // HYBRIDPRED syntax element
        {
            // use top predictor
            predictor_post_x = predictorA_x
            predictor_post_y = predictorA_y
        }
        else
        {
            // use left predictor
            predictor_post_x = predictorC_x
            predictor_post_y = predictorC_y
        }
```

```
    }
    else
    {
        if (predictorC is intra)
            sum = abs(predictor_pre_x) + abs(predictor_pre_y)
        else
            sum = abs(predictor_pre_x – predictorC_x) + abs(predictor_pre_y – predictorC_y)
        if (sum > 32)
        {
            // read next bit to see which predictor candidate to use
            if (get_bits(1) == 1)
            {
                // use top predictor
                predictor_post_x = predictorA_x
                predictor_post_y = predictorA_y
            }
            else
            {
                // use left predictor
                predictor_post_x = predictorC_x
                predictor_post_y = predictorC_y
            }
        }
        else
        {
            predictor_post_x = predictor_pre_x
            predictor_post_y = predictor_pre_y
        }
    }
}
```

Figure 55: Hybrid Motion Vector: Preliminary Prediction

8.3.5.3.6 Motion Vector Predictors in Skipped Macroblocks

If a macroblock is coded as skipped, then the predicted motion vector computed as described above shall be used as the motion vector for the block (in a 4-MV skipped macroblock), or macroblock (in a 1-MV skipped macroblock). The block or macroblock referenced by the motion vector shall be used as the current block or macroblock in the current picture. The HYBRIDPRED syntax element (**7.1.3.9**) may be present in the macroblock layer indicating which of the predictor candidates to use. A skipped macroblock with 4 motion vectors may have up to 4 hybrid motion predictor (HYBRIDPRED) syntax elements, i.e. up to 4 bits.

8.3.5.4 Reconstructing Motion Vectors

The following sections describe how to reconstruct the luma and color-difference motion vectors for 1-MV and 4-MV macroblocks.

The flowchart defining the reconstruction of the luma motion vectors in is shown in Figure 56.

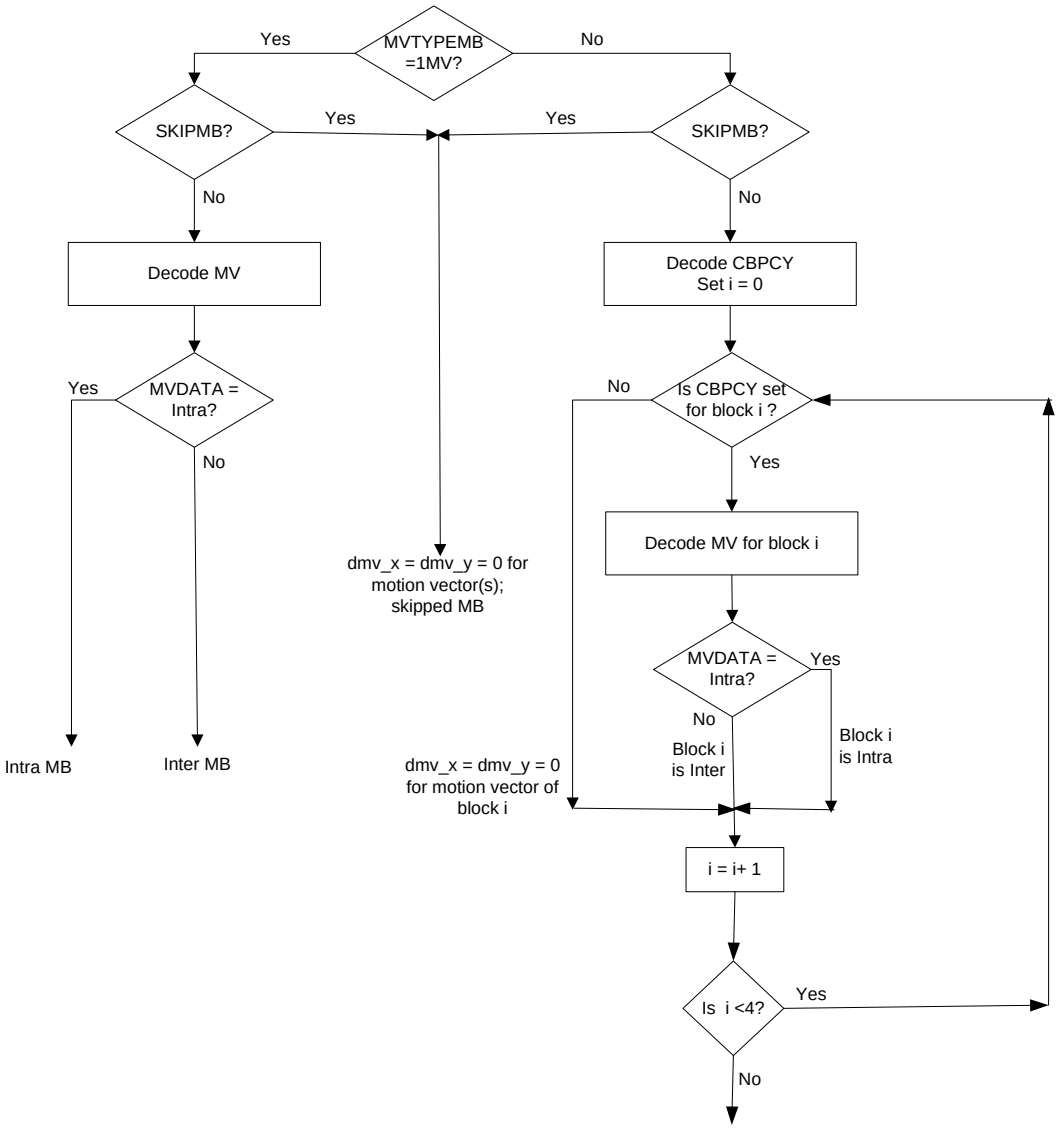


Figure 56: Flowchart depicting luma motion vector reconstruction

8.3.5.4.1 Luma Motion Vector Reconstruction

In all cases (1-MV and 4-MV macroblocks) the luma motion vector shall be reconstructed by adding the differential to the predictor as follows:

mv_x = (dmv_x + predictor_post_x) smod range_x

mv_y = (dmv_y + predictor_post_y) smod range_y

The smod operator ensures that the reconstructed vectors are valid. (A smod b) lies within –b and b – 1. range_x (motion vector range in the horizontal direction) and range_y (motion vector range in the vertical direction) depend on MVRANGE (**7.1.1.9**) and shall be as specified in Table 75.

If half pixel mode is used for signaling motion vectors, dmv_x and dmv_y shall be multiplied by 2 to convert them to quarter-pixel units before this operation.

The following subsections define luma motion vector constraints in 1-MV and 4-MV macroblocks.

1-MV Macroblock

In 1-MV macroblocks there is a single motion vector for all the 4 blocks that make up the luma component of the macroblock (see 8.3.5.1.1).

If the SKIPMB (**7.1.1.37**) syntax element in the picture layer indicates that the macroblock is skipped, then dmv_x = 0 and dmv_y = 0 (mv_x = predictor_post_x and mv_y = predictor_post_y).

If the macroblock is not skipped, and the MVDATA syntax element (**7.1.3.8**) shall be decoded to indicate (via the intra_flag) whether the macroblock is Intra-coded (as described in the section 8.3.5.2.1 above).

4-MV Macroblock

Each of the Inter-coded luma blocks in a macroblock has its own motion vector. Therefore, there are between 0 and 4 luma motion vectors in each 4-MV macroblock.

If the SKIPMB syntax element in the picture layer indicates that the macroblock is skipped, then dmv_x = 0 and dmv_y = 0 (mv_x = predictor_post_x and mv_y = predictor_post_y) for each block in the macroblock.

If the macroblock is not skipped, the CBPCY syntax element (described in section 8.3.5.5) in the macroblock shall indicate whether the block is not coded. If a block is not coded, then dmv_x = 0 and dmv_y = 0 (thus mv_x = predictor_post_x and mv_y = predictor_post_y for each block in the macroblock).

8.3.5.4.2 Color-Difference Motion Vector Reconstruction

The color-difference motion vectors are derived from the luma motion vectors. Also, for 4-MV macroblocks, the decision of whether the color-difference blocks are coded as Inter or Intra shall be made based on the inter-coded status of the luma blocks as defined in section 8.3.5.4.4. The following sections describe how to reconstruct the color-difference motion vectors for 1-MV and 4-MV macroblocks. The color-difference vectors are reconstructed in two steps.

1. As a first step, the nominal color-difference motion vector shall be obtained by combining and scaling the luma motion vectors appropriately as defined in sections 8.3.5.4.3 and 8.3.5.4.4.

   Note: The scaling is performed in such a way that half-pixel offsets are preferred over quarter pixel locations.
2. In the second step, the 1-bit syntax element FASTUVMC syntax element (**6.2.6**, Annex J) is used to determine if further rounding of color-difference motion vectors is necessary. If FASTUVMC == 0, no rounding shall be performed in the second stage. If FASTUVMC == 1, the color-difference motion vectors that are at quarter pel offsets shall be rounded to the nearest half or full pel positions as defined in Section 8.3.5.4.5.

   Note: The purpose of this mode is speed optimization of the decoder.

Only bilinear filtering shall be used for all color-difference interpolation (see section 8.3.6.5.1).

In the sections that follow, cmv_x and cmv_y denote the color-difference motion vector components and lmv_x and lmv_y denote the luma motion vector components.

8.3.5.4.3 First-stage Color-Difference Motion Vector Reconstruction – 1-MV Color-Difference Motion Vector Case:

In a 1-MV macroblock, the color-difference motion vectors shall be derived from the luma motion vectors as follows:

```
// s_RndTbl[0] = 0, s_RndTbl[1] = 0, s_RndTbl[2] = 0, s_RndTbl[3] = 1
cmv_x = (lmv_x  + s_RndTbl[lmv_x & 3]) >> 1
cmv_y = (lmv_y  + s_RndTbl[lmv_y & 3]) >> 1
```

where the luma motion vector is (lmv_x, lmv_y) and the chroma motion vector is (cmv_x, cmv_y), and s_RndTbl[] is the rounding value table defined below.

8.3.5.4.4 First-stage Color-difference Motion Vector Reconstruction – 4-MV Color-difference Motion Vector Case:

The pseudo-code of Figure 57 defines how the color-difference motion vectors shall be derived from the motion information in the 4 luma blocks in 4-MV macroblocks. In this section, ix and iy are temporary variables.

```
int ix, iy // local vars
if (all 4 luma blocks are Inter-coded)
{
      // lmv0_x, lmv0_y is the motion vector for block 0
      // lmv1_x, lmv1_y is the motion vector for block 1
      // lmv2_x, lmv2_y is the motion vector for block 2
      // lmv3_x, lmv3_y is the motion vector for block 3
      ix = median4(lmv0_x, lmv1_x, lmv2_x, lmv3_x)
      iy = median4(lmv0_y, lmv1_y, lmv2_y, lmv3_y)
}
else if (3 of the luma blocks are Inter-coded)
{
      // lmv0_x, lmv0_y is the motion vector for the first Inter-coded block
      // lmv1_x, lmv1_y is the motion vector for the second Inter-coded block
      // lmv2_x, lmv2_y is the motion vector for the third Inter-coded block
      ix = median3(lmv0_x, lmv1_x, lmv2_x)
      iy = median3(lmv0_y, lmv1_y, lmv2_y)
}
else if (2 of the luma blocks are Inter-coded)
{
      // lmv0_x, lmv0_y is the motion vector for the first Inter-coded block
      // lmv1_x, lmv1_y is the motion vector for the second Inter-coded block
      ix = (lmv0_x + lmv1_x) / 2
      iy = (lmv0_y + lmv1_y) / 2
}
else
      Color-difference blocks are coded as Intra

// s_RndTbl[0] = 0, s_RndTbl[1] =   0, s_RndTbl[2] = 0, s_RndTbl[3] = 1
cmv_x = (ix + s_RndTbl[ix & 3]) >> 1
cmv_y = (iy + s_RndTbl[iy & 3]) >> 1
```

Figure 57: Color-difference MV Reconstruction for Progressive: Pseudo-Code

8.3.5.4.5 Second Stage Color-difference Rounding

If the bit FASTUVMC == 1, then a second level of rounding shall be done on the color-difference motion vectors as follows:

```
      // RndTbl[-1] = +1, RndTbl[0] = 0, RndTbl[1] = -1
      cmv_x =  cmv_x + RndTbl[cmv_x % 2];
            cmv_y =       cmv_y + RndTbl[cmv_y % 2];
```

In the above, cmv_x and cmv_y represent the x and y coordinates of the color-difference motion vector in units of quarter pels. Thus, when cmv_x (or cmv_y) is divisible by 4, there is an integer pel offset; when cmv_x % 4 == +/-2, there is a half pel offset, and when cmv_x % 2 == +/-1 there is a quarter pel offset. As may be seen by the above re-

mapping operation, the quarter pel positions shall be disallowed by rounding the color-difference motion vector to the nearest integer or half pel position towards zero (half pel positions are left unaltered).

This forces the color-difference co-ordinates to be remapped to integer and half pel positions. Second stage rounding shall not be performed if FASTUVMC == 0.

8.3.5.5 Coded Block Pattern

Figure 25 shows the position of the CBPCY syntax element (7.1.3.1) within the P picture macroblock layer. The CBPCY syntax element is a variable-length code that decodes to a 6-bit syntax element.

The code table used to decode CBPCY is specified by the CBPTAB syntax element in the picture layer.

The CBPCY syntax element is used differently depending on whether the macroblock is 1-MV or 4-MV. The following sub-sections define how CBPCY is used for each macroblock type.

8.3.5.5.1 CBPCY in 1-MV Macroblocks

The CBPCY syntax element shall be present in the 1-MV macroblock layer if the MVDATA syntax element (7.1.3.8) indicates that at least one block contains coefficient information. This is indicated by the 'more_present flag' value decoded from MVDATA. See section 8.3.5.2 for a description of MVDATA decoding.

If the CBPCY syntax element is present, then it shall decode to a 6-bit syntax element indicating which of the blocks contains at least one non-zero coefficient. Table 70 shall define how the bit positions in the CBPCY syntax element correspond to the block numbers.

A '1' in one of the positions shall define that the corresponding block has at least one non-zero AC coefficient if the macroblock is Intra-coded or has at least one non-zero DC or AC coefficient if the macroblock is Inter-coded.

A '0' in one of the positions shall define that the corresponding block does not contain any non-zero AC coefficients if the macroblock is Intra-coded nor contain any non-zero DC or AC coefficients if the macroblock is Inter-coded.

8.3.5.5.2 CBPCY in 4-MV Macroblocks

The CBPCY syntax element shall always be present in the 4-MV macroblock layer. The CBPCY bit positions, defined in Table 70, for the luma blocks (bits 2-5) have a different meaning than the bit positions for color-difference blocks (bits 0 and 1) as defined next.

For the luma blocks:

A '0' defines that the corresponding block shall not contain motion vector information or any non-zero coefficients. In this case, the BLKMVDATA syntax element (7.1.3.11) shall not be present for that block and the predicted motion vector shall be used as the motion vector and there shall be no residual data. If the motion vector predictors indicate that hybrid motion vector prediction is used, then the HYBRIDPRED syntax element shall be present indicating the motion vector predictor candidate to use. Refer to section 8.3.5.3 for a description of computing the motion vector predictor.

A '1' defines that the BLKMVDATA syntax element shall be present for the block. The BLKMVDATA syntax element shall indicate whether the block is Inter or Intra-coded and whether coefficient data shall be present for the block. If it is Inter coded, the BLKMVDATA syntax element shall also contain the motion vector differential. If the **'more_present flag'** decoded from BLKMVDATA (described in section 8.3.5.2.1) decodes to 0, then no AC coefficient information shall be present if the block is Intra-coded or no DC or AC coefficient shall be present if the block is Inter-coded. If the **'more_present flag'** decodes to 1, then there shall at least be one non-zero AC coefficient if the block is Intra-coded or at least one non-zero DC or AC coefficient if the block is Inter-coded.

For the color-difference blocks:

A '0' defines that the block shall not contain any non-zero AC coefficients if the block is Intra-coded or any non-zero DC or AC coefficients if the block is Inter-coded.

A '1' defines that the corresponding block shall have at least one non-zero AC coefficient if the block is Intra-coded or at least one non-zero DC or AC coefficient if the block is Inter-coded.

8.3.5.6 MB-level Transform Type

The TTMB syntax element (7.1.3.10) is present only in Inter macroblocks. As described in section 7.1.3.10 TTMB codes the transform type, the signaling mode and the transform subblock pattern.

If the signaling mode is macroblock signaling, then the transform type decoded from the TTMB syntax element shall be the same for all blocks in the macroblock. If the transform type is 8x4 or 4x8, then the subblock pattern shall also be decoded from the TTMB syntax element. In this case, the subblock pattern shall apply to the first coded block in the macroblock. If the transform type is 4x4, then the subblock pattern is decoded in the SUBBLKPAT syntax element (7.1.4.17) at the block level (in each coded block). If the transform type is 8x4 or 4x8, then the subblock patterns for all the coded blocks after the first one shall be decoded in the SUBBLKPAT syntax element at the block level.

If the signaling mode is block signaling, then the transform type decoded from the TTMB syntax element shall be applied to the first coded block in the macroblock and the TTBLK syntax element shall not be present for the first coded block. For the remaining coded blocks, the TTBLK syntax element shall indicate the transform type for that block. If TTMB syntax element indicates that the first transform type is 8x4 or 4x8, then a subblock pattern shall also be decoded from the TTMB syntax element. In this case, the subblock pattern shall apply to the first coded block in the macroblock. If the transform type is 4x4 for a block, then the subblock pattern shall be decoded in the SUBBLKPAT syntax element at the block.

8.3.6 Block Layer Decode

This section defines the Block layer decoding for progressive P-frame pictures for simple, main and advanced profiles.

8.3.6.1 Intra Coded Block Decode

The process for decoding Intra blocks in P pictures shall be identical to the process for decoding Intra blocks in I pictures defined with section 8.1.3 with the following differences.

8.3.6.1.1 Coefficient Scaling

The process for coefficient scaling is identical to the process defined in section 8.1.3.9.

8.3.6.1.2 DC Prediction

The process of DC prediction is defined in section 8.1.3.2. The procedure specified in Figure 40 shall be used for calculating the DC predictor and the prediction direction. If the macroblock quantizer is different from those of the predictor blocks, the DC components shall be scaled as specified in section 8.1.3.9 before selection of the DC predictor and the prediction direction.

8.3.6.1.3 AC Prediction in Intra blocks in 4-MV macroblocks.

Refer to section 8.1.3.7 for a description of AC prediction. AC prediction in Intra-coded blocks within 4-MV macroblocks is identical except for the differences defined in the remainder of this section.

If the top predictor is selected, then the top row of AC coefficients from the block above the current block shall be used as the predictors for the top row of AC coefficients from the current block. If the left predictor is selected, then the first column of AC coefficients from the block to the left of the current block shall be used as the predictors for the left column of AC coefficients from the current block. As defined in section 7.1.2, with respect to prediction, the first row of macroblocks in the slice shall be considered to be the first row of macroblocks in the picture.

The pseudo-code of Figure 58 shall specify the process for deciding the AC predictors in Intra blocks in 4-MV macroblocks.

In the pseudo-code, predictorA is the block located immediately above the current block, predictorC is the block located immediately to the left of the current block and predictorB is the block immediately above and to the left of the current block. The result of the pseudo-code is that the variable *use_ac_prediction* shall determine whether AC prediction is used and *prediction_direction* shall determine which block is used as the predictor.

Note: In the pseudo-code, the coefficients in the predictor blocks are scaled if the macroblock quantizer scales are different. Section 8.1.3.9 defines the scaling operation.

```
// prediction_direction TOP refers to the top predictor
// prediction_direction LEFT refers to the left predictor
use_ac_prediction = FALSE
is_nonzero_predictor = FALSE
if ((predictorA is Intra) && (predictorC is Intra))
{
    is_nonzero_predictor = TRUE;
    if (predictorB is not intra)
    {
        set predictorB's DC coefficient to be the default predictor which is zero.
    }

     if (abs(predictorB's DC coefficient – predictorC's DC coefficient) <
        abs(predictorB's DC coefficient – predictorA's DC coefficient))
     {
        prediction_direction = TOP
     }
     else
        prediction_direction = LEFT
}
else if ((predictorA is Intra) || (predictorC is Intra))
{
     is_nonzero_predictor = TRUE
     if (predictorA is Intra)
        prediction_direction = TOP
     else
        prediction_direction = LEFT
}
```

Figure 58: Calculating DC Predictor Direction: Pseudo-Code

After all the (up to six) predictors are computed (for the up to six intra blocks in the 4-MV macroblock), and only when at least one of the blocks has the flag is_nonzero_predictor set to the value TRUE, the one bit element defining ACPRED shall be decoded.

```
if (get_bits(1) == 1)          // ACPRED syntax element
    use_ac_prediction = TRUE
```

The rule to apply AC prediction is described in the next section.

8.3.6.1.4 AC Prediction in Intra blocks in 1-MV macroblocks

AC prediction in Intra blocks within 1-MV macroblocks shall be the same as AC prediction in Intra blocks of I pictures as defined in section 8.1.3.7, except as follows. If the top predictor block and left predictor block are not Intra-coded, then AC prediction shall not be used, even if ACPRED == 1 in the macroblock layer. If just one of the predictors is Intra coded (either the top or the left), then it shall be used as the predictor. If both are Intra-coded, then the method described in section 8.1.3.7 shall be used. In this case, if the top-left block is not Intra, then the DC value of the top-left block shall be assumed to be 0.

8.3.6.1.5 Zigzag Scan

The zigzag scan order used to scan the run-length decoded Transform coefficients into the 8x8 array shall be the same as that used for the 8x8 Inter block as described in section 8.3.6.2.5. This differs from Intra blocks in I pictures which use one of three zigzag scans depending on the prediction direction. The process of scanning out coefficients is defined in Figure 43.

8.3.6.1.6 Coding Sets

The TRANSACFRM syntax element (**7.1.1.11**) shall be used to specify the coding set index used for decoding the Y, C_b, and C_r AC coefficients (see section 8.1.3.4 for a definition of the AC coding sets). The index decoded from the TRANSACFRM syntax element shall be used to select the intra coding set used to decode the Y blocks and shall be used to select the inter coding set used to decode the C_b and C_r blocks. This differs from the process used for I pictures where the TRANSACFRM specifies the index for the inter coding set and the TRANSACFRM2 syntax element (**7.1.1.12**) specifies the index for the intra coding set. The P picture header shall not contain the TRANSACFRM2 syntax element. The correspondence between the coding set index and the coding set depends on the value of PQINDEX. Table 76 and Table 77 below shall specify the correspondence for PQINDEX <= 8 and PQINDEX > 8 (**7.1.1.6**). Section 11.8 contains the table information that shall be used.

Table 76: Index/Coding Set Correspondence for PQINDEX <= 8

Y blocks		C_b and C_r blocks
Index	Table	Table
0	High Rate Intra (Table 219-Table 225)	High Rate Inter (Table 226- Table 232)
1	High Motion Intra (Table 177-Table 183)	High Motion Inter (Table 184-Table 190)
2	Mid Rate Intra (Table 205-Table 211)	Mid Rate Inter (Table 212-Table 218)

Table 77: Index/Coding Set Correspondence for PQINDEX > 8

Y blocks		C_b and C_r blocks
Index	Table	Table
0	Low Motion Intra (Table 191-Table 197)	Low Motion Inter (Table 198-Table 204)
1	High Motion Intra (Table 177-Table 183)	High Motion Inter (Table 184-Table 190)
2	Mid Rate Intra (Table 205-Table 211)	Mid Rate Inter (Table 212-Table 218)

8.3.6.1.7 Inverse Transform

After reconstruction of the transform coefficients, the resulting 8 × 8 blocks shall be processed by a separable two-dimensional inverse transform of size 8 by 8, and shall comply with Annex A. The inverse transform output has a

dynamic range of 10 bits, as a signed integer. Finally, the constant value of 128 shall be added to the reconstructed intra block and the result shall be clamped to the range [0 255] and shall form the reconstruction prior to loop filtering.

8.3.6.2 Inter Coded Block Decode

Figure 59 shows the steps required reconstructing Inter blocks. For illustration, the figure shows the reconstruction of a block whose 8x8 error signal is coded with two 8x4 transforms. The 8x8 error block may also be transformed with two 4x8 Transforms or one 8x8 transform. The steps required to reconstruct an inter-coded block include: 1) transform type selection (8.3.6.2.1), 2) sub-block pattern decode (8.3.6.2.2), 3) coefficient decode (8.3.6.2.3, 8.3.6.2.4, 8.3.6.2.5, 8.3.6.3), 4) inverse transform (8.3.6.4), 5) obtain predicted block by motion compensation (8.3.6.5.1, 8.3.6.5.2) and 6) adding predicted and error blocks (8.3.6.5.3). The following sections define these steps.

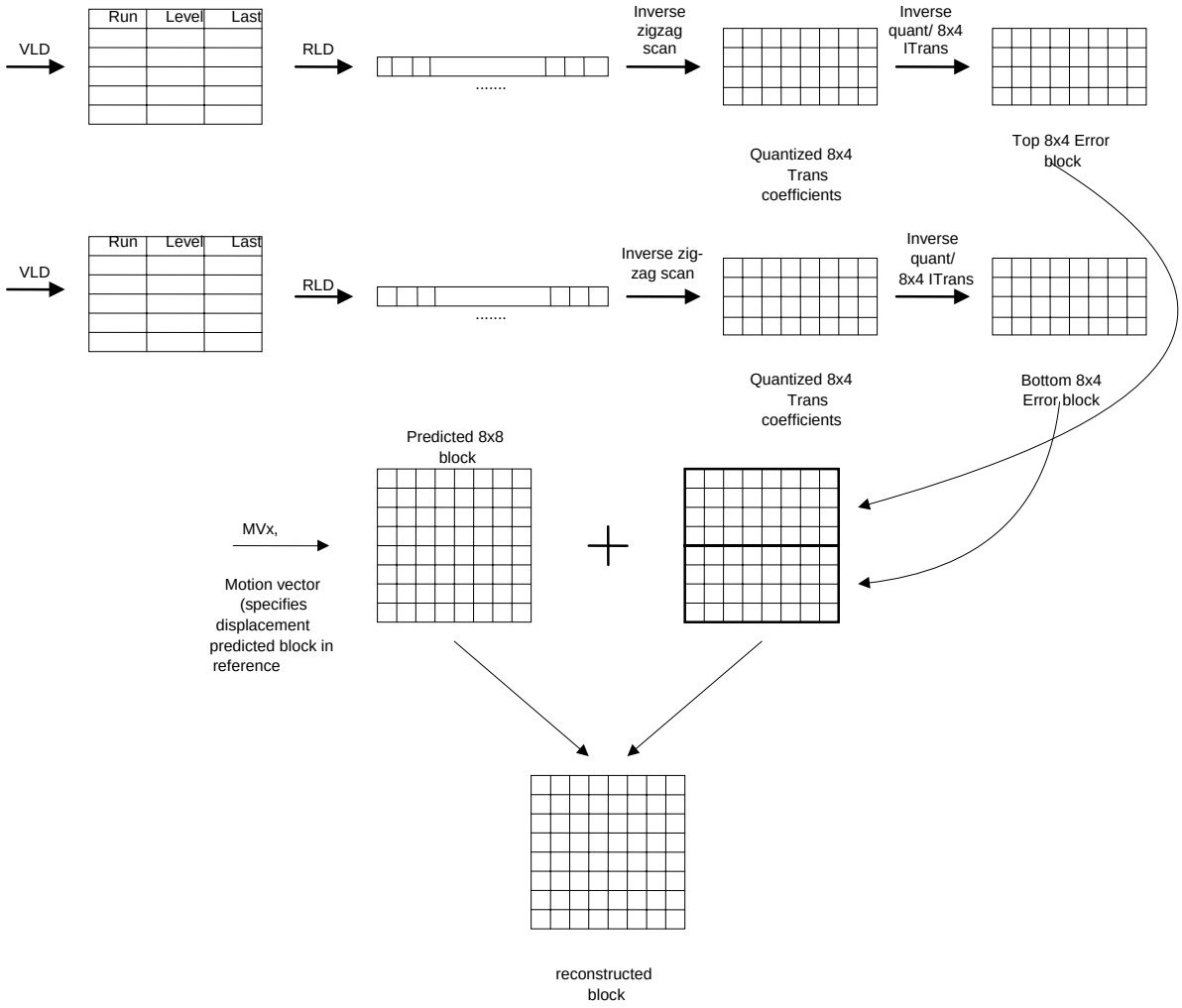


Figure 59: Inter block reconstruction

Note: The steps shown in Figure 59 for inter block reconstruction apply to interlace field P pictures as well as progressive P pictures.

8.3.6.2.1 Transform Type Selection

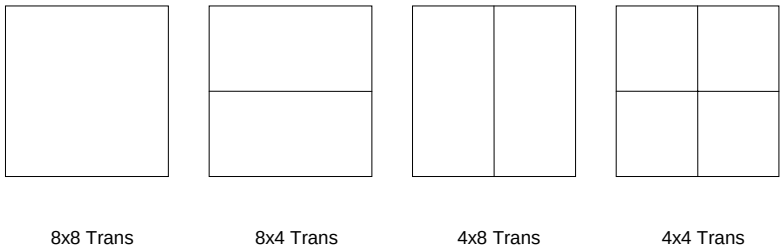


Figure 60: Transform Types

If variable-sized transform coding is enabled (signaled by the syntax element VSTRANSFORM == 1) (**6.2.9**, Annex J.1.14), then the 8x8 error block may be transformed using one 8x8 transform, it may be divided horizontally and transformed with two 8x4 transforms or divided vertically and transformed with two 4x8 transforms or divided into 4 quadrants and transformed with 4 4x4 transforms as shown in Figure 60. The transform type is signaled at the picture, macroblock or block level. If TTMB (**7.1.3.10**) indicates that the signal level is 'Block', then the transform type shall be signaled at the block level and the TTBLK syntax element (**7.1.4.16**) shall be present within the bitstream as shown in Figure 28. This syntax element shall define the transform type used for the block as indicated by **7.1.4.16** (Table 62, Table 63 and Table 64).

If variable-sized transform coding is not enabled, then the 8x8 transform shall be used for all blocks.

8.3.6.2.2 Subblock Pattern Decode

If the transform type is 8x4, 4x8 or 4x4, then the decoder shall be given information about which of the subblocks have non-zero coefficients. For 8x4 and 4x8 transform types, the subblock pattern shall be decoded as part of the TTMB or TTBLK syntax element. If the transform type is 4x4, then the SUBBLKPAT syntax element (**7.1.4.17**) shall be present in the bitstream as shown in Figure 28.

If the subblock pattern indicates that no non-zero coefficients are present for the subblock, then no other information for that subblock shall be present in the bitstream. For the 8x4 transform type, the data for the top subblock (if present) shall be coded first followed by the bottom subblock. For the 4x8 transform type, the data for the left subblock (if present) shall be coded first followed by the right subblock. For the 4x4 transform type, the data for the upper left subblock shall be coded first followed, in order, by the upper right, lower left and lower right subblocks.

8.3.6.2.3 Coefficient Bitstream Decode

The first step in reconstructing the inter-coded block is to reconstruct the transform coefficients. The process for decoding the bitstream to obtain the run, level and last_flag for each non-zero coefficient in the block or sub-block is nearly identical to the process described in section 8.1.3.4 for decoding the AC coefficients in intra blocks. The two differences are:

1) Unlike the decoding process for intra blocks, the DC coefficient is not differentially coded. No distinction is made between the DC and AC coefficients and all coefficients shall be decoded using the same method (defined in section 8.1.3.4).

2) Unlike the decoding process for intra blocks in I pictures (defined in section 8.1.3.4) where the Y block coefficients are decoded using one of the three intra coding sets and the C_b and C_r block coefficients are decoded using one of the three inter coding sets, the Y, C_b and C_r inter blocks shall all use the same inter coding set (defined in Table 78 and Table 79).

3) The correspondence between the coding set index and the coding set depends on the value of PQINDEX. Table 78 and Table 79 shall define the correspondence for PQINDEX <= 8 and PQINDEX > 8 respectively.

Table 78: Index/Coding Set Correspondence for PQINDEX <= 8

Y, C_b and C_r blocks	
Index	Table
0	High Rate Inter

	(Table 226- Table 232)
1	High Motion Inter (Table 184-Table 190)
2	Mid Rate Inter (Table 212-Table 218)

Table 79: Index/Coding Set Correspondence for PQINDEX > 8

Y, C_b and C_r blocks	
Index	Table
0	Low Motion Inter (Table 198-Table 204)
1	High Motion Inter (Table 184-Table 190)
2	Mid Rate Inter (Table 212-Table 218)

8.3.6.2.4 Run-level Decode

The process for decoding the run-level pairs obtained in the coefficient decoding process described above shall be the same as described in section 8.1.3.5, except as described here. The difference is that because all coefficients are run-level encoded (not just the AC coefficients as in intra blocks) the run-level decode process produces a 16-element array in the case of 4x4 Transform, a 32-element array in the case of 8x4 or 4x8 Transform blocks or a 64-element array in the case of 8x8 Transform blocks.

8.3.6.2.5 Zigzag Scan of Coefficients

The one-dimensional array of quantized coefficients produced in the run-level decode process described above shall be scanned out into a two-dimensional array in preparation for the Inverse Transform. The process shall be the same as that described in section 8.1.3.6 for intra blocks, except:

1) Each Transform type shall have an associated zigzag scan array.

2) The zigzag scan arrays for some transform types are different between the interlace mode and progressive mode of the advanced profile.

3) Unlike the zigzag scanning process for intra blocks where one of three arrays is used depending on the DC prediction direction, only one array (as defined below) shall be used for inter blocks.

The zigzag scan arrays for Inter blocks in simple and main profiles shall be as defined by Table 236 to Table 239, respectively. The scan arrays in the advanced profile depend on whether interlace or progressive mode is used. The process of scanning out coefficients is defined in Figure 43.

In progressive mode of advanced profile, the scan arrays for Inter 8x8 and 4x4 blocks shall be identical to those for simple and main profiles, and shall be defined by Table 236 and Table 239 respectively. The scan arrays for Inter 8x4 and 4x8 blocks shall be as defined by Table 240 and Table 241 respectively. In interlaced mode of the advanced profile, the scan arrays for Inter 8x8, 8x4, 4x8 and 4x4 blocks shall be as defined by Table 242 to Table 245 respectively.

8.3.6.3 Inverse Quantization

The non-zero quantized coefficients reconstructed as described in the sections above shall be inverse quantized as described in section 8.1.3.8.

8.3.6.4 Inverse Transform

After reconstruction of the transform coefficients, the resulting 8x8, 8x4, 4x8 or 4x4 blocks shall be processed by the appropriate two-dimensional inverse transforms. The 8x8 blocks shall be transformed using the 8x8 two-dimensional inverse transform, the 8x4 blocks shall be transformed using the 8x4 two-dimensional inverse transform, the 4x8 blocks shall be transformed using the 4x8 two-dimensional inverse transform and the 4x4 blocks shall be transformed using the 4x4 two-dimensional inverse transform.

The two-dimensional inverse transform shall comply with Annex A.

8.3.6.5 Motion Compensation

The 8x8, 8x4, 4x8 or 4x4 error block or blocks shall be added to the predicted 8x8 block to produce the reconstructed block. The motion vector decoded in the macroblock header (described in section 8.3.5.2) shall be used to obtain the predicted block in the reference frame.

The horizontal and vertical motion vector components represent the displacement between the block currently being decoded and the corresponding location in the reference frame. Positive values represent locations that are below and to the right of the current location. Negative values represent locations that are above and to the left of the current location. The actual reconstructed motion vector shall be used by subsequent macroblocks as a reference for motion vector predictor calculation.

For simple and main profile, the reconstructed motion vectors points shall be adjusted as necessary prior to motion compensation as defined by the pseudo-code in Figure 61 for luma motion vectors, and Figure 62 for color-difference motion vectors.

The following operation shall be applied to adjust the luma motion vector for simple and main profile:

Let iXCoord, iYCoord be the spatial location of the top left corner of the current macroblock (e.g. if the current macroblock is located on the 3rd column and 2nd row, then iXCoord = 2 * 16 and iYCoord = 1 * 16).

Let iMvX, iMvY be the reconstructed luma motion vector in quarter pixel unit.

Let iNumMBX, iNumMBY be the number of macroblocks in a row and a column, respectively.

Then the adjusted luma motion vector iMvXComp, iMvYComp for motion compensation shall be computed according to the pseudo-code of Figure 61.

```
int iPosX = iXCoord + (iMvX >> 2);
int iPosY = iYCoord + (iMvY >> 2);
iMvXComp = iMvX;
iMvYComp = iMvY;
if   (iPosX < -16) {
       iMvXComp = ((-16 – iXCoord)<<2) + (iMvX & 3);
} else if (iPosX > iNumMBX * 16) {
       iMvXComp = ((iNumMBX * 16 – iXCoord)<<2) + (iMvX & 3);
}
if   (iPosY < -16) {
       iMvYComp = ((-16 – iYCoord)<<2) + (iMvY & 3);
} else if (iPosY > iNumMBY * 16) {
       iMvYComp = ((iNumMBY * 16 – iYCoord)<<2) + (iMvY & 3);
           }
```

Figure 61: Adjusting Reconstructed Luma Motion Vector in Simple/Main Profile

Similarly, the following operation shall be applied to adjust the color-difference motion vector for simple and main profile:

Let iCMvX, iCMvY be the reconstructed color-difference motion vector.

Let iCXCoord, iCYCoord be the spatial location of the top left corner of the current color-difference block (e.g. if the current block is located on the 3rd column and 2nd row, then iCXCoord = 2 * 8 and iCYCoord = 1 * 8).

Then the adjusted color-difference motion vector iCMvXComp, iCMvYComp for motion compensation shall be computed according to the pseudo-code of Figure 62.

```
    iPosX = iCXCoord + (iCMvX >> 2);
    iPosY = iCYCoord + (iCMvY >> 2);
    iCMvXComp = iCMvX;
    iCMvYComp = iCMvY;
    if   (iPosX < -8) {
         iCMvXComp = ((-8 – iCXCoord)<<2) + (iCMvX & 3);
    } else if (iPosX > iNumMBX * 8) {
         iCMvXComp = ((iNumMBX * 8 – iCXCoord)<<2) + (iCMvX & 3);
}
    if   (iPosY < -8) {
         iCMvYComp = ((-8 – iCYCoord)<<2) + (iCMvY & 3);
    } else if (iPosY > iNumMBY * 8) {
         iCMvYComp = ((iNumMBY * 8 – iCYCoord)<<2) + (iCMvY & 3);
            }
```

Figure 62: Adjusting Reconstructed Color-difference Motion Vector in Simple/Main Profile

If the picture layer syntax element MVMODE (see section 7.1.1.32) indicates that 1-MV Halfpel or 1-MV Halfpel Bilinear is used as the motion compensation mode, then the decoded motion vector differential shall be expressed in half-pixel resolution. The motion vector reconstruction process described in these sections assumes that the delta motion vector and the predicted motion vector are in quarter pixel units. Therefore, for the case where the MVMODE indicates that the motion vector use half-pixel precision, the delta motion vectors shall be multiplied by 2 to obtain the equivalent quarter pixel representation.

For example, a horizontal motion component of 4 indicates a position 2 pixels to the right of the current position and a value of 5 indicates a position of 2 ½ pixels to the right. If the picture layer syntax element MVMODE (see section 7.1.1.32) indicates that 1-MV or Mixed-MV is used as the motion compensation mode, then all motion vectors shall be expressed in quarter-pixel resolution. For example, a horizontal motion component of 4 indicates a position 1 pixel to the right of the current position and a value of 5 indicates a position of 1 1/4 pixels to the right.

Integer pixel motion vectors do not require the computation of interpolated pixels. In 1-MV Halfpel Bilinear mode, all non-integer pixel motion vector offsets shall use a bilinear filter to compute the interpolated pixels. In all other modes, all non-integer pixel motion vector offsets shall use a bicubic filter to compute the interpolated pixels. The repeat pad operation shall be performed before the subpixel interpolation.

8.3.6.5.1 Bilinear Interpolation

The following sections describe the bilinear filter operations. The bilinear filter shall operate as shown in Figure 63.

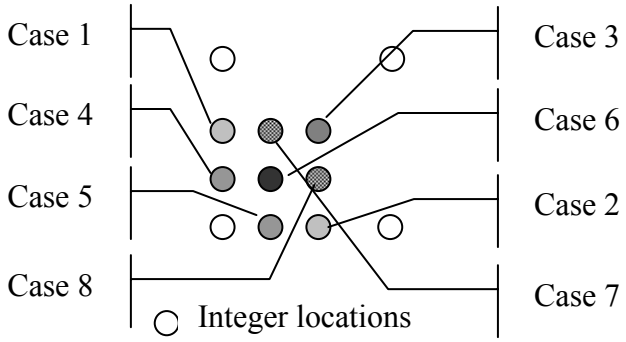


Figure 63: Bilinear filter cases

Figure 63 shows all the possible unique interpolated positions. They are:

Case 1: full-pel horizontal, half-pel vertical

Case 2: half-pel horizontal, full-pel vertical

Case 3: half-pel horizontal, half-pel vertical

Case 4: full-pel horizontal, quarter-pel vertical

Case 5: quarter-pel horizontal, full-pel vertical

Case 6: quarter-pel horizontal, quarter-pel vertical

Case 7: quarter-pel horizontal, half-pel vertical

Case 8: half-pel horizontal, quarter-pel vertical

Note: The other possible interpolated positions are identical to one of the cases 1-8, and hence are not shown in Figure 63.

Although the bilinear interpolator is defined for quarter pixel motion vector resolution, only half pixel motion shall be allowed for the luma blocks. In other words, in Figure 63, only cases 1, 2 and 3 shall be permitted for luma. Cases 4 through 8 shall be used only for color-difference.

For a non-integer pixel motion vector value, the value of the interpolated pixel shall be computed from the four closest integer pixel motion vector locations in the reference picture. In Figure 63, the relative positions of these reference integer pixel locations are: *a* (bottom-left), *b* (top-left), *c* (top-right), *d* (bottom-right).

For the cases 1, 2 and 3, the interpolated pixel *p* shall be derived by the following equations:

$p = (a + b + 1 - RND) >> 1$:case 1

$p = (a + d + 1 - RND) >> 1$:case 2

$p = (a + b + c + d + 2 - RND) >> 2$:case 3

where *RND* is the frame level rounding control value as described in section 8.3.7.

The general rule that applies to all cases is shown below. The indices *x* and *y* are the sub-pixel shifts in the horizontal (left to right) and vertical (bottom to top) directions, multiplied by 4. Their values range from 0 through 4 within the area bounded by the four pixels shown in Figure 63, with the origin located at *a*. Arrays F and G are the filter coefficients. F[] = { 4, 3, 2, 1, 0 } and G[] = { 0, 1, 2, 3, 4 }. The interpolated value *p* shall be given by:

$p = (F[x] F[y] a + F[x] G[y] b + G[x] G[y] c + G[x] F[y] d + 8 - RND) >> 4$

For example, consider case 8. The subpixel shifts for case 8 are $x=2$, $y=1$. *p* is therefore:

$p = (6 a + 2 b + 2 c + 6 d + 8 - RND) >> 4$

The above expression is identical to

$p = (3 a + b + c + 3 d + 4 - RND) >> 3$

when *RND* is 0 or 1, which is the case with the rounding control value.

Note: It can be shown that cases 1 through 3 simplify to their earlier definitions.

8.3.6.5.2 Bicubic Interpolation

The following section describes the bicubic filter operations.

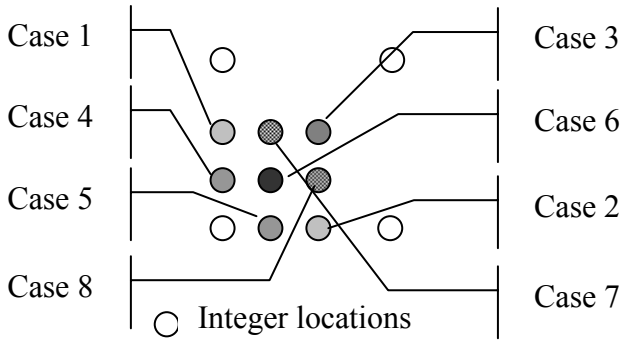


Figure 64: Bicubic filter cases

Figure 64 shows all the possible unique interpolated positions. They are:

Case 1: full-pel horizontal, half-pel vertical

Case 2: half-pel horizontal, full-pel vertical

Case 3: half-pel horizontal, half-pel vertical

Case 4: full-pel horizontal, quarter-pel vertical

Case 5: quarter-pel horizontal, full-pel vertical

Case 6: quarter-pel horizontal, quarter-pel vertical

Case 7: quarter-pel horizontal, half-pel vertical

Case 8: half-pel horizontal, quarter-pel vertical

Note: The other possible interpolated positions are identical to one of the cases 1-8, and hence are not shown in Figure 64.

In Figure 64, the relative position of the integer pixel locations: *a* (bottom-left), *b* (top-left), *c* (top-right), *d* (bottom-right).

Bicubic Filter Coefficients for different shift locations

Figure 65 shows the pixels that shall be used to compute the interpolated pixels for each case. S denotes the sub-pixel position. P1, P2, P3 and P4 represent the integer pixel positions. The figure shows horizontal interpolation but the same operation shall apply to vertical interpolation.

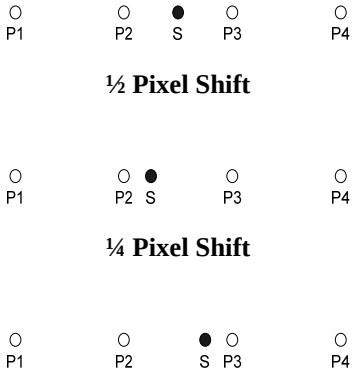


Figure 65: Pixel Shifts

The following filters shall be used for the possible shift locations of interpolated pixel:

½ pel shift F1: [-1 9 9 -1]

¼ pel shift F2: [-4 53 18 -3]

¾ pel shift F3: [-3 18 53 -4]

The following equations shall specify the filtered result, S, for the possible shift locations of the interpolated pixel:

S = (-1*P1 + 9*P2 + 9*P3 -1*P4) (1/2 pixel shift)

S =(-4*P1 + 53*P2 + 18*P3 – 3*P4) (1/4 pixel shift)

S = (-3*P1 + 18*P2 + 53*P3 -4*P4) (3/4 pixel shift)

One-dimensional Bicubic Interpolation (Cases 1, 2, 4 and 5)

In Figure 64, cases 1, 2, 4 and 5 represent the cases where interpolation occurs in only one dimension – either horizontal or vertical

The following rounding shall be applied after the filtering operation for each case:

(S + 8 – *r*) >> 4 (1/2 pixel shift)

(S + 32 – *r*) >> 6 (1/4 pixel shift)

(S +32 – *r*) >> 6 (3/4 pixel shift)

where S is the filtered result after applying the filter appropriate for the location of the interpolated pixel.

The value *r* in the equations above depends on *RND*, the frame-level round control value (see section 8.3.7 for a description) and the interpolation direction as follows:

$$r = \begin{cases} 1 - RND & (vertical\ direction - cases\ 1\ and\ 4) \\ RND & (horizontal\ direction - cases\ 2\ and\ 5) \end{cases}$$

Two-dimensional Bicubic Interpolation

In Figure 64, cases 3, 6, 7 and 8 are the cases where interpolation occurs in both the horizontal and vertical directions.

Two-dimensionally interpolated pixel locations shall first interpolate along the vertical direction, and then along the horizontal direction using the appropriate filter in each direction among F1, F2 and F3 specified above. Rounding

shall be applied separately after vertical filtering and after horizontal filtering. The output of the vertical filtering after rounding shall be used as the input for the horizontal filtering.

For example, the two-dimensional interpolation for case 8, which has a quarter-pel shift vertically and a half-pel shift horizontally, uses the filter F2 for interpolation along the vertical direction and F1 for interpolation along the horizontal direction.

The rounding rule after vertical filtering shall be defined as

$$(S + rndCtrlV) >> shiftV$$

where

S = vertically filtered result after applying the filter appropriate for the vertical shift location of the interpolated pixel, i.e. -1*P1 + 9*P2 + 9*P3 -1*P4 for ½ pixel shift

shiftV = { 1, 5, 3, 3 } for cases 3, 6, 7 and 8 respectively.

$rndCtrlV = 2^{shiftV-1} - 1 + RND$ (see section 8.3.7 for a description of *RND*)

The rounding rule after horizontal filtering shall be defined as:

$$(S + 64 - RND) >> 7.$$

where

S = horizontally filtered result after applying the filter appropriate for the horizontal shift location of the interpolated pixel,

RND = frame level round control value (see section 8.3.7)

All of the bicubic filtering cases potentially produce an interpolated pixel whose value is negative, or larger than the maximum range (255). In these cases, the clip( ) operator shall be applied so that the output lies within the range.

8.3.6.5.3 Adding Error and Predictor

The 8x8 predicted block is added to the 8x8 error block to form the reconstructed 8x8 block. The pseudo-code in Figure 66 shall specify this process.

```
// predblock[8][8] represents 8x8 predicted block
// errorblock[8][8] represents 8x8 error block
// reconblock[8][8] represents reconstructed 8x8 block
for (row= 0; row < 8; row++)
{
  for (col = 0; col < 8; col++)
        reconblock[row][col] = clip(predblock[row][col] + errorblock[row][col])
}
```

Figure 66: Inter block reconstruction pseudo-code

8.3.7 Rounding Control (RND)

Section 8.3.6.2 defines the interpolation operations used to generate subpixel values in the reference blocks. Rounding is controlled by a value *RND* called the rounding control value.

In simple and main profiles, *RND* is derived as follows:

- For each I or BI frame, the value of *RND* shall be set to 1.

- For each P Frame, the value of *RND* shall toggle back and forth between 0 and 1. Thus, the value of *RND* for the first P frame following an I frame is 0.
- For each B frame or a skipped frame, the value of *RND* shall remain the same as that of the closest I, BI or P frame which was decoded prior to the B frame.

In advanced profile, the value of *RND* shall be decoded from the RNDCTRL syntax element in the picture header.

8.3.8 Intensity Compensation

If the picture layer syntax element MVMODE (7.1.1.32) indicates that intensity compensation is used for the frame, then the pixels in the reference frame shall be remapped prior to using them as predictors for the current frame. As defined by section 8.3.4.3, when intensity compensation is used, the LUMSCALE (**7.1.1.34**) and LUMSHIFT (**7.1.1.35**) syntax elements are present in the picture bitstream. The pseudo-code of Figure 67 shall define how the LUMSCALE and LUMSHIFT values shall be used to build the lookup table used to remap the reference frame pixels.

```
// As defined in section 8.3.8, LUTY and LUTUV are lookup tables used to
//   remap the Y component, and the Cb and Cr component of the reference frame.

int iScale, iShift
if (LUMSCALE == 0)
{
      iScale = - 64
      iShift = 255 * 64   - LUMSHIFT *2 * 64
                  if (LUMSHIFT > 31)
                       iShift += 128 * 64;
}
else {
      iScale = LUMSCALE + 32
      if (LUMSHIFT > 31)
            iShift = LUMSHIFT * 64 - 64 * 64;
               else
            iShift = LUMSHIFT * 64;
          }

// build LUTs
for (i = 0; i < 256; i++)
{
      j = (iScale * i + iShift + 32) >> 6
      if (j > 255)
            j = 255
      else if (j < 0)
                        j = 0
      LUTY[i] = j
      j = (iScale * (i - 128) + 128 * 64 + 32) >>6
      if (j > 255)
            j = 255
      else if (j < 0)
            j = 0
      LUTUV[i] =   j
}
```

Figure 67: Intensity Compensation pseudo-code

The Y component of the reference frame shall be remapped using the LUTY[] table generated above, and the C_b and C_r components of the reference frame shall be remapped using the LUTUV[] table as follows:

$$\overline{p}_Y = LUTY[p_Y]$$

$$\overline{p}_{UV} = LUTUV[p_{UV}]$$

Where p_Y is the original luma pixel value in the reference frame and $\overline{p}_Y$ is the remapped luma pixel value in the reference frame and p_{UV} is the original C_b or C_r pixel value in the reference frame and $\overline{p}_{UV}$ is the remapped C_b or C_r pixel value in the reference frame. The remapped reference frame shall also be used for prediction of all subsequent B frames.

8.4 Progressive B Frame Picture Decoding

B frames are coded as bidirectional predicted frames and both forward and backward frames are needed for motion compensation. Progressive BI frame pictures are defined above in section 8.2. B Frames are permitted only in main and advanced profile.

Unlike P frames there shall be no "4-MV" motion compensation mode in B Frames. At the frame level, only two choices for motion vector resolution are permitted – quarter pel bicubic and half pel bilinear.

The following sections describe the process for decoding progressive B frame pictures. Section 8.4.1 defines the handling of skipped anchor frames, section 8.4.2 defines the handling of out-of-bounds reference pixels, section 8.4.3 defines the B picture types, section 8.4.4 defines the picture layer decoding, section 8.4.5 defines macroblock layer decoding, and section 8.4.6 defines block layer decoding.

8.4.1 Skipped Anchor Frames

If an anchor frame is coded as a skipped frame then it shall be treated as a P frame which is identical to its reference frame. In this case, both anchor frames shall be identical for the intervening B frames. For example, if the frames are coded as follows in display order:

I0 B1 P2 B3 P4 B5 S6 (I0 P2 B1 P4 B3 S6 B5 in coding order) where S6 is the skipped frame

then this is treated effectively as:

I0 B1 P2 B3 P4 B5 P4

because the skipped frame (S6) is treated as being identical to its reference (P4).

The motion vectors for the skipped anchor frame shall be set to zero.

Note: This is used for computation of direct mode motion vectors when the subsequent anchor is a skipped frame.

See section 8.3.1 for a description of skipped frames.

8.4.2 Out-of-bounds Reference Pixels

These shall be treated the same way as in progressive P frame pictures. Refer to section 8.3.2.

8.4.3 Progressive B Frame Picture Types

All B pictures are 1-MV type. Mixed-MV shall not be used with progressive B pictures.

8.4.4 Progressive B Frame Picture Layer Decode

Some B frame specific information is transmitted at the picture level. Apart from the syntax element PTYPE (**7.1.1.4**), BFRACTION syntax element (7.1.1.14) shall be present in the picture header. For main profile BFRACTION shall indicate whether it is a BI frame, or the scaling factor used to derive the direct motion vectors (explained in section

7.1.1.14). For advanced profile, BFRACTION shall only indicate the scaling factor. The details of B picture layer decoding are provided in the following sub-sections.

8.4.4.1 Picture-level Quantizer Scale (PQUANT)

This shall be identical to P pictures defined in 8.3.4.1.

8.4.4.2 Motion Compensation Mode (MVMODE)

The MVMODE syntax element (**7.1.1.32**) shall only take the values 0 or 1 in progressive B pictures. If MVMODE is 1, then the picture shall use 1-MV with quarter pel bicubic motion compensation and if it is 0 then it shall use 1-MV with half pel bilinear motion compensation. Intensity compensation shall not be signaled in B pictures (LUMSCALE, LUMSHIFT are not present).

8.4.4.3 Skipped Macroblock Decoding (SKIPMB)

The B picture layer contains the SKIPMB syntax element (**7.1.1.37**), and this shall be signaled the same way as with P pictures as defined in section 8.3.4.4. When a macroblock is skipped, then it may only contain the BMVTYPE syntax element to signal the prediction type (i.e. forward, backward, or interpolated is signaled by BMVTYPE).

Note: Hybrid MVs (HYBRIDPRED) are not used in B pictures.

8.4.4.4 Motion Vector Table (MVTAB)

These shall be identical to P pictures as defined in section 8.3.4.5.

8.4.4.5 Coded Block Pattern Table (CBPTAB)

The table used to decode the coded block pattern is indicated by the CBPTAB syntax element as described in section **7.1.1.39**, and its use shall be identical to P pictures as defined in section 8.3.5.5.

8.4.4.6 Macroblock-level Transform Type Flag (TTMBF)

This shall be identical to P pictures as defined in section 8.3.4.7.

8.4.4.7 Frame-level Transform Type (TTFRM)

This shall be identical to P pictures as defined in section 8.3.4.8.

8.4.4.8 Frame-level Transform AC Coding Set Index (TRANSACFRM)

This shall be identical to P pictures as defined in section 8.3.4.9.

8.4.4.9 Intra Transform DC Table (TRANSDCTAB)

This shall be identical to P pictures as defined in section 8.3.4.10.

8.4.4.10 Bitplane Coding

As in P frames, some information is coded as a bitplane that is sent at the frame level. For progressive B frames, two such bitplanes are sent – one denoting skipped macroblocks (refer to section 8.3.4.4) and the other denoting macroblocks coded in Direct mode. This information shall be sent at the macroblock level when the chosen coding mode is ‘Raw’. See section 8.7 for a description of bitplane coding.

8.4.4.11 Rounding Control (RND)

The rounding control parameter used by B frames shall be as described in 8.3.7.

8.4.4.12 Sync Markers

B frames shall not contain sync markers (see section 8.8).

8.4.4.13 Picture Resolution (RESPIC

If variable resolution coding is enabled for the sequence (signaled by the MULTIRES flag in the sequence header) then the resolution of the B frame shall be equal to the resolution of the two reference frames. See section 8.1.1.3 for a definition of how the current resolution is signaled. The two reference frames shall always have the same resolution.

This restriction means that B frames shall not occur temporally between an I and P or two I frames where the I frame has a different resolution to the preceding I or P frame.

Note: This restriction is always satisfied for B frames that occur temporally between two P frames since a resolution change can only occur at I frames (see 8.1.1.3).

8.4.4.14 Range Reduction Frame (RANGEREDFRM)

The RANGEREDFRM syntax element (**7.1.1.3**) shall only be present when RANGERED == 1 at the sequence level. RANGEREDFRM shall have the same value as the RANGEREDFRM syntax element of the temporally subsequent (in display order) anchor frame. This implies that no scaling shall be performed for performing motion compensation from the temporally subsequent (in display order) frame. However, scaling may be required for prediction from the temporally preceding anchor frame. This scaling shall be identical to the scaling performed for predicting P frames, and shall be as described in section 8.3.4.11. When RANGEREDFRM == 1 for the current B frame, the current decoded frame shall be scaled up similar to the scaling of the P frame, and shall be as described in section 8.3.4.11.

8.4.5 B Frame Macroblock Layer Decode

8.4.5.1 Macroblock Types

Macroblocks in B frames shall belong to one of four prediction modes: *backward*, *forward*, *direct* and *interpolated*. The direct mode is signaled by the DIRECTMB bitplane (defined in section 7.1.1.42), or DIRECTBBIT (defined in section 7.1.3.12). The other three modes are signaled in BMVTYPE (defined in 7.1.3.14). The forward mode is akin to conventional P picture prediction. Additionally, each macroblock may also be intra-coded (signaled by BMV1 element) or skipped. If the current macroblock is inter-coded, then the BMV1 syntax element shall also specify the motion vector differential as in P pictures. The decoding procedure for BMV1 shall be identical to the procedure for MVDATA, and shall be as described in section 8.3.5.2.1. All macroblocks in progressive B pictures shall only use 1-MV mode, i.e. 4-MV is not permitted. In the forward mode, the B macroblock shall be motion compensated from its temporally previous anchor frame only. Likewise, backward mode macroblocks shall be motion compensated from their temporally subsequent (in display order) anchor frame.

8.4.5.1.1 Skipped Macroblocks

Skipped macroblocks shall be signaled the same way as in P pictures (defined in section 8.3.5.1.3). Even if a macroblock is skipped, the prediction mode shall be decoded.

Note: The prediction mode of a macroblock is signaled by DIRECTMB bitplane or DIRECTBBIT, and if not direct, by BMVTYPE as described in 8.4.5.1.

8.4.5.2 Long and Short Types to signal Forward and Backward modes

The use of BMVTYPE to signal forward and backward modes is defined in section 7.1.3.14.

Note: When a B frame is closer to its temporally previous reference, it is expected that the forward coding mode will be used more often. Likewise, when a B frame is closer to the end of its inter-anchor interval, it is expected that the backward coding mode will be used more often. This statistical behavior is exploited by flagging the backward and forward mode using two codewords whose interpretation is switched across two sides of the midpoint of the inter-anchor interval.

8.4.5.3 Direct and Interpolated Modes

Macroblocks for which prediction mode is either direct or interpolated use both the anchors for prediction. They shall use two sets of motion vectors (MVs), one each to reference into the previous and next anchor frame. In both cases the pixels shall be interpolated from the two reference frames, which shall be followed by a pixel average operation with round-up to compute the pixels in the motion compensated macroblock:

Pixel value = (Interpolated value from anchor 1 + Interpolated value from anchor 2 + 1) >> 1

In interpolated mode the forward and backward motion vectors shall be explicitly coded within the bitstream. In interpolated mode, the backward motion vector shall be the first motion vector that is decoded (corresponding to BMV1), and the forward motion vector shall be the second motion vector that is decoded (corresponding to BMV2). In

direct mode the forward and backward motion vectors shall be derived by scaling the co-located motion vectors from the backward reference frame. These scaling operations shall be performed in ¼ pel units.

8.4.5.4 Decoding Direct Mode Motion Vectors

Direct mode MVs shall not be explicitly signaled or decoded. Instead, they shall be computed based on motion vectors buffered from the temporally subsequent (in display order) anchor frame and scaling logic as described below. The direct mode MVs shall be set to (0, 0) when the co-located macroblock of the subsequent anchor frame is intra-coded.

If the subsequent anchor P frame's co-located MV was 1-MV, then that MV shall be buffered for the next B frame to be coded.

If the P frame's co-located MV was 4-MV,

- median4( ) of the 4 MVs shall be used if none of the blocks is intra-coded;
- 'median3( ) of the 3 MVs shall be used if one of the blocks is intra-coded;
- averaging with integer division by 2 with truncation towards zero of 2 MVs, i.e. (MVa + MVb)/2, shall be used if two out of four blocks are intra-coded, where MVa and MVb are the MVs of the two inter-coded blocks.

The resulting MV shall be buffered. If the P frame's co-located MV was 4-MV, and if 3 or 4 of the blocks are intra, then the direct mode MVs shall be set to (0,0).

In the main profile only (and not in advanced profile), prior to buffering these MVs from the P frame to use in the subsequently decoded B frames, pullback operations shall be performed on these MVs (MV_X_To_Buffer, MV_Y_To_Buffer) according to the pseudo-code of Figure 68.

```
//MB_X_Offset is the horizontal position of the MB in units of macroblocks
//MB_Y_Offset is the vertical position of the MB in units of macroblocks
// iNumXMBtimes8 is the number of MBs along a row multiplied by 8
// iNumYMBtimes8 is the number of MBs along a column multiplied by 8
//iX0B and iY0B are intermediate variables used in the calculation.
iX0B = (MB_X_Offset << 3) + (MV_X_To_Buffer >> 2);
// full-pel resolution
iY0B = (MB_Y_Offset << 3) + (MV_Y_To_Buffer >> 2);
if (iX0B < -8)
    MV_X_To_Buffer -= (iX0B + 8) * 4;
else if (iX0B > iNumXMBtimes8)
    MV_X_To_Buffer -= (iX0B – iNumXMBtimes8) * 4;
if (iY0B < -8)
    MV_Y_To_Buffer -= (iY0B + 8) * 4;
else if (iY0B > iNumYMBtimes8)
    MV_Y_To_Buffer -= (iY0B - iNumYMBtimes8) * 4;
```

Figure 68: Pseudo-code for Pullback of Direct mode MVs in main profile

Given that the subsequent anchor (in display order) frame was a P frame (in case the next frame was I, all the direct mode motion vectors shall be (0,0)), and the co-located macroblock buffered the motion vector MV (MV_X, MV_Y), the direct mode shall compute two sets of motion vectors, one referencing into the forward or previous anchor frame, (MV_X_F , MV_Y_F), and the other referencing into the backward or subsequent (in display order) anchor frame, (MV_X_B, MV_Y_B) according to the pseudo-code of Figure 69.

```
Scale_Direct_MV (IN MV_X, IN MV_Y, OUT MV_XF , OUT MV_YF, OUT MV_XB, OUT MV_YB)
//ScaleFactor computed according to the pseudo-code of Figure 70.
// Total_MBs_along_X is the number of MBs along a row.
// Total_MBs_along_Y is the number of MBs along a column.
```

```
// MB_X_Offset_in_qpels the horizontal position of the MB in units of pixels multiplied by 4.
// MB_Y_Offset_in_qpels the vertical position of the MB in units of pixels multiplied by 4.
// All computations are performed in ¼ pel units.
if (MVMode == 1-MV Half-pel || MVMode == 1-MV Half-pel Bilinear) {
        MV_X_F = 2 * ((MV_X * ScaleFactor + 255) >> 9);
        MV_Y_F = 2 * ((MV_Y * ScaleFactor + 255) >> 9);
        MV_X_B = 2 * ((MV_X * (ScaleFactor - 256) + 255) >> 9);
        MV_Y_B = 2 * ((MV_Y * (ScaleFactor - 256) + 255) >> 9);
}
else {          /* Quarter pel units */
        MV_X_F = (MV_X * ScaleFactor + 128) >> 8;
        MV_Y_F = (MV_Y * ScaleFactor + 128) >> 8;
        MV_X_B = (MV_X * (ScaleFactor - 256) + 128) >> 8;
        MV_Y_B = (MV_Y * (ScaleFactor - 256) + 128) >> 8;
}

if (Main or Advanced profiles, Progressive Only)    /* Do pullback */
// Note that this condition is satisfied by definition for progressive B pictures
{
    // limit motion vector around periphery of the frame
    Int iMinCoordinate = -60;
            // -15 pixels in quarter pel units
    Int iMaxX = Total_MBs_along_X * 64 - 4;
    Int iMaxY = Total_MBs_along_Y * 64 - 4;
    Int iX1 = MB_X_Offset_in_qpels + MV_X_F;
    Int iY1 = MB_Y_Offset_in_qpels + MV_Y_F;

    if (iX1 < iMinCoordinate)
        MV_X_F = iMinCoordinate - MB_X_Offset_in_qpels;
    else if (iX1 > iMaxX)
        MV_X_F = iMaxX - MB_X_Offset_in_qpels;
    if (iY1 < iMinCoordinate)
        MV_Y_F = iMinCoordinate - MB_Y_Offset_in_qpels;
    else if (iY1 > iMaxY)
        MV_Y_F = iMaxY - MB_Y_Offset_in_qpels;
    iX1 = MB_X_Offset_in_qpels + MV_X_B;
    iY1 = MB_Y_Offset_in_qpels + MV_Y_B;
    if (iX1 < iMinCoordinate)
        MV_X_B = iMinCoordinate - MB_X_Offset_in_qpels;
    else if (iX1 > iMaxX)
        MV_X_B = iMaxX - MB_X_Offset_in_qpels;
    if (iY1 < iMinCoordinate)
        MV_Y_B = iMinCoordinate - MB_Y_Offset_in_qpels;
    else if (iY1 > iMaxY)
        MV_Y_B = iMaxY - MB_Y_Offset_in_qpels;
}
```

```
End Scale_Direct_MV
```

Figure 69: Pseudo-code for Computation of Direct mode MVs

"ScaleFactor" shall be computed at the start of decoding each B frame, according to the pseudo-code of Figure 70.

```
Int NumShortVLC[] = {1, 1, 2, 1, 3, 1, 2};
Int DenShortVLC[] = {2, 3, 3, 4, 4, 5, 5};
Int NumLongVLC[] =   {3, 4, 1, 5, 1, 2, 3, 4, 5, 6, 1, 3, 5, 7};
Int DenLongVLC[] =   {5, 5, 6, 6, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8};
Int Inverse[] = { 256, 128, 85, 64, 51, 43, 37, 32 };
//Numerator, Denominator, and FrameReciprocal are variables used in ScaleFactor computation
Frame_Initialization(code word)
// codewords 000b through 110b are "short" code words, and the rest are "long" code words.
if (long code word)
{
        Numerator       = NumLongVLC[code word - 112];
        Denominator = DenLongVLC[code word - 112];
}
else /* short code word */
{
        Numerator      = NumShortVLC[code word];
        Denominator = DenShortVLC[code word];
}
FrameReciprocal = Inverse[Denominator - 1];
ScaleFactor = Numerator * FrameReciprocal;
End Frame_Initialization
```

Figure 70: Pseudo-code for Computation of ScaleFactor in Direct mode

The code word shall be obtained by decoding the frame level syntax element BFRACTION (**7.1.1.4**) according to Table 40. In this table, the code-words 000b through 110b shall be known as the "short" code words and the remainder shall be known as the "long" code words.

Figure 71 illustrates how direct mode scales the motion vectors from the next P frame, while the scaling operation is defined by Figure 69.

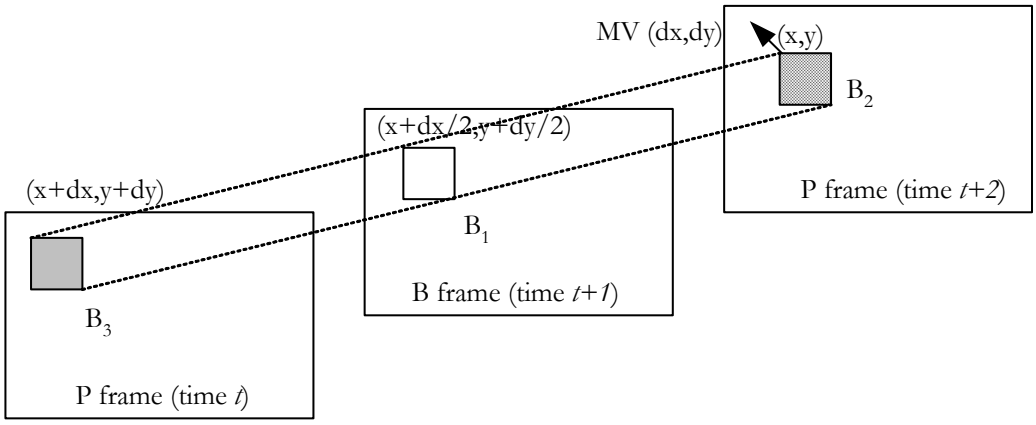


Figure 71: Illustration of Direct Mode Prediction

8.4.5.5 Decoding Motion Vector Differentials

This shall be identical to P pictures defined in section 8.3.5.2.1.

Note: 4-MV is not present in B pictures, and the B frame motion vectors are referred to as BMV1 and BMV2 instead of MVDATA and BLKMVDATA.

8.4.5.6 Motion Vector Predictors

This shall be the same as the motion vector prediction (as defined in 8.3.5.3.1) for 1-MV P pictures, with the addition of separate prediction contexts for forward and backward mode MVs. The forward prediction contexts shall be used only to predict forward MVs and the backward prediction contexts shall be used only for the prediction of backward MVs.

8.4.5.6.1 Populating the forward and backward prediction contexts

The forward and backward motion vectors shall be buffered separately, and they shall be used separately to predict forward and backward motion vectors respectively. For 'Interpolated' macroblocks, the forward prediction buffer shall be used to predict the forward MV, i.e. BMV2, and the backward buffer shall be used to predict the backward MV, i.e. BMV1. When the MB is of type 'Direct' or 'Interpolated', the forward MV component shall be buffered in the forward buffer and the backward MV component shall be buffered in the backward buffer.

When the MB is of type 'Forward', the forward MV shall be buffered after it is decoded in the forward prediction buffer. Then the backward MV component of the direct mode shall be computed to fill in the corresponding position in the backward buffer (MV_X_B, MV_Y_B). When the MB is of type 'Backward', the backward MV shall be buffered after it is decoded in the backward prediction buffer. Then the forward MV component of the direct mode shall be computed to fill in the corresponding position in the forward buffer (MV_X_F, MV_Y_F). Thus, in progressive B frames the direct mode motion vector shall be calculated even for an MB that is only forward (or backward predicted).

If a macroblock in the backward reference frame was coded as an intra macroblock then the co-located macroblock positions in the forward and backward contexts shall also be considered to be intra-coded.

Note: One way to implement this is by using an "intra motion vector" to fill in both forward and backward motion prediction planes if the MB is intra-coded. Any consistent representation of "intra motion vector" can be chosen by the decoder implementation. E.g. If the MVs are being stored in a 2-byte short array, then "intra motion vector" could be represented as a unique large constant that is filled into the MV array to indicate that the MB was coded as intra.

8.4.5.7 Motion Vector Prediction

MV prediction logic for progressive B pictures shall be the same as in 1-MV P frames (as defined in section 8.3.5.3.1). Hybrid prediction and 4-MV shall not be used in B pictures. The MV predictor pullback operation for B pictures is defined in section 8.4.5.8. As described above, forward MVs shall be used to predict an incoming forward MV, and backward MVs shall be used to predict an incoming backward MV.

8.4.5.8 Motion Vector Predictor Pull-back

A pull-back operation shall be performed on the predicted motion vectors as follows:

- Let X = 16*((CodedWidth+15) /16) * 4 – 4 and
- Y = 16*((CodedHeight+15) /16) * 4 – 4 and
- let (MBx, MBy) be the macroblock in the current frame with predicted motion vector (predictor_pre_x, predictor_pre_y).
- Let Xmain = 8*((CodedWidth+15) /16) * 4 – 4 and
- Ymain = 8*((CodedHeight+15) /16) * 4 – 4 and
- Let the values qx and qy represent the quarter pixel coordinates of the top left corner of the macroblock as qx = MBx * 16 * 4, qy = MBy * 16 * 4.

In the main profile, the pullback operation on the motion vector predictor of the macroblock shall be implemented as follows:

- If qx/2 +predictor_pre_x <-28, then predictor_pre_x shall be set to the value (-28 –qx/2).
- If qx/2 +predictor_pre_x > Xmain, then predictor_pre_x shall be set to the value (Xmain – qx/2).
- If qy/2 +predictor_pre_y <-28, then predictor_pre_y shall be set to the value (-28 –qy/2).
- If qy/2+predictor_pre_y >Ymain, then predictor_pre_y shall be set to the value of (Ymain –qy/2).

In the advanced profile, the pullback operation on the motion vector predictor of the macroblock shall be implemented as follows:

- If qx +predictor_pre_x <-60, then predictor_pre_x shall be set to the value (-60 –qx).
- If qx +predictor_pre_x > X , then predictor_pre_x shall be set to the value (X –qx).
- If qy +predictor_pre_y <-60, then predictor_pre_y shall be set to the value (-60 –qy).
- If qy +predictor_pre_y >Y, then predictor_pre_y shall be set to the value of (Y – qy).

8.4.5.9 Motion Vector Decoding in B Frames

Whether or not a macroblock is coded as 'Direct' is known at the start of decoding macroblock level information. Non-direct macroblocks shall have one or two associated motion vectors, and direct macroblocks shall have none. Skipped macroblocks that are direct coded shall have zero residual Transform coefficients, and skipped non-direct coded macroblocks shall also have zero residual motion. If a MB is "skipped", the MB mode shall be signaled, to identify whether the "skipped" MB shall use 'Direct', 'Forward', 'Backward' or 'Interpolated' prediction. For a skipped MB, prediction shall be performed as usual (treating each mode with the appropriate decoding rules) and the predicted MVs shall be the reconstructed motion vectors.

Since motion vector information is jointly coded with the **intra_flag**, intra macroblocks shall be identified by decoding the first motion vector. Thus no mode information is sent after an intra motion vector is received.

Note: Thus, some efficiency is gained by coding the mode of the non-direct macroblock *after* sending the first motion vector.

When the first motion vector is non-intra, the macroblock type shall be present. The coding of BMVTYPE is defined in section 7.1.3.14, and the motivation for this coding scheme is defined in section 8.4.5.2.

The second motion vector shall be sent only if the macroblock is interpolated and if the **'more_present flag'** (see section 8.3.5.2.1) component of the first motion vector (BMV1) is nonzero. If the **'more_present flag'** component is zero for an interpolated macroblock, it implies that the second residual motion vector of the interpolated block is zero, and so are the residual Transform terms.

Note: In the interpolated mode BMV1 is the backward and BMV2 is the forward motion vector.

8.4.5.10 Motion Vector Prediction in Skipped Macroblocks

In skipped macroblocks, only the prediction mode, i.e. BMVTYPE, shall be decoded. Then the appropriate (i.e. forward or backward prediction) buffers shall be used to predict the motion vector of that type. The predicted motion vector shall be the actual motion vector in this case, as there are no residual differential MVs. This motion vector shall then be added to the appropriate (forward or backward) prediction context.

8.4.5.11 Reconstructing Motion Vectors

The following sections describe how to reconstruct the luma and color-difference motion vectors for B pictures.

8.4.5.11.1 Luma Motion Vector Reconstruction

This shall be identical to the corresponding operation in 1-MV macroblocks of P pictures as defined in section 8.3.5.4.1.

8.4.5.11.2 Color-difference Motion Vector Reconstruction

In the first step, the nominal color-difference motion vector shall be obtained by combining and scaling the luma motion vectors appropriately. The scaling is performed in such a way that half-pixel offsets are preferred over quarter pixel locations.

In the second stage, the 1-bit FASTUVMC syntax element (**6.2.6**, Annex J.1.11) is used to determine if further rounding of color-difference motion vectors is necessary. If FASTUVMC == 0, no rounding shall be performed in the second stage. If FASTUVMC == 1, the color-difference motion vectors that are at quarter pel offsets shall be rounded to the nearest half or full pel positions.

These operations shall be specified by the pseudo-code of Figure 72.

```
 // weak coercion to half pel positions
 // s_RndTbl[0] = 0, s_RndTbl[1] = 0, s_RndTbl[2] = 0, s_RndTbl[3] = 1
//LumaMVx and LumaMVy are x and y components of luma motion vectors.
//ColordifferenceMVx and ColordifferenceMVy are x and y components of color-difference motion vectors.
    ColordifferenceMVx = (LumaMVx + s_RndTbl[(LumaMVx) & 3]) >> 1;
    ColordifferenceMVy = (LumaMVy + s_RndTbl[(LumaMVy) & 3]) >> 1;
    // strong coercion to half pel positions
    if (FASTUVMC == 1) {
        if ((ColordifferenceMVx) & 1) {
            if (ColordifferenceMVx > 0) ColordifferenceMVx = ColordifferenceMVx - 1;
            else ColordifferenceMVx = ColordifferenceMVx + 1;
        }
        if ((ColordifferenceMVy) & 1) {
            if (ColordifferenceMVy > 0) ColordifferenceMVy = ColordifferenceMVy - 1;
            else ColordifferenceMVy = ColordifferenceMVy + 1;
        }
    }
```

Figure 72: Color-difference MV Reconstruction in B Pictures

8.4.5.12 Coded Block Pattern

This shall be identical to P pictures as defined in section 8.3.5.5, except restricted to the 1-MV case.

8.4.5.13 MB-level Transform Type (TTMB)

This shall be identical to P pictures as defined in section 8.3.5.6.

8.4.5.14 Subpixel Interpolation

Subpixel interpolation of B frames shall be performed in the same manner as interpolation of P frames as defined in section 8.3.6.5 except as follows.

- The valid modes shall be quarter pel bicubic and half pel bilinear.

8.4.5.15 Reconstructing and Adding Error

The decoding, dequantization, inverse transform, and addition of error blocks to the predicted blocks shall be performed in a manner identical to that used in P frames as defined in sections 8.3.6.1, 8.3.6.2, 8.3.6.3, 8.3.6.4, 8.3.6.5.3 and 8.3.6 except as follows:

- Intra macroblocks shall be decoded as they are in P frames.
- However, the overlapped smoothing operation applied to edges between intra blocks in P frames shall not be performed in B frames.

8.4.6 B Block Layer Decode

Block decoding syntax and operations shall be the same as for P pictures as defined in section 8.3.6, except as follows.

- The only points of slight difference occur in the pullback of motion vectors during motion compensation, and these are defined in the subsection below.

8.4.6.1 Motion Compensation

The motion compensation of blocks in B pictures shall follow the same rules as those described for P blocks in section 8.3.6.5.

The following operation shall be applied to adjust the motion vectors for main profile:

Let iXCoord, iYCoord be the spatial location of the top left corner of the current macroblock (e.g. if the current macroblock is located on the 3rd column and 2nd row, then iXCoord = 2 * 16 and iYCoord = 1 * 16).

Let iMvX, iMvY be the reconstructed luma motion vector in quarter pixel unit in the forward or backward direction.

Let iNumMBX, iNumMBY be the number of macroblocks in a row and a column, respectively.

Then the adjusted luma motion vector iMvXComp, iMvYComp for motion compensation shall be computed using the pseudo-code of Figure 73.

```
// iPosX and iPosY are intermediate variables.
        iPosX = iXCoord + (iMvX >> 2);
iPosY = iYCoord + (iMvY >> 2);
iMvXComp = iMvX;
iMvYComp = iMvY;
if   (iPosX < -16) {
      iMvXComp = ((-16 – iXCoord)<<2) + (iMvX & 3);
} else if (iPosX > iNumMBX * 16) {
      iMvXComp = ((iNumMBX * 16 – iXCoord)<<2) + (iMvX & 3);
}
if   (iPosY < -16) {
      iMvYComp = ((-16 – iYCoord)<<2) + (iMvY & 3);
} else if (iPosY > iNumMBY * 16) {
      iMvYComp = ((iNumMBY * 16 – iYCoord)<<2) + (iMvY & 3);
   }
```

Figure 73: Pullback of Reconstructed MVs in B Pictures

If the BMVTYPE is ‘Direct’ or ‘Interpolated’, then the above operations shall be performed for both sets of (i.e. forward and backward pointing) motion vectors.

These adjusted luma motion vectors, iMvXComp, iMvYComp, shall then be used to generate the color-difference motion vectors in B pictures as described in section 8.4.5.11.2 (referred there as LumaMVx, LumaMVy). In case of BMVTYPE being ‘Direct’ or ‘Interpolated’, two sets of color-difference motion vectors shall be generated. But no further pullback operations shall be applied to the color-difference motion vectors.

8.5 Overlapped Transform

If the syntax element OVERLAP (**6.2.10**, Annex J.1.15) is set to 1, then a filtering operation shall be conditionally performed across the edges of two neighboring Intra blocks, for both the luma and color-difference channels. This filtering operation (referred to as *overlap smoothing*) shall be performed subsequent to decoding the frame, and prior to in-loop deblocking.

Note: Overlap smoothing can be done immediately after the relevant macroblock slices are decoded as this is functionally equivalent to smoothing after decoding the entire frame.

Macroblock aligned frame dimensions shall be used for performing the overlap smoothing (i.e. if the frame dimensions are 158x118, overlap smoothing shall be performed over the macroblock aligned dimensions of 160x128).

Note: Overlapped transforms are modified block based transforms that exchange information across the block boundary. With a well designed overlapped transform, blocking artifacts can be minimized. For intra blocks, an overlapped transform is simulated by coupling an 8x8 block transform with overlap smoothing. Edges of an 8x8 block that separate two intra blocks are smoothed.

Figure 74 shows a portion of a P frame with I blocks. This could be either the Y or C_b or C_r channel. I blocks are gray (or crosshatched) and P blocks are white. The edge interface over which overlap smoothing is applied is marked

with a crosshatch pattern. Overlap smoothing is applied to two pixels on either side of the separating boundary. The right bottom area of frame is shown here as an example. Pixels occupy individual cells and blocks are separated by heavy lines. The dark circle marks the 2x2 pixel corner subblock that is filtered in both directions.

The lower inset in Figure 74 shows four labeled pixels, a0 and a1 are to the left and b1, b0 to the right of the vertical block edge. The upper inset shows pixels marked p0, p1, q1 and q0 straddling a horizontal edge. The next section describes the filter applied to these four pixel locations.

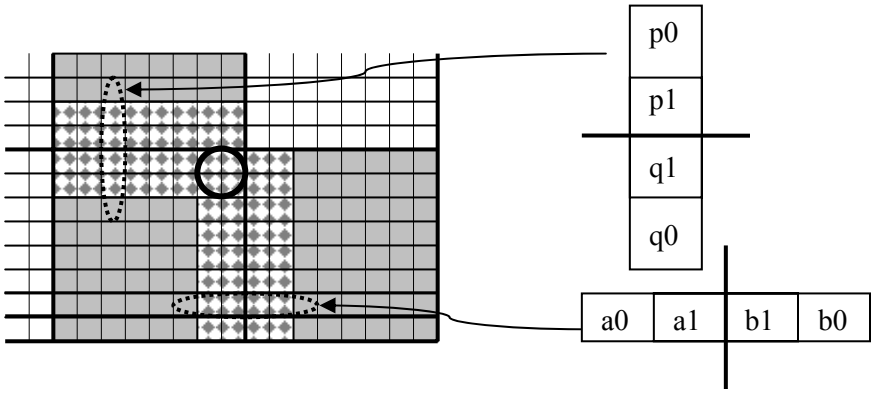


Figure 74: Example showing overlap smoothing

Overlap smoothing shall be carried out on the unclamped 10 bit reconstruction. In other words, the input to the overlap smoothing process shall be the inverse transformed spatial block of pixels whose dynamic range is 10 bits.

Note: This is necessary because the forward process associated with overlap smoothing can result in range expansion beyond the permissible 8 bit range for pixel values.

Vertical edges (pixels a0, a1, b1, b0 in the above example) shall be filtered first, followed by the horizontal edges (pixels p0, p1, q1, q0). The core filter applied to the four pixels straddling either the vertical or horizontal edge is defined below:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \left(\begin{pmatrix} 7 & 0 & 0 & 1 \\ -1 & 7 & 1 & 1 \\ 1 & 1 & 7 & -1 \\ 1 & 0 & 0 & 7 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} r_0 \\ r_1 \\ r_0 \\ r_1 \end{pmatrix} \right) >> 3$$

The original pixels being filtered are (x0, x1, x2, x3). r0 and r1 are rounding parameters, which shall take on alternating values of 3 and 4. For both horizontal and vertical edge filters, the rounding values shall be r0 = 4, r1 = 3 for odd-numbered columns and rows respectively, assuming the numbering within a block to start at 1. For even-numbered columns and rows, the rounding values shall be r0 = 3 and r1 = 4 respectively.

For vertical edge filtering, the pixels (a0, a1, b1, b0) correspond to (x0, x1, x2, x3), which in turn get filtered to (y0, y1, y2, y3). Likewise, for horizontal edge filtering, the correspondence is with (p0, p1, q1, q0) respectively.

Pixels in the 2x2 corner, shown by the dark circle in Figure 74, are filtered in both directions. Vertical edge filtering shall be performed first, followed by horizontal edge filtering. For these pixels, the intermediate result after vertical edge filtering shall be retained to the full precision of 11 bits (10 bit input plus 1 bit worst case range expansion due to overlap smoothing).

Subsequent to filtering, the constant value of 128 shall be added to each pixel of the block, which shall then be clamped to the range [0 255] to produce the reconstructed output.

8.5.1 Overlap Smoothing in Main and Simple Profiles

Overlap smoothing in main and simple profiles shall be applied subject to the following rules:

1. No overlap smoothing shall be performed for any frame if the OVERLAP == 0; The remainder of these rules shall apply only when OVERLAP == 1.
    a. Overlap smoothing shall be applied only if the frame level quantization step size PQUANT is 9 or above. If this condition is satisfied,
        i. All 8x8 block boundaries between adjacent 8x8 blocks shall be smoothed for I frames and BI frames.
        ii. Only block boundaries separating two intra blocks shall be smoothed for P frames.
        iii. No overlap smoothing shall be performed for predicted B frames, i.e. B frames that are not coded as BI.
        iv. There shall be no dependence on DQUANT or differential quantization across macroblocks

8.5.2 Overlap Smoothing in Advanced Profile

Overlap smoothing shall be applied subject to the following rules:

1. No overlap smoothing shall be performed for any frame if OVERLAP == 0; the remainder of these rules shall apply only when OVERLAP == 1.
2. No overlap smoothing shall be performed for predicted B frames, (i.e. non BI frames).
3. Only block boundaries separating two intra blocks shall be smoothed for P frames such that:
    a. Picture level quantization step size PQUANT is 9 or higher, regardless of HALFQP.
4. For I frames, and BI frames, 8x8 block boundaries (which are defined as boundaries between adjacent 8x8 blocks) shall be smoothed as per the following rules:
    a. When picture level quantization step size PQUANT is 9 or higher (regardless of HALFQP), all 8x8 block boundaries shall be smoothed.
    b. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), no 8x8 block boundaries shall be smoothed if the conditional overlap flag CONDOVER == 0b.
    c. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), all 8x8 block boundaries shall be smoothed if the conditional overlap flag CONDOVER == 10b.
    d. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), some 8x8 block boundaries shall be smoothed if the conditional overlap flag CONDOVER == 11b, and the following additional rules apply:
        i. Internal 8x8 block boundaries (i.e. boundaries between blocks in the same macroblock) within the luma plane of a macroblock shall be smoothed when the decoded binary symbol from OVERFLAGS bitplane, or OVERFLAGMB when the raw mode is used to code OVERFLAGS bitplane, for the macroblock is 1.
        ii. 8x8 block boundaries between adjacent macroblocks (both luma and color-difference) shall be smoothed only when the decoded binary symbols from OVERFLAGS bitplane, or OVERFLAGMB when the raw mode is used to code OVERFLAGS bitplane, for both adjacent macroblocks are 1.
5. There shall be no dependence on DQUANT or differential quantization across macroblocks.
6. There shall be no overlap across a block boundary, if the adjacent macroblocks (both color-difference and luma) belong to different slices.

Note: Conditional overlap is applicable only for I frames and BI frames. Conditional overlap allows the selective smoothing of 8x8 block boundaries within macroblocks and between adjacent macroblocks. The signaling is based on one binary symbol per macroblock – which is interpreted in a strict sense to mean that an edge between macroblocks is filtered only if OVERFLAGS == 1 for both macroblocks. There is no block or block edge level control.

8.6 In-loop Deblock Filtering

If the syntax element LOOPFILTER == 1 (**6.2.5**, Annex J.1.9), then a filtering operation shall be performed on each reconstructed frame. This filtering operation shall be performed prior to using the reconstructed frame as a reference for motion predictive coding. When there are multiple slices in a picture, the loop filter for each slice shall be performed independently as defined in section 7.1.2.

Since the intent of loop filtering is to smooth out the discontinuities at block boundaries the filtering process operates on the pixels that border neighboring blocks. For P pictures, the block boundaries may occur at every 4^{th}, 8^{th}, 12^{th}, etc pixel row or column depending on whether an 8x8, 8x4 or 4x8 Inverse Transform is used. For I pictures filtering occurs at every 8^{th}, 16^{th}, 24^{th}, etc pixel row and column.

8.6.1 I Picture In-loop Deblocking

For I pictures, deblock filtering shall be performed at all 8x8 block boundaries. Figure 75 and Figure 76 show the pixels that are filtered along the horizontal and vertical border regions. The figures show the upper left corner of a component (Y, C_b or C_r) plane. The crosses represent pixels and the circled crosses represent the pixels that are filtered.

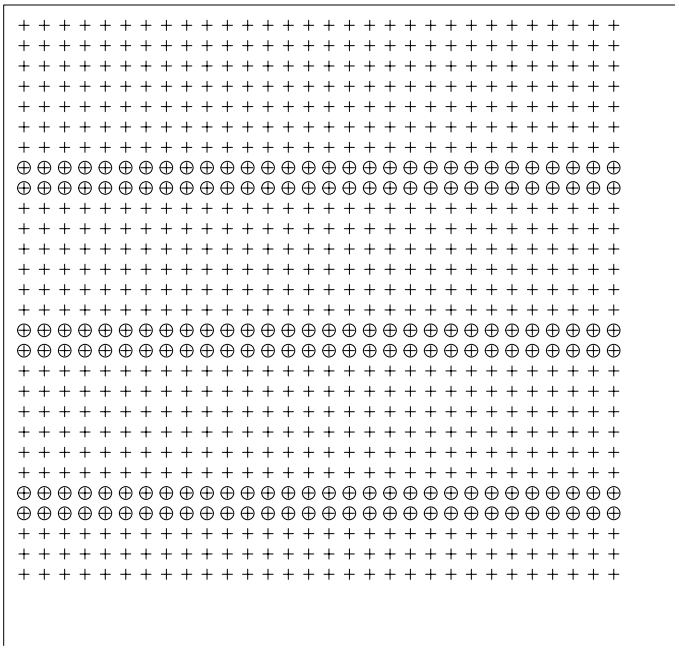


Figure 75: Filtered horizontal block boundary pixels in I picture

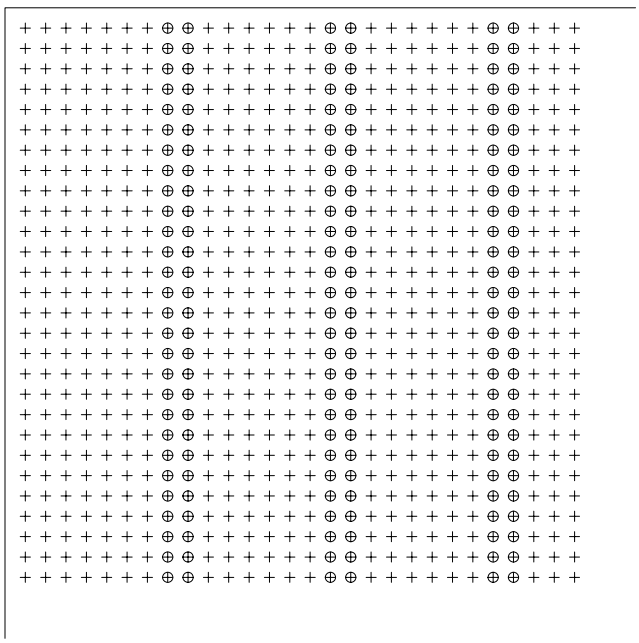


Figure 76: Filtered vertical block boundary pixels in I picture

As the figures show, the top horizontal line and first vertical line shall not be filtered. Although not depicted, the bottom horizontal line and last vertical line shall also not be filtered. In more formal terms, the following lines shall be filtered:

Horizontal lines (7,8), (15,16) … ((N – 1)*8 – 1, (N –1)*8) shall be filtered.

Vertical lines (7, 8), (15, 16) … ((M-1)*8 - 1, (M – 1)*8) shall be filtered.

Where:

N = the number of horizontal 8x8 blocks in the plane (N*8 = horizontal frame size).

M = the number of vertical 8x8 blocks in the frame (M*8 = vertical frame size).

All the horizontal boundary lines in the frame shall be filtered first followed by the vertical boundary lines.

8.6.2 P Picture In-loop Deblocking

For P pictures, blocks may be Intra or Inter-coded. Intra-coded blocks shall use an 8x8 Inverse Transform to reconstruct the samples, whereas inter coded blocks shall either use an 8x8, 8x4, 4x8 or 4x4 Inverse Transform. The boundary between transform blocks or subblocks shall be filtered, unless the following exception holds. When the transform blocks (or subblocks) on either side of the boundary are both inter coded, and when the motion vectors of these blocks (or subblocks) are identical, and when both blocks (or subblocks) have all transform coefficients equal to zero, filtering shall not be performed.

Figure 77 shows examples of when filtering between neighboring blocks (or subblocks) does and does not occur. In this example it is assumed that the motion vectors for both blocks are the same (if the motion vectors are different, then the boundary is always filtered). The shaded blocks or subblocks represent the cases where at least one nonzero coefficient is present.

Clear blocks or subblocks represent cases where no Transform coefficients are present. Thick lines represent the boundaries that are filtered. Thin lines represent the boundaries that are not filtered. These examples illustrate only horizontal block neighbors. The same applies for vertical block neighbors.

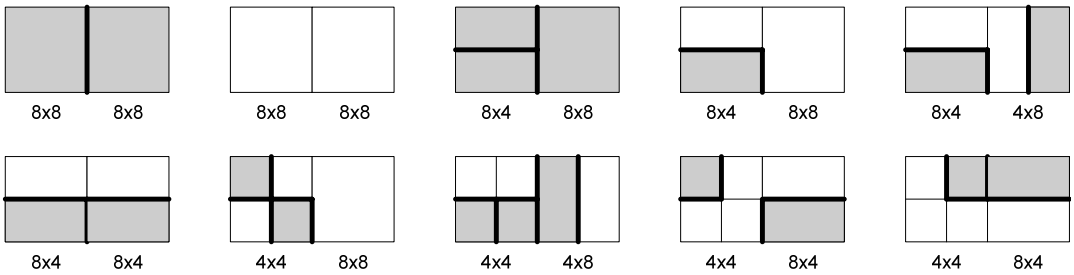


Figure 77: Example filtered block boundaries in P frames

Figure 78 and Figure 79 shows an example of the pixels that could be filtered in a P frame. The crosses represent pixel locations and the circled crosses represent the boundary pixels that are filtered if the conditions specified above are met.

Figure 78 shows pixels filtered along horizontal boundaries. As the figure shows, the pixels on either side of the block or subblock boundary are candidates to be filtered. For the horizontal boundaries this could be every 4th and 5th, 8th and 9th, 12th and 13th etc pixel row in the frame as these are the 8x8 and 8x4 horizontal boundaries.

Figure 79 shows pixels filtered along vertical boundaries. For the vertical boundaries, every 4th and 5th, 8th and 9th, 12th and 13th etc pixel column in the frame may be filtered as these are the 8x8 and 4x8 vertical boundaries.

The first and last row and the first and last column in the frame shall not be filtered.

The order in which pixels are filtered is important. First, all blocks or subblocks that have a horizontal boundary along the 8th, 16th, 24th, etc horizontal lines shall be filtered. Next, all subblocks that have a horizontal boundary along the 4th, 12th, 20th, etc horizontal lines shall be filtered. Next, all blocks or subblocks that have a vertical boundary along the 8th, 16th, 24th, etc columns shall be filtered. Lastly, all subblocks that have a vertical boundary along the 4th, 12th, 20th, etc columns shall be filtered. In all cases, the rules specified above shall be used to determine whether the boundary pixels are filtered for each block or subblock.

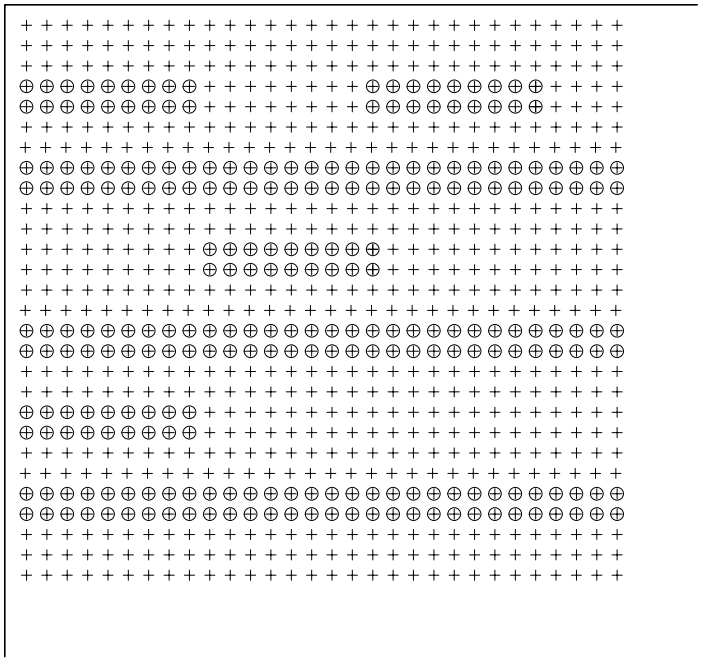


Figure 78: Horizontal block boundary pixels in P picture

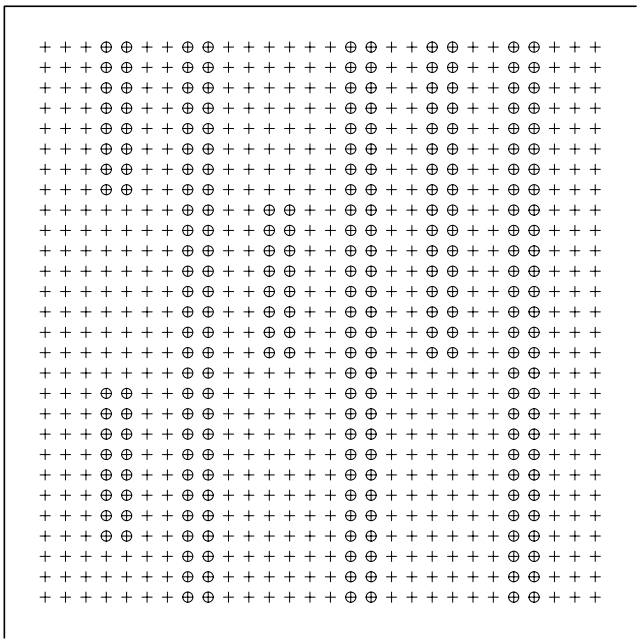


Figure 79: Vertical block boundary pixels in P picture

8.6.3 B Picture In-loop Deblocking

For B pictures, in-loop deblocking is exactly the same as I picture in-loop deblocking (see section 8.6.1), i.e. the 8x8 block boundaries shall be filtered, and MVs or 4x8/8x4 blocks are not considered.

8.6.4 Filter Operation

This section describes the filtering operation that is performed on the boundary pixels in I, BI, P, and B frames.

For P frames the decision criteria listed in section 8.6.2 determines which vertical and horizontal boundaries are filtered. For I, B and BI frames, all the 8x8 vertical and horizontal block boundaries are filtered. Since the minimum number of consecutive pixels that are filtered in a row or column is four and the total number of pixels in a row or column is always a multiple of four, the filtering operation shall be performed on segments of four pixels.

For example, if the eight pixel pairs that make up the vertical boundary between two blocks are filtered, then the eight pixels are divided into two 4-pixel segments as shown in Figure 80 where the dotted line defines the boundary between the two 4-pixel segments. In each 4-pixel segment, the third pixel pair shall be filtered first as indicated by the X's. The result of this filter operation shall determine whether the other three pixels in the segment are also filtered, as defined below.

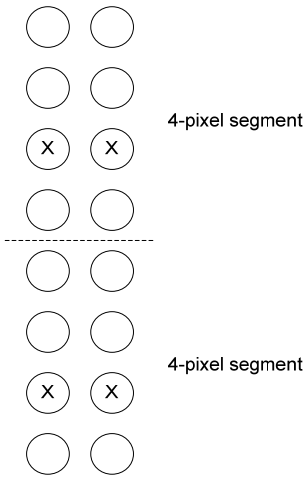


Figure 80: Four-pixel segments used in loop filtering

Figure 81 shows the pixels that shall be used in the filtering operation performed on the 3rd pixel pair. Pixels P4 and P5 are the pixel pairs that can be changed as a result of the filter operation.

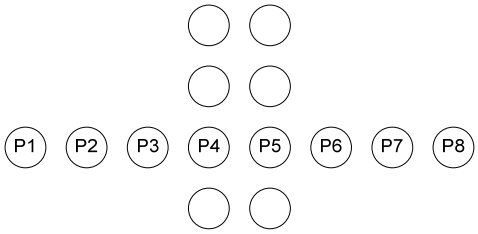


Figure 81: Pixels used in filtering operation

The pseudo-code of Figure 82 shall specify the filtering operation performed on the 3rd pixel pair in each segment. The Boolean value 'filter_other_3_pixels' defines whether the remaining 3 pixel pairs in the segment are also filtered. If 'filter_other_3_pixels' == TRUE, then the other three pixel pairs shall be filtered. If 'filter_other_3_pixels == FALSE, then they shall not be filtered, and the filtering operation proceeds to the next 4-pixel segment. The pseudo-code of Figure 83 shall specify the filtering operation that is performed on the 1st, 2nd and 4th pixel pair if 'filter_other_3_pixels' == TRUE.

```
filter_other_3_pixels = TRUE
int a0, a1, a2, a3 clip, d // local vars

a0 = (2*(P3 - P6) - 5*(P4 – P5) + 4) >> 3
if (|a0| < PQUANT) {
        a1 = (2*(P1 - P4) - 5*(P2 - P3) + 4) >> 3
        a2 = (2*(P5 - P8) - 5*(P6 - P7) + 4) >> 3
        a3 = min(|a1|, |a2|)
        if (a3 < |a0|)
        {
                d = 5*((sign(a0) * a3) - a0)/8
                clip = (P4 – P5)/2
                if (clip == 0)
                        filter_other_3_pixels = FALSE
                else
                {
```

```
                        if (clip > 0)
                        {
                                if (d < 0)
                                        d = 0
                                if (d > clip)
                                        d = clip
                        }
                        else
                        {
                                if (d > 0)
                                        d = 0
                                if (d < clip)
                                        d = clip
                        }
                                P4 = P4 - d
                                P5 = P5 + d
                }
        }
    else
                filter_other_3_pixels = FALSE
}
else
                filter_other_3_pixels = FALSE
```

Figure 82: Pseudo-code illustrating filtering of 3rd pixel pair in segment

```
int a0, a1, a2, a3 clip, d // local vars

a0 = (2*(P3 - P6) - 5*(P4 – P5) + 4) >> 3
if (|a0| < PQUANT)
{
        a1 = (2*(P1 - P4) - 5*(P2 - P3) + 4) >> 3
        a2 = (2*(P5 - P8) - 5*(P6 - P7) + 4) >> 3
        a3 = min(|a1|, |a2|)
         if (a3 < |a0|)
         {
                d = 5*((sign(a0) * a3) - a0)/8
                clip = (P4 – P5)/2

                if (clip > 0)
                {
                        if (d < 0)
                                d = 0
                        if (d > clip)
                                d = clip
                        P4 = P4 - d
                        P5 = P5 + d
```

```
                }
                else if (clip < 0)
                {
                        if (d > 0)
                                d = 0
                        if (d < clip)
                                d = clip
                         P4 = P4 - d
                         P5 = P5 + d
                }
        }
}
```

Figure 83: Pseudo-code illustrating filtering of 1st, 2nd and 4th pixel pair in segment

This section used the vertical boundary for example purposes. The same operation shall be used for filtering the horizontal boundary pixels, where pixels P1, P2, P3 and P4 reside to the left of the horizontal boundary, and pixels P5, P6, P7 and P8 reside to the right of the horizontal boundary.

8.6.4.1 Main Profile Deblocking for P picture

If the syntax element LOOPFILTER = 1 (**6.2.5**, Annex J.1.9), the decoder shall perform the following exception operations for the main profile in-loop deblocking filter. These operations differ slightly from the generic operations as defined next:

1. If the first macroblock in the frame is coded as intra or if the upper left luma block (block 0) of the first macroblock in the frame is coded as intra then the top and left block boundaries of all the inter-coded macroblocks that use the 1-MV motion compensation mode shall not use the motion vector, coded-block status or subblock pattern to decide which block boundary segments are candidates for deblock filtering. In other words, the entire 8-sample top boundary and the entire the 8-sample left boundary are filtered. The internal subblock boundaries are filtered as described in section 8.6.2.

2. The criteria used to decide whether to filter the left boundary of block 3 (the lower-right luma block) shall be derived from the motion vector status of blocks 2 and 3 as intended but instead of using the coded-block status and subblock patterns (if present) for blocks 2 and 3 as intended, the coded-block status and subblock patterns of blocks 1 and 3 shall be used.

A high-level overview to illustrate exception 2 is shown in Figure 84 and Figure 85.

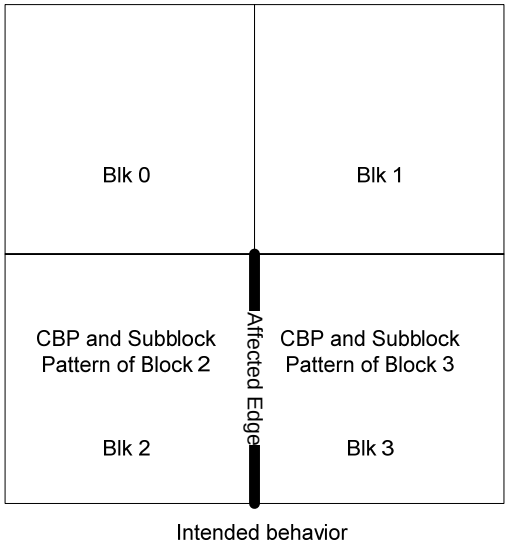


Figure 84: Overview Figure A to illustrate Exception 2

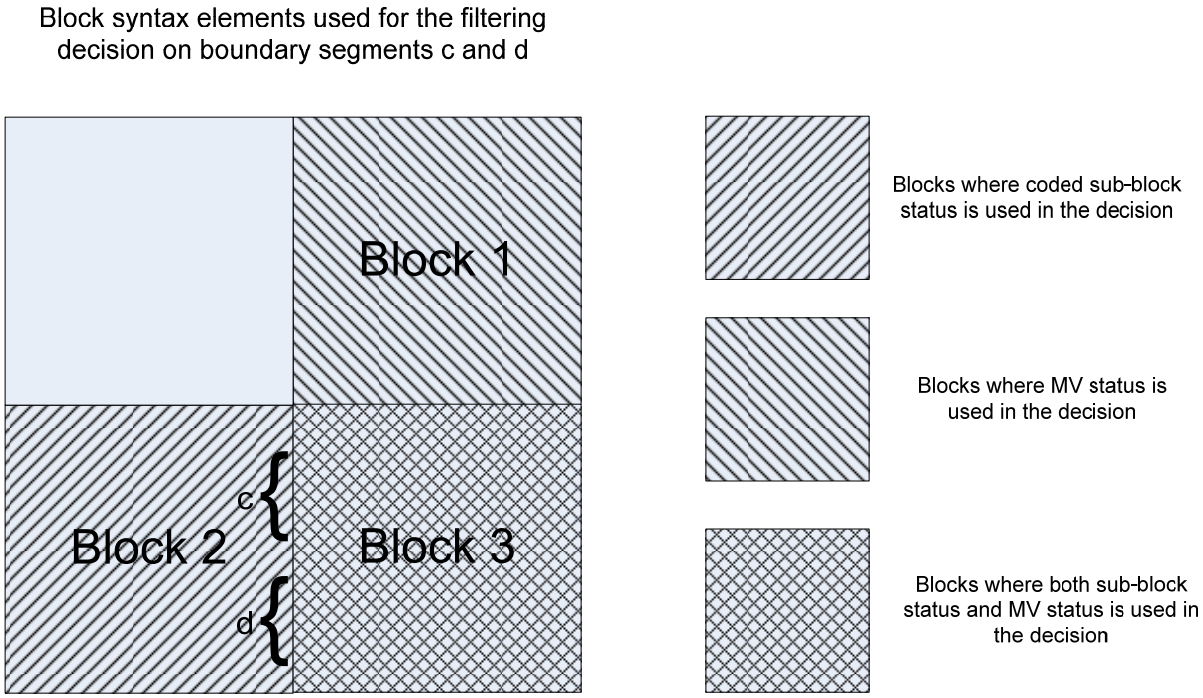


Figure 85: Overview Figure B to illustrate Exception 2

3. The decision of which block boundary segments to filter does not match the intended behavior if either of the blocks was coded using the 4x4 transform. The full top or left block boundary of the current block shall be filtered if either block uses a 4x4 transform.

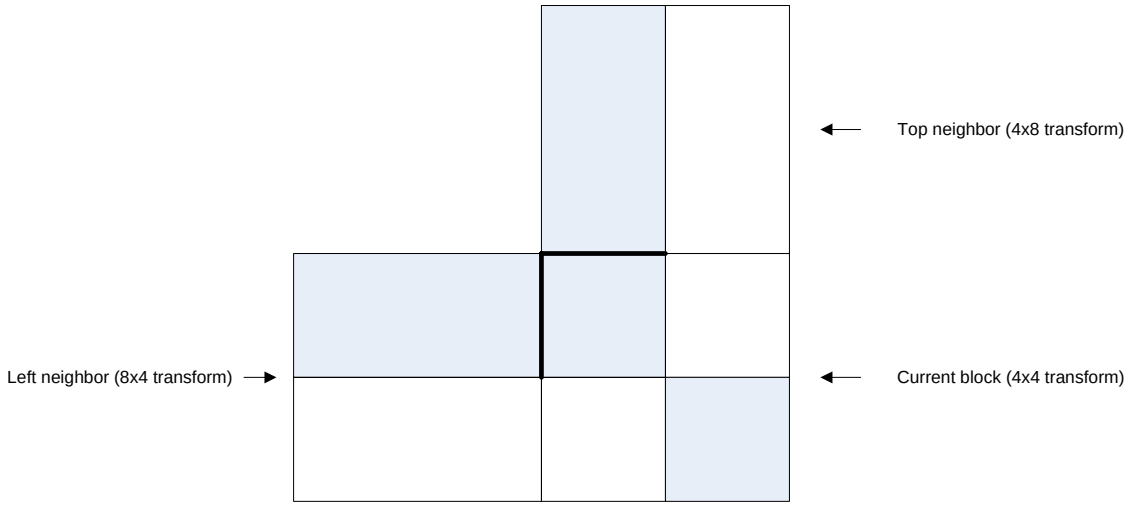
Therefore:

- If the current block was coded using the 4x4 transform then both the 8 pixel top boundary and the 8 pixel left boundary shall be filtered regardless of the subblock pattern of any of the blocks.
- If the current block was coded using the 8x8, 8x4 or 4x8 transform and the block above was coded using the 4x4 transform then the 8 pixel top boundary shall be filtered regardless of the subblock pattern of any of the blocks.
- If the current block was coded using the 8x8, 8x4 or 4x8 transform and the block to the left was coded using the 4x4 transform then the 8 pixel left boundary shall be filtered regardless of the subblock pattern of any of the blocks.

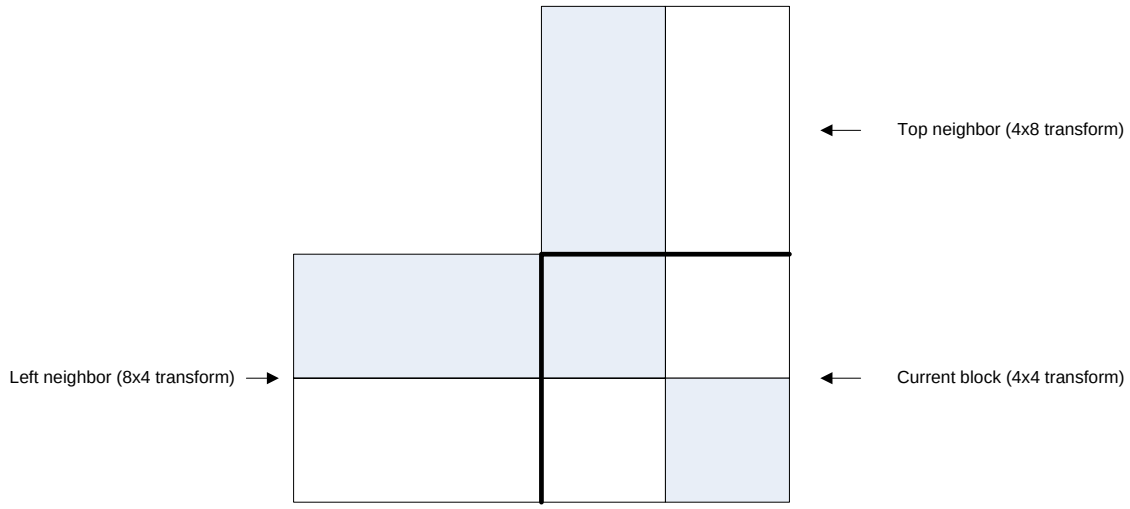
For example, if the block currently being filtered is coded using the 4x4 transform and the block above was coded using the 4x8 transform then the entire 8 pixel top boundary is filtered regardless of the subblock pattern of the 4x8 or 4x4 transformed blocks. This differs from the generic behavior which is to filter a subblock boundary only if at least one of subblocks contains a non-zero coefficient. So for example, in the generic case, if the upper-left subblock of the 4x4 transformed block and the left 4x8 transform of the block above do not contain any non-zero coefficients, then the first 4 pixels in the top boundary between these subblocks are not filtered.

One exception to exception 3 is for the case where the macroblock is coded using TTMB == 4x4 Transform. In this case, all blocks in the macroblock use the 4x4 transform. Further, if the coded-block status for a block is 0 (no coefficients present in the entire block) then it is considered to be an 8x8 transform block for the purposes of the in-loop deblocking decision process. For example, the macroblock was coded using TTMB == 4x4 and blocks 0 and 1 both had the coded-block status == 0 (no coefficients present in either block) then exception 3 does not apply and the boundary is filtered as intended.

A high-level overview to illustrate exception 3 is shown in Figure 86, Figure 87 and Figure 88, where shaded subblocks indicate that at least one non-zero coefficient is present, bold lines indicate that the boundary is filtered, and each block is assumed to have the same motion vector.



Intended behavior



Exception 3 behavior

Figure 86: Overview Figure A to illustrate Exception 3

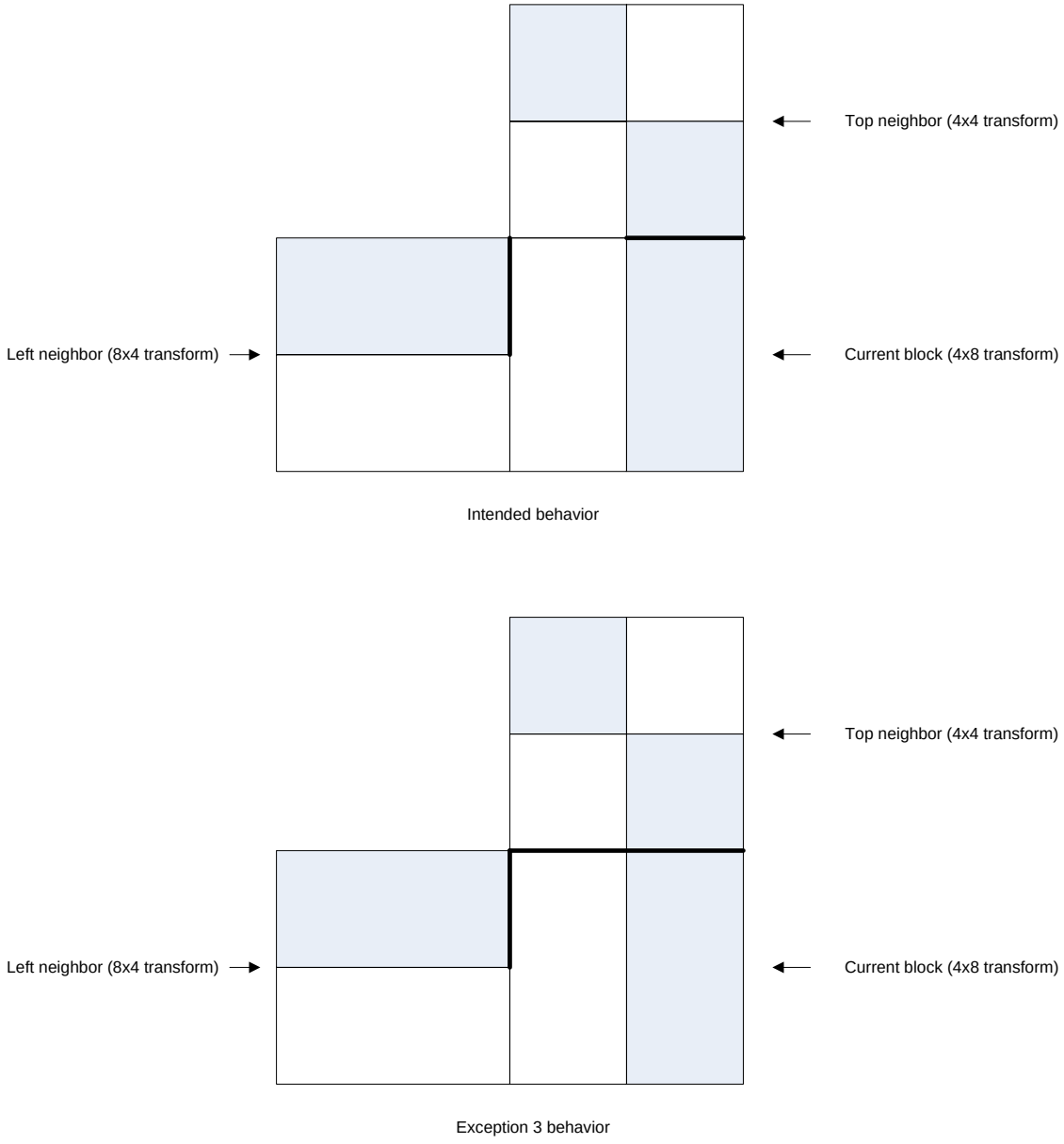


Figure 87: Overview Figure B to illustrate Exception 3

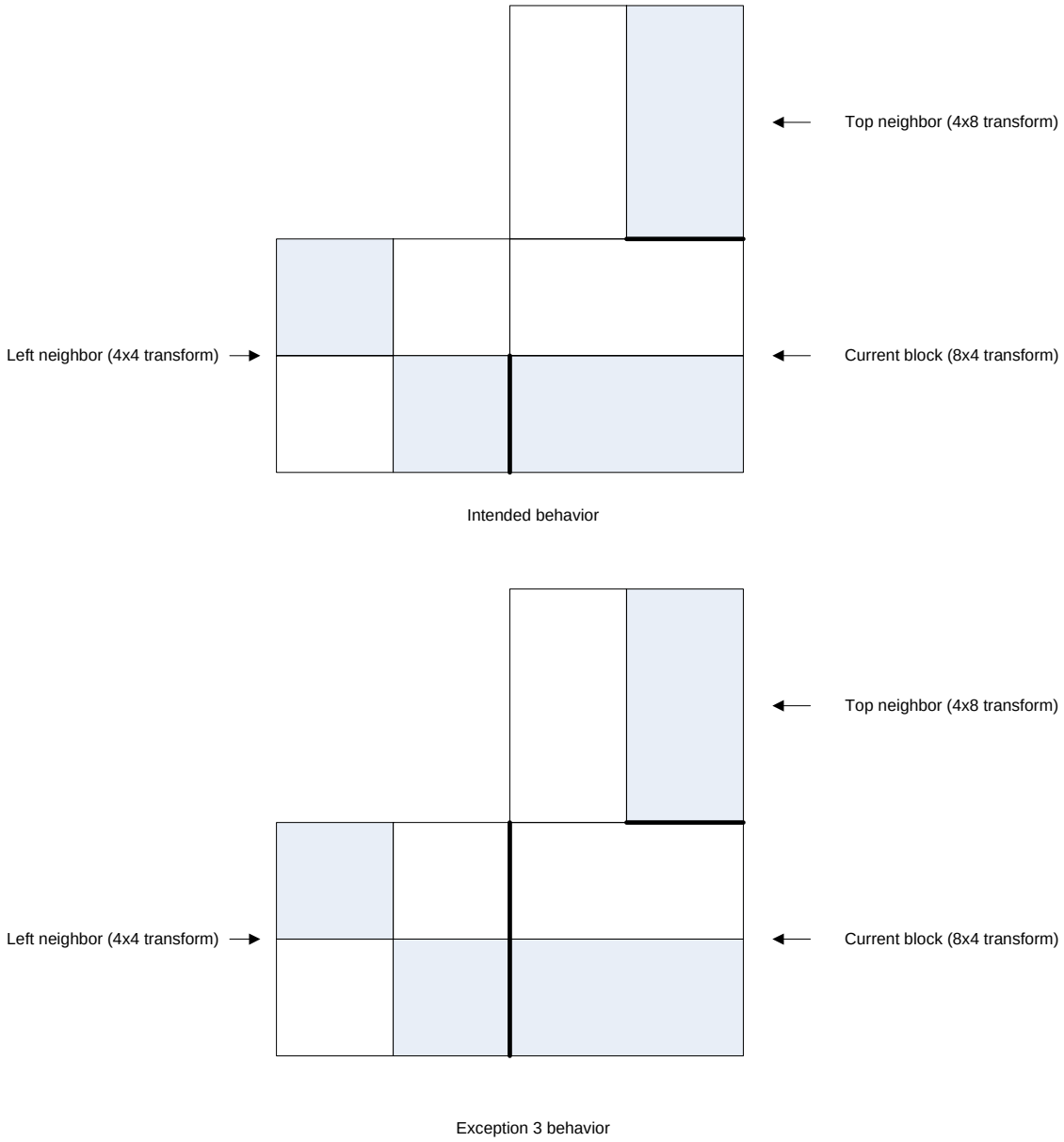


Figure 88: Overview Figure C to illustrate Exception 3

4. The decision criteria for filtering color-difference block boundaries shall use the range-limited color-difference motion vectors (iCMvXComp and iCMvYComp) as calculated in section 8.3.6.5.

The interaction between the exceptions described in exceptions 2 and 3 is as follows:

If block 1 or block 3 (or both) are coded using the 4x4 transform then the entire 8 pixel left boundary of block 3 shall be filtered.

8.7 Bitplane Coding

This section describes the bitplane coding scheme.

Certain macroblock-specific information can be encoded in one binary symbol per macroblock. For example, whether or not any information is present for a macroblock (i.e., whether or not it is skipped) can be signaled with one binary symbol or bit. In these cases, the status for all macroblocks in a frame may be coded as a bitplane and transmitted in the frame header. The only exception to this rule is if the bitplane coding mode (described below) is set to 'Raw' mode. In this case, the status for each macroblock shall be coded as one bit per symbol, and transmitted along with other

macroblock level syntax elements. 'Raw' mode shall be the only allowed bitplane mode when multiple slices are used to code the frame. Bitplane coding is used in seven different cases to signal information about the macroblocks in a frame. These are:

1) signaling skipped macroblocks,

2) signaling field or frame macroblock mode,

3) signaling 1-MV or 4-MV motion vector mode for each macroblock,

4) signaling of conditional overlap flag (OVERLAP) in I frames in advanced profile,

5) signaling of ACPRED flag in advanced profile,

6) signaling of DIRECTMB flag in B pictures, and

7) signaling of FORWARDMB flag in interlace field coded pictures.

Picture-level bitplane coding shall be used to code two-dimensional binary arrays. The size of each array is *rowMB* × *colMB*, where *rowMB* and *colMB* are the number of macroblock rows and columns respectively. Within the bitstream, each array is coded as a set of consecutive bits. One of seven modes shall be used to code each array.

The seven modes are enumerated below.

1. *Raw mode* – coded as one bit per symbol, and transmitted as part of MB level syntax.
2. *Normal-2 mode* – two symbols coded jointly
3. *Differential*-2 *mode* – differential coding of bitplane, followed by coding two residual symbols jointly
4. *Normal*-6 *mode* – six symbols coded jointly
5. *Differential*-6 *mode* – differential coding of bitplane, followed by coding six residual symbols jointly
6. *Rowskip mode* – one bit skip to signal rows with no set bits
7. *Columnskip mode* – one bit skip to signal columns with no set bits

Section 7.2 defines the syntax elements that make up the bitplane coding scheme. The following sections define how to decode the bitstream and reconstruct the bitplane.

8.7.1 INVERT

The INVERT syntax element shown in the syntax diagram of Figure 32 is a one bit code. If IMODE != Diff-2 && IMODE != Diff-6 modes, and if INVERT == 1, the value of the interpreted bitplane shall be inverted. If IMODE != Diff-2 && IMODE != Diff-6 modes, and if INVERT == 0, the value of the interpreted bitplane shall not be inverted. If IMODE == Diff-2 || IMODE == Diff-6 modes, the value of INVERT syntax shall be used to control the **Diff**$^{-1}$ operation as defined in section 8.7.3.8. The value of this bit shall be ignored when the raw mode is used.

Note: Invert can be set to 1 when the bitplane has more set bits than zero bits, and can be set to 0 otherwise.

8.7.2 IMODE

The IMODE syntax element shown in the syntax diagram of Figure 32 signals the mode used to code the bitplane. The seven modes are described in section 8.7.3. The IMODE syntax element shall be as defined in Table 69.

8.7.3 DATABITS

The DATABITS syntax element shown in the syntax diagram of Figure 32 shall be an entropy coded stream of symbols that is based on the coding mode. DATAMB consists of *rowMB* × *colMB* binary symbols. The seven coding modes are defined in the following sections.

8.7.3.1 Raw mode

In this mode, the bitplane shall be decoded as one bit per symbol, and shall be present as part of the macroblock layer.

8.7.3.2 Normal-2 mode

If *rowMB* × *colMB* is odd, the first symbol shall be decoded with a 1 bit word. Subsequent symbols shall be decoded pair-wise, in natural scan order. The binary VLC table in Table 80 shall be used to decode symbol pairs.

Table 80: Norm-2/Diff-2 Code Table

SYMBOL 2N	SYMBOL 2N + 1	CODEWORD
0	0	0b
1	0	100b
0	1	101b
1	1	11b

8.7.3.3 Diff-2 mode

The Normal-2 method shall be used to produce the bitplane as defined in section 8.7.3.2 and then the **Diff^{-1}** operation shall be applied to the bitplane as defined in section 8.7.3.8.

8.7.3.4 Normal-6 mode

In the *Norm-6* mode, the bitplane shall be coded in groups of six pixels. These pixels shall be grouped into either 2x3 or 3x2 tiles. The bitplane shall be tiled maximally using a set of rules defined in the next paragraph, and the remaining pixels shall be decoded using a variant of *row-skip* and *column-skip* modes. The tiles are signaled in natural scan order.

2x3 "vertical" tiles shall be used if and only if *rowMB* is a multiple of 3 and *colMB* is not. Else, 3x2 "horizontal" tiles shall be used, as shown in Figure 89.

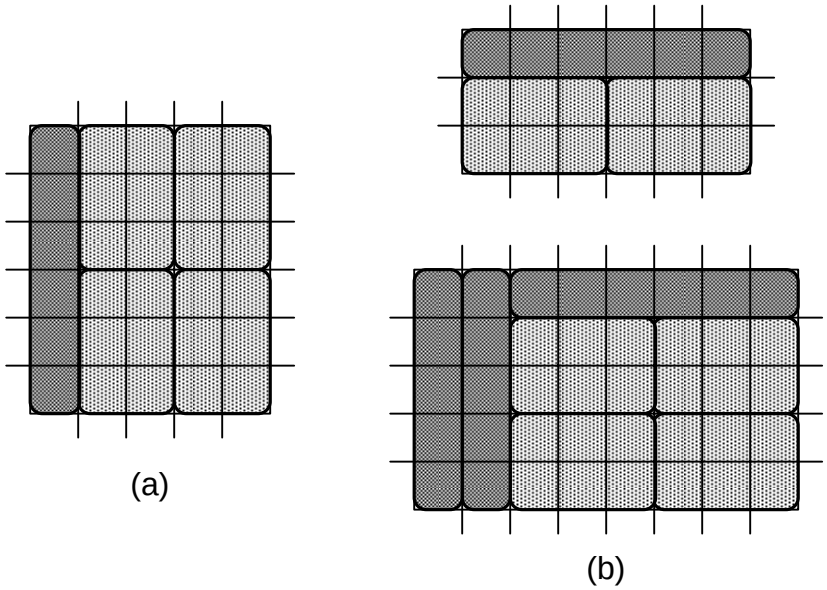


Figure 89: An example of 2x3 "vertical" tiles (a) and two examples of 3x2 "horizontal" tiles (b) – the elongated dark rectangles are 1 pixel wide and encoded using row-skip and column-skip coding.

When the picture is tiled as shown in Figure 89, (with linear tiles along the top and left edges of the picture), the coding order of the tiles shall be as follows. The 6-element tiles shall be decoded first, followed by the *column-skip* and *row-skip* coded linear tiles. If the array size is a multiple of 2x3 or of 3x2, the *column-skip* and *row-skip* linear tiles shall not exist and the bitplane is perfectly tiled.

The pseudo-code of Figure 90 shall specify the decoding order of the tiles.

```
//rowMB = Number of macroblock rows
//colMB = Number of macroblock columns
if (rowMB is multiple of 3 && colMB is not multiple of 3) {
   decode 2x3 vertical tiles with starting MB   position colMB%2;
```

```
   NumColSkipTiles = colMB%2;
   NumRowSkipTiles = 0;
}
else {
   decode 3x2 horizontal tiles with starting MB position (rowMB%2)*colMB+colMB%3
   NumColSkipTiles = colMB%3
   NumRowSkipTiles = rowMB%2
}

if (NumColSkipTiles) {
  //NumColSkipTiles can be 1 or 2
   for (i =0; i< NumColSkipTiles; i++)
     decode column-skip tiles with starting MB position i
}

if (NumRowSkipTiles) {
   //NumRowSkipTiles can only be 1
    decode row-skip tiles with starting MB position NumColSkipTiles
}
```

Figure 90: Decoding Norm-6 Bitplane: Pseudo-code

The 6-element rectangular tiles shall be decoded using Table 81 which has the following structure:

Let N be the number of set bits in the tile, i.e. $0 \le N \le 6$. For $N < 3$, a VLC is used to decode the tile. For $N = 3$, a fixed length escape shall be followed by a 5 bit fixed length code. For $N > 3$, another fixed length escape shall be followed by a VLC.

Note: For $N > 3$, the VLC which follows the escape is identical to the VLC used to code the complement of this tile for the $N < 3$ case. The fixed length escape used for the case of $N > 3$ differs from the fixed length escape of the case of $N = 3$.

Each rectangular tile contains 6 decoded bits of information. Let k be the code associated with the tile, where $k = \sum_i b_i 2^i$, b_i is the binary value of the i^{th} bit in natural scan order within the tile. Hence $0 \le k < 64$. Table 81 shall be used to decode k.

Table 81: Code table for 3x2 and 2x3 tiles

	VLC / Escape symbol		Followed by	
k	Codeword	Codelength	Codeword	Codelength
0	1	1		
1	2	4		
2	3	4		
3	0	8		
4	4	4		
5	1	8		

6	2	8		
7	2	5	7	5
8	5	4		
9	3	8		
10	4	8		
11	2	5	11	5
12	5	8		
13	2	5	13	5
14	2	5	14	5
15	3	5	14	8
16	6	4		
17	6	8		
18	7	8		
19	2	5	19	5
20	8	8		
21	2	5	21	5
22	2	5	22	5
23	3	5	13	8
24	9	8		
25	2	5	25	5
26	2	5	26	5
27	3	5	12	8
28	2	5	28	5
29	3	5	11	8
30	3	5	10	8
31	3	5	7	4
32	7	4		
33	10	8		
34	11	8		
35	2	5	3	5
36	12	8		
37	2	5	5	5

38	2	5	6	5
39	3	5	9	8
40	13	8		
41	2	5	9	5
42	2	5	10	5
43	3	5	8	8
44	2	5	12	5
45	3	5	7	8
46	3	5	6	8
47	3	5	6	4
48	14	8		
49	2	5	17	5
50	2	5	18	5
51	3	5	5	8
52	2	5	20	5
53	3	5	4	8
54	3	5	3	8
55	3	5	5	4
56	2	5	24	5
57	3	5	2	8
58	3	5	1	8
59	3	5	4	4
60	3	5	0	8
61	3	5	3	4
62	3	5	2	4
63	3	5	1	1

8.7.3.5 Diff-6 mode

The Normal-6 method shall be used to produce the bitplane as defined in section 8.7.3.4 and then the **Diff**$^{-1}$ operation shall be applied to the bitplane as defined in section 8.7.3.8.

8.7.3.6 Row-skip mode

In the row-skip coding mode, all-zero rows shall be skipped with one bit overhead. The process shall be as shown in Figure 91.

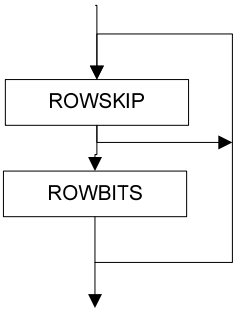


Figure 91: Syntax diagram of row-skip coding

From Figure 91, for each row, ROWSKIP is a 1-bit syntax element that shall always be present. If ROWSKIP == 0, the syntax element ROWBITS shall not be present and the entire row of symbols shall be set to zero. If ROWSKIP == 1, the syntax element ROWBITS shall be present and shall be decoded as one bit per symbol for the entire row.

In other words, if the entire row is zero, a zero bit is sent as the ROWSKIP symbol, and ROWBITS is skipped. If there is a set bit in the row, ROWSKIP is set to 1, and the entire row is sent raw (ROWBITS).

Rows shall be scanned from the top to the bottom of the frame.

8.7.3.7 Column-skip mode

Column-skip is the transpose of row-skip. Columns shall be scanned from the left to the right of the frame.

8.7.3.8 Diff^{-1}: Inverse differential decoding

If either differential mode (Diff-2 or Diff-6) is used, a bitplane of "differential bits" shall be first decoded using the corresponding normal modes (Norm-2 or Norm-6 respectively). The differential bits shall be used to regenerate the original bitplane. The regeneration process is a 2-D DPCM on a binary alphabet. In order to regenerate the bit at location (i, j), the predictor $b_p(i,j)$ shall be generated as follows (from bits b(i, j) at positions (i, j)):

$$b_p(i,j) = \begin{cases} A & i = j = 0, \; or \;\; b(i,j-1) \neq b(i-1,j) \\ b(0,j-1) & i == 0 \\ b(i-1,j) & otherwise \end{cases}$$

For the differential coding mode, the bitwise inversion process based on INVERT shall be not performed. However, the INVERT flag (**7.2.1**) is used in a different capacity to indicate the value of the symbol A for the derivation of the predictor shown above. More specifically, A shall be set to 0 if INVERT == 0, and A shall be set to 1 if INVERT == 1. The actual value of the bitplane shall be obtained by XOR-ing the predictor with the decoded differential bit value. In the above equation, b(i,j) is the bit at the i,jth position after final decoding (i.e. after doing Norm-2/Norm-6, followed by differential xor with its predictor).

8.8 Sync Markers (Simple and Main Profiles only)

Sync markers are defined sequences of bits that are inserted at important locations in the bitstream to clearly identify these locations. There are several reasons to insert sync markers – the important ones are for error resilience and for parallel decoding of the bitstream. Sync markers may be inserted in the simple and main profile bitstreams. The sequence level flag SYNCMARKER (Annex J.1.16) determines whether sync markers are enabled in the sequence. If they are enabled, sync markers may be sent only for I and P pictures. No sync markers shall be allowed in B pictures, including B pictures coded as Intra. When SYNCMARKER is enabled, all bitplanes shall be coded as raw bitplanes and the relevant data (e.g. 4-MV/1-MV, skipbit) shall be present at the macroblock level. Sync markers shall be placed only at byte boundaries.

The sync markers in simple/main profiles are not guaranteed to be unique. Sync markers are 24 bits in length and it is to be expected that even if they do randomly occur in a bitstream, such occurrences will be rare.

Note: Assuming a uniform distribution, it may be expected that sync marker is randomly emulated once in a 2^{24} byte long stream. For a bitrate of 1Mbps, this is equivalent to one random sync marker emulation every two minutes, or one occurrence every 3900 frames.

Sync markers may only occur at the start of a row of macroblocks (abbreviated as *MB row*). No sync marker shall be permitted in the first MB row. When the sequence level SYNCMARKER is enabled, a single bit shall be present at the end of every MB row, except for the last MB row in the frame, to indicate whether or not a sync marker follows. If this bit is one, it signals that no sync marker shall follow. If this bit is zero, the remainder of the current byte shall be flushed out. Subsequently, the 24 bit byte aligned sync marker shall be read from the bitstream.

Two sync markers are defined. These are the *short* and *long* sync markers. Both the codes are 24 bits in length, but the *payload* (data following the sync marker) differs in length. The short sync marker, whose hex representation is 0x0000AA, shall be followed by a 5 byte payload. The long sync marker, whose hex representation is 0x0000AB, shall be followed by an 11 byte payload.

Note: The first two bytes of both sync markers are zeros. This design makes the implementation of hardware-based sync marker detection schemes easier.

The decoder need not do anything with the payload, however the decoder shall decode bitstreams that have embedded sync markers, assuming that otherwise no errors are present. The payload may be used to transmit parity, error detection and error recovery information.

Figure 92 represents an example coded (I or P) frame. Subfigure (a) shows successive macroblocks coded when SYNCMARKER is zero, (b) shows coded macroblocks when SYNCMARKER is one but no sync markers are actually sent, and (c) shows the case when both long and short sync markers are sent in the frame. The frame header, sync markers and payloads are byte aligned. The trailing 0 or 1 in all but the last slice is sent when SYNCMARKER is 1. This is necessary to ensure byte flushing in the case that a sync marker is sent at the start of the next slice. "FB" in the figure stands for *flush bits* or the process of stuffing between zero and seven bits to reach the end of the current byte. The value of the flush bits is zero. "SC" and "PL" stand for sync marker and payload respectively. The sync marker 0x0000AA is followed by a 5 byte payload and sync marker 0x0000AB is followed by an 11 byte payload. There are no sync markers in B frames. For B frames, the entropy coded stream follows the order shown in Figure 92 regardless of whether SYNCMARKER is 0 or 1.

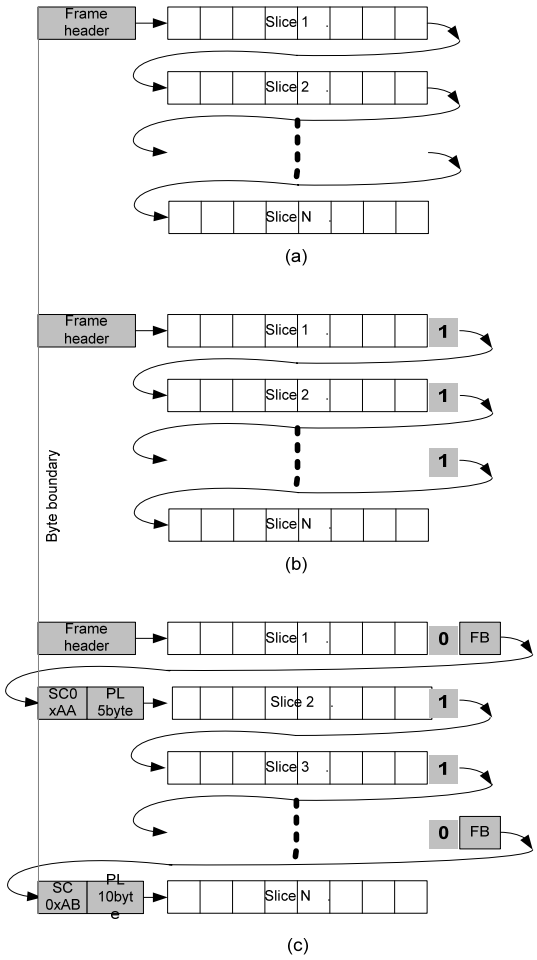


Figure 92: Sync markers– (a) shows sequence of entropy coded data with SYNCMARKER set to zero, (b) SYNCMARKER is 1 but no sync markers are actually sent and (c) SYNCMARKER is 1, a long and a short sync marker are sent, some slices do not have sync markers

8.9 Pan Scan

Pan scan window information shall be present only in the advance profile picture headers if the entry point header syntax element PANSCAN_FLAG == 1 (**6.2.3**). In this case, each picture header in the entry point segment has the PS_PRESENT syntax element, as defined in section 7.1.1.20. If PS_PRESENT == 1 then for each window in the frame there are four syntax elements – PS_HOFFSET (**7.1.1.21**), PS_VOFFSET (**7.1.1.22**), PS_WIDTH (**7.1.1.23**) and PS_HEIGHT (**7.1.1.24**) - that define the size and location of the window within the frame. See Annex I.5 for information on alternative methods of signaling pan scan regions.

8.9.1 Number of Pan Scan Windows

If PS_PRESENT == 1, then there are from one to four pan scan windows in each frame. The number of pan scan windows is determined by the sequence header syntax elements: INTERLACE (**6.1.9**), PULLDOWN (**6.1.8**) and PSF (**6.1.13**) and the frame header syntax elements RFF (**7.1.1.18**) and RPTFRM (**7.1.1.19**). The pseudo-code of Figure 93 defines how the number of pan scan windows shall be determined.

```
If (INTERLACE == 1 && PSF == 0)
{
    If (PULLDOWN == 1)
        NumberOfPanScanWindows = 2 + RFF
    else
```

```
        NumberOfPanScanWindows = 2
}
else
{
    If (PULLDOWN == 1)
        NumberOfPanScanWindows = 1 + RPTFRM
    else
        NumberOfPanScanWindows = 1
}
```

Figure 93: Pseudo-code for Computing Number of Pan Scan Windows

If the sequence header INTERLACE == 1 then there may be one pan scan window for each displayed field in the frame. Therefore there are either 2 (for the case where RFF == 0) or 3 (for the case where RFF == 1). If the sequence header INTERLACE == 0, then there may be one pan scan window for each displayed frame. Since RPTFRM defines how many times the current frame is repeated for display, the number of pan scan windows is 1 + RPTFRM.

For each pan window there shall be a set of four pan scan window syntax elements in the frame header: PS_HOFFSET, PS_VOFFSET, PS_WIDTH and PS_HEIGHT. The order of the pan windows in the frame header bitstream shall be the same as the display order of the fields or frames – meaning that the first set of pan scan window syntax elements shall correspond to the first field or frame in display order.

8.9.2 Pan Scan Parameters

For each pan scan window, the PS_HOFFSET, PS_VOFFSET, PS_WIDTH and PS_HEIGHT shall define the size and location of the window within the main frame.

PS_HOFFSET is an 18 bit value that shall define the horizontal distance between the left border of the main frame and the left border of the pan scan window. PS_HOFFSET shall be in units of 1/16 luma samples. Therefore the maximum horizontal offset shall be 16383 $^{15}/_{16}$ luma samples.

PS_VOFFSET is an 18 bit value that shall define the vertical distance between the top border of the main frame and the top border of the pan scan window. PS_HOFFSET shall be in units of 1/16 luma samples. Therefore the maximum vertical offset shall be 16383 $^{15}/_{16}$ luma samples.

PS_WIDTH is a 14 bit value that shall define the width of the pan scan window in luma samples. The maximum width of the pan scan window shall be 16384 luma samples.

PS_HEIGHT is a 14 bit value that shall define the height of the pan scan window in luma samples. The maximum height of the pan scan window shall be 16384 luma samples.

8.9.3 Pan Scan Restrictions

If the entry point header syntax element PANSCAN_FLAG == 1 then the first coded frame in the entry point header shall contain pan scan window information – i.e., PS_PRESENT shall be 1 for the first coded frame.

If PS_PRESENT == 0 for any of the other frames in the entry point segment then the pan scan window size and offsets shall be equal to the most recent pan scan window parameters in display order.

For interlace fields, the pan scan window dimensions and offsets shall be specified at frame resolution.

9 Interlace Bitstream Syntax and Semantics

9.1 Picture-level Syntax and Semantics

This section defines the syntax and semantics of the picture layer, slice layer, macroblock layer, and block layer, when a picture is coded in interlace mode. Figure 94 through Figure 106 define the bitstream elements that make up each layer. Table 82 through Table 95 shall define the syntax elements of a picture that is coded in interlace mode. A picture is coded in interlace mode only in advanced profile.

The syntax of a skipped interlaced-frame coded picture shall be identical to that of a skipped progressive picture as defined in Figure 20 and Table 23. A skipped interlaced-frame coded picture shall be present only if the sequence level syntax element INTERLACE == 1 (6.1.9), and if present, the value of the FCM syntax element (7.1.1.15) shall be 10b.

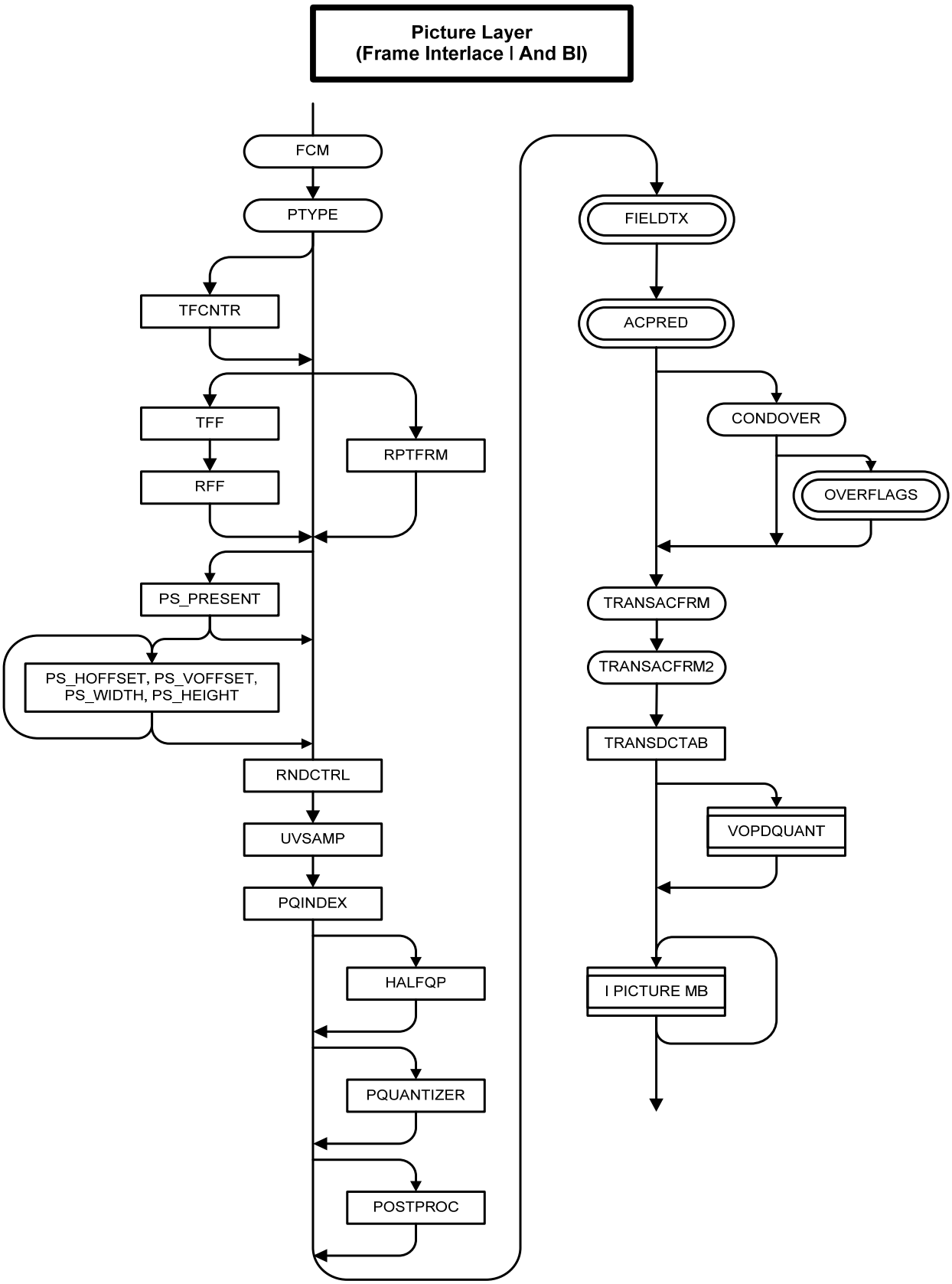


Figure 94: Syntax diagram for the picture layer bitstream in Interlace Frame I and BI picture

Table 82: Interlaced Frame I and BI picture layer bitstream for Advanced Profile

I AND BI INTERLACED FRAME PICTURE ( ) {	Number of bits	Descriptor	Reference
FCM	Variable size	Vlclbf	9.1.1.1 INTERLACE ==1and FCM==10b here.
PTYPE	Variable size	Vlclbf	9.1.1.2
if (TFCNTRFLAG) {	8	Uimsbf	6.1.10
TFCNTR			9.1.1.3
}			
if (PULLDOWN == 1) {			6.1.8
if (PSF == 1) {			6.1.13
RPTFRM	2	Uimsbf	9.1.1.6
}			
else {			
TFF	1	Uimsbf	9.1.1.4
RFF	1	Uimsbf	9.1.1.5
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	Uimsbf	9.1.1.7
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	Uimsbf	9.1.1.8
PS_VOFFSET	18	Uimsbf	9.1.1.9
PS_WIDTH	14	Uimsbf	9.1.1.10
PS_HEIGHT	14	Uimsbf	9.1.1.11
}			
}			
}			
RNDCTRL	1	Uimsbf	9.1.1.12
UVSAMP	1	Uimsbf	9.1.1.13
PQINDEX	5	Uimsbf	9.1.1.14

if (PQINDEX <= 8) {			
HALFQP	1	Uimsbf	9.1.1.15
}			
if (QUANTIZER == 01) {			6.2.11 (Annex J)
PQUANTIZER	1	Uimsbf	9.1.1.16
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	Uimsbf	9.1.1.17
}			
FIELDTX	Bitplane		9.1.1.18
ACPRED	Bitplane		9.1.1.19
if (OVERLAP == 1 && PQUANT <= 8) {			OVERLAP (6.2.10, Annex J) PQUANT shall be computed from PQINDEX as defined in 7.1.1.6
CONDOVER	Variable size	Vlclbf	9.1.1.20
if (CONDOVER == 11) {			
OVERFLAGS	Bitplane		9.1.1.21
}			
}			
TRANSACFRM	Variable size	Vlclbf	9.1.1.22
TRANSACFRM2	Variable size	Vlclbf	9.1.1.23
TRANSDCTAB	1	Uimsbf	9.1.1.24
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	Vlclbf	9.1.1.25
}			
for (“all macroblocks”) {			‘all macroblocks’ represents all macroblocks in this BDU.
I INTERLACED FRAME MB ( )			**Table 90**
}			
}			

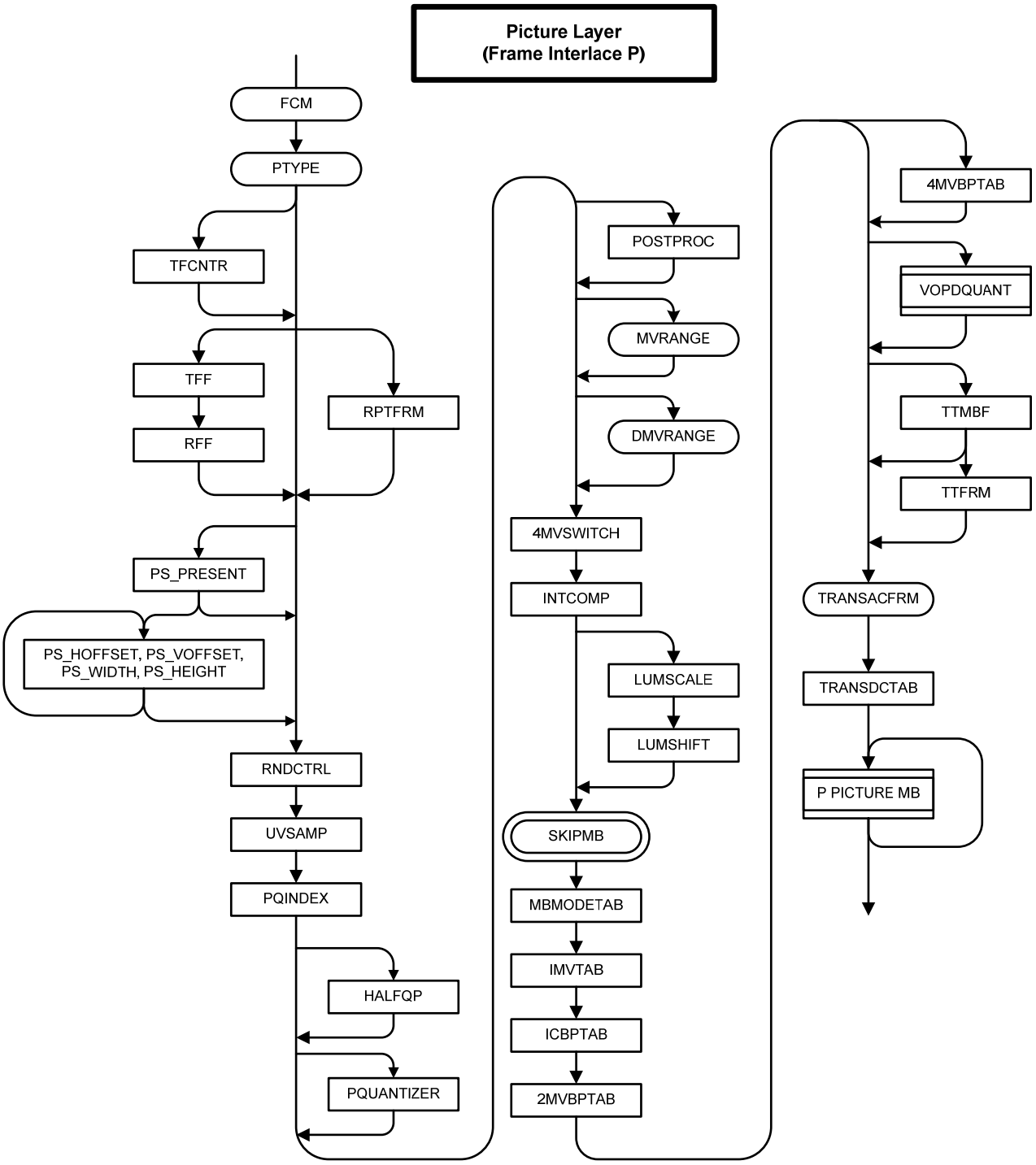


Figure 95: Syntax diagram for the picture layer bitstream in Interlace Frame P picture

Table 83: Interlaced Frame P picture layer bitstream for Advanced Profile

P INTERLACED FRAME PICTURE ( ) {	Number of bits	Descriptor	Reference
FCM	Variable size	vlclbf	9.1.1.1 INTERLACE ==1and FCM==10b here.

PTYPE	Variable size	vlclbf	9.1.1.2
if (TFCNTRFLAG) {	8	uimsbf	6.1.10
TFCNTR			9.1.1.3
}			
if (PULLDOWN == 1) {			6.1.8
if (PSF == 1) {			6.1.13
RPTFRM	2	uimsbf	9.1.1.6
}			
else {			
TFF	1	uimsbf	9.1.1.4
RFF	1	uimsbf	9.1.1.5
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	9.1.1.7
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	9.1.1.8
PS_VOFFSET	18	uimsbf	9.1.1.9
PS_WIDTH	14	uimsbf	9.1.1.10
PS_HEIGHT	14	uimsbf	9.1.1.11
}			
}			
}			
RNDCTRL	1	uimsbf	9.1.1.12
UVSAMP	1	uimsbf	9.1.1.13
PQINDEX	5	uimsbf	9.1.1.14
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	9.1.1.15
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	9.1.1.16

}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	9.1.1.17
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	9.1.1.26
}			
if (EXTENDED_DMV == 1) {			6.2.14
DMVRANGE	Variable size	vlclbf	9.1.1.27
}			
4MVSWITCH	1	uimsbf	9.1.1.28
INTCOMP	1	uimsbf	9.1.1.29
if (INTCOMP) {			
LUMSCALE	6	uimsbf	9.1.1.30
LUMSHIFT	6	uimsbf	9.1.1.31
}			
SKIPMB	Bitplane		9.1.1.32
MBMODETAB	2	uimsbf	9.1.1.33
IMVTAB	2	uimsbf	9.1.1.34
ICBPTAB	3	uimsbf	9.1.1.35
2MVBPTAB	2	uimsbf	9.1.1.36
if (4MVSWITCH == 1) {			9.1.1.28
4MVBPTAB	2	uimsbf	9.1.1.37
}			
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	9.1.1.25
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	9.1.1.38
if (TTMBF == 1) {			
TTFRM	2	uimsbf	9.1.1.39
}			
}			
TRANSACFRM	Variable size	vlclbf	9.1.1.22
TRANSDCTAB	1	uimsbf	9.1.1.24

for (‘all macroblocks’) {			‘all macroblocks’ represents all macroblocks in this BDU.
P INTERLACED FRAME MB ( )			**Table 91**
}			
}			

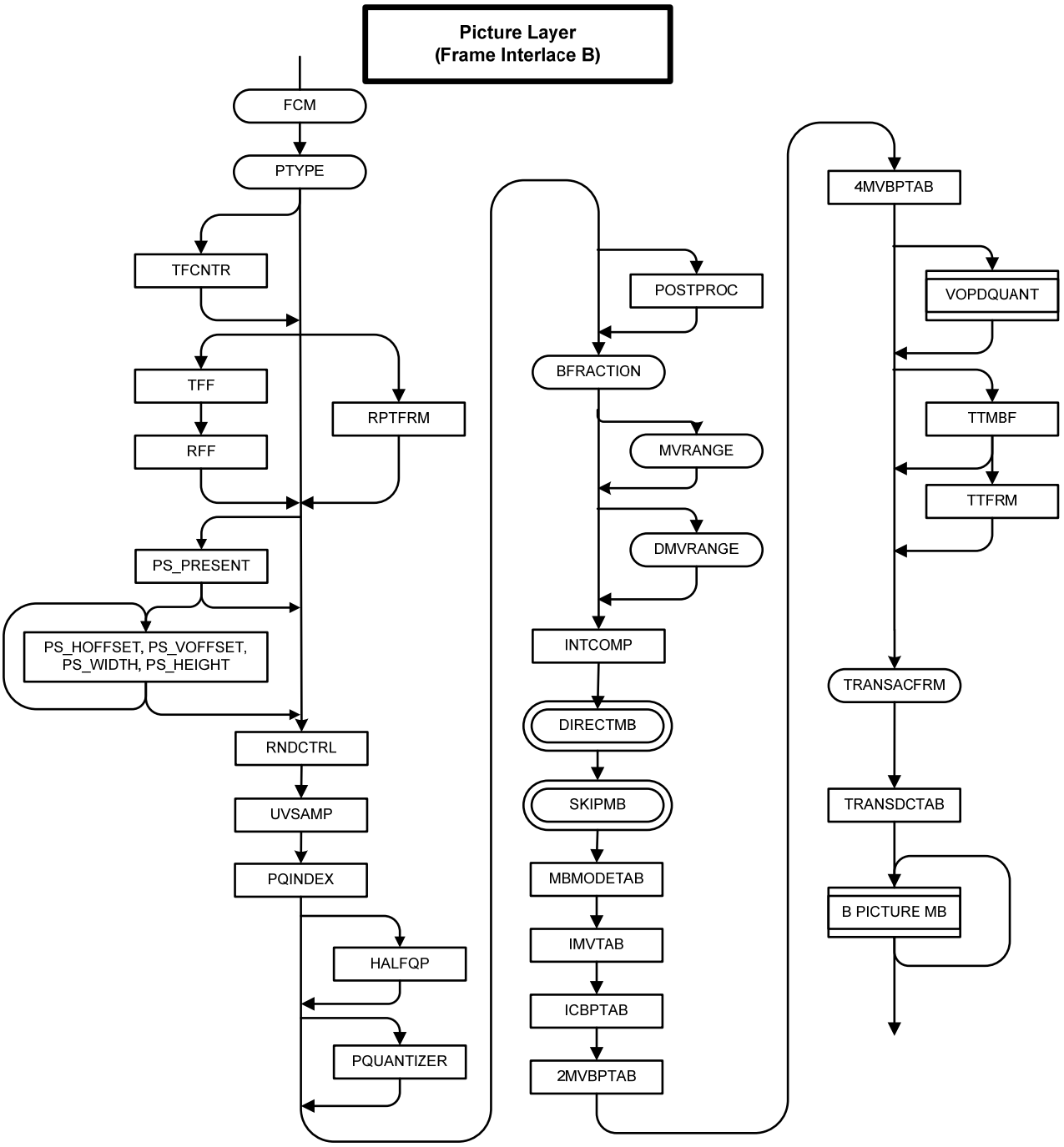


Figure 96: Syntax diagram for the picture layer bitstream in Interlace Frame B picture

Table 84: Interlaced Frame B picture layer bitstream for Advanced Profile

B INTERLACED FRAME PICTURE ( ) {	Number of bits	Description	Reference
FCM	Variable size	vlclbf	9.1.1.1 INTERLACE ==1and FCM==10b here.
PTYPE	Variable size	vlclbf	9.1.1.2
if (TFCNTRFLAG) {	8	uimsbf	6.1.10
TFCNTR			9.1.1.3
}			
if (PULLDOWN == 1) {			6.1.8
if (PSF == 1) {			6.1.13
RPTFRM	2	uimsbf	9.1.1.6
}			
else {			
TFF	1	uimsbf	9.1.1.4
RFF	1	uimsbf	9.1.1.5
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	9.1.1.7
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	9.1.1.8
PS_VOFFSET	18	uimsbf	9.1.1.9
PS_WIDTH	14	uimsbf	9.1.1.10
PS_HEIGHT	14	uimsbf	9.1.1.11
}			
}			
}			
RNDCTRL	1	uimsbf	9.1.1.12
UVSAMP	1	uimsbf	9.1.1.13
PQINDEX	5	uimsbf	9.1.1.14
if (PQINDEX <= 8) {			

HALFQP	1	uimsbf	9.1.1.15
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	9.1.1.16
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	9.1.1.17
}			
BFRACTION	Variable size	vlclbf	9.1.1.40
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	9.1.1.26
}			
if (EXTENDED_DMV == 1) {			6.2.14
DMVRANGE	Variable size	vlclbf	9.1.1.27
}			
INTCOMP	1	uimsbf	9.1.1.29 Shall always be FALSE
DIRECTMB	Bitplane		9.1.1.41
SKIPMB	Bitplane		9.1.1.32
MBMODETAB	2	uimsbf	9.1.1.33
IMVTAB	2	uimsbf	9.1.1.34
ICBPTAB	3	uimsbf	9.1.1.35
2MVBPTAB	2	uimsbf	9.1.1.36
4MVBPTAB	2	uimsbf	9.1.1.37
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	9.1.1.25
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	9.1.1.38
if (TTMBF == 1) {			
TTFRM	2	uimsbf	9.1.1.39
}			
}			
TRANSACFRM	Variable size	vlclbf	9.1.1.22
TRANSDCTAB	1	uimsbf	9.1.1.24

for (“all macroblocks”) {			‘all macroblocks’ represents all macroblocks in this BDU.
B INTERLACED FRAME MB ( )			**Table 92**
}			
}			

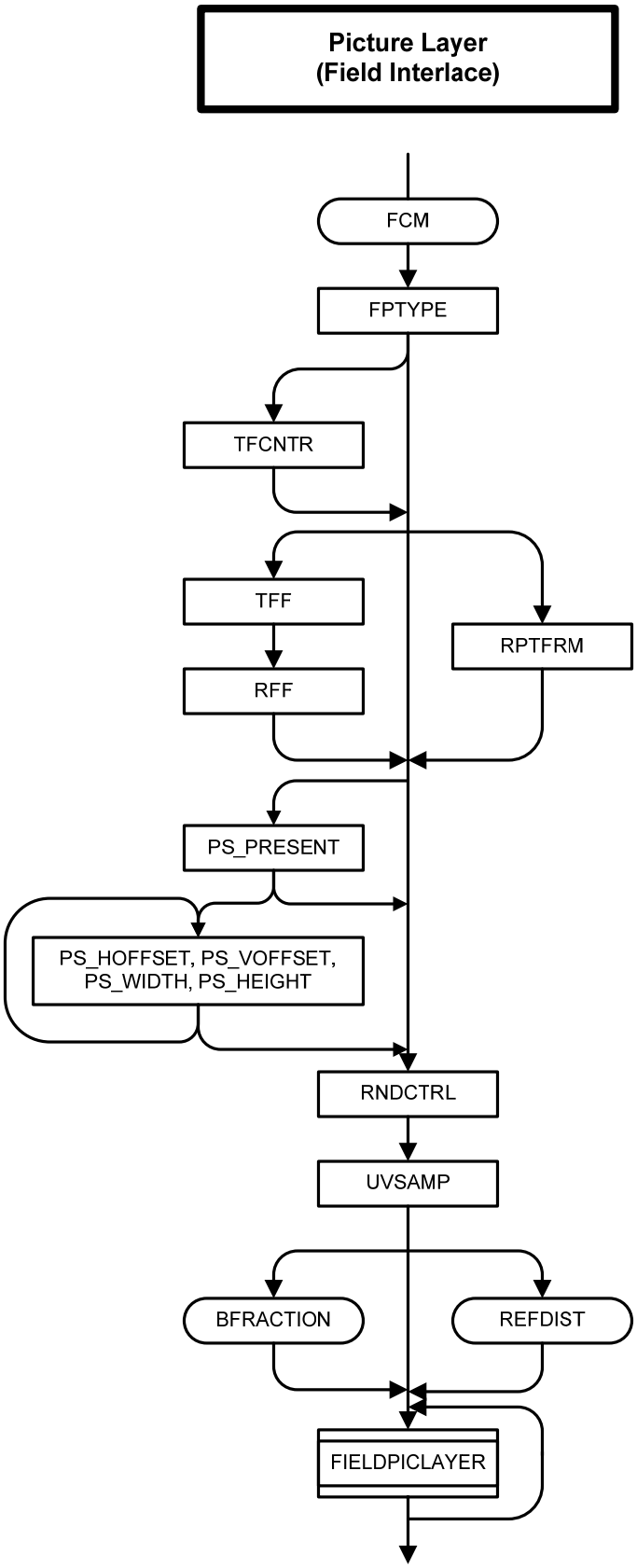


Figure 97: Syntax diagram for the picture layer bitstream in Interlace Field pictures for Field1

Table 85: Picture Layer bitstream for Field 1 of Interlace Field Picture for Advanced Profile

INTERLACE FIELD PICTURE FIELD1 ( ) {	Number of bits	Descriptor	Reference
FCM	Variable size	vlclbf	9.1.1.1 INTERLACE ==1and FCM==11b here. Field1 Picture shall be preceded by Frame start code as described Annex G.
FPTYPE	3	uimsbf	9.1.1.42
if (TFCNTRFLAG) {	8	uimsbf	6.1.10
TFCNTR			9.1.1.3
}			
if (PULLDOWN == 1) {			6.1.8
if (PSF == 1) {			6.1.13
RPTFRM	2	uimsbf	9.1.1.6
}			
else {			
TFF	1	uimsbf	9.1.1.4
RFF	1	uimsbf	9.1.1.5
}			
}			
if (PANSCAN_FLAG == 1) {			6.2.3
PS_PRESENT	1	uimsbf	9.1.1.7
if (PS_PRESENT == 1)			
{			
for (i = 0; i < (NumberOfPanScanWindows); i++)			NumberOfPanScanWindows is computed as shown in Figure 93 in 8.9.1
{			
PS_HOFFSET	18	uimsbf	9.1.1.8
PS_VOFFSET	18	uimsbf	9.1.1.9
PS_WIDTH	14	uimsbf	9.1.1.10
PS_HEIGHT	14	uimsbf	9.1.1.11
}			
}			
}			
RNDCTRL	1	uimsbf	9.1.1.12

UVSAMP	1	uimsbf	9.1.1.13
if (REFDIST_FLAG == 1 && (FPTYPE == I/I,I/P, P/I or P/P)) {			6.2.4, 9.1.1.42
REFDIST	Variable size	vlclbf	9.1.1.43
}			
if (FPTYPE == B/B, B/BI, BI/B or BI/BI) {			9.1.1.42
BFRACTION	Variable size	vlclbf	9.1.1.40
}			

if (FPTYPE == I/I, I/P, BI/B or BI/BI) {
// the first field is I or BI
I AND BI INTERLACE FIELDPIC( )
} else if (FPTYPE == P/I or P/P) {
// the first field is P
P INTERLACE FIELDPIC( )
} else { // FPTYPE == B/B or B/BI
// the first field is B
B INTERLACE FIELDPIC( )
}
			Field 1 (Table 87, Table 88, Table 89 according to FPTYPE)

}			

Table 86: Picture Layer bitstream for Field 2 of Interlace Field Picture for Advanced Profile

INTERLACE FIELD PICTURE FIELD2 ( ) {	Number of bits	Descriptor	Reference

if (FPTYPE == I/I, P/I, B/BI or BI/BI) {
// the second field is I or BI
I AND BI INTERLACE FIELDPIC( )
} else if (FPTYPE == I/P or P/P) {
// the second field is P
P INTERLACE FIELDPIC( )
} else { // FPTYPE == B/B or BI/B
// the second field is B
B INTERLACE FIELDPIC( )
}
			Field 2 (Table 87, Table 88, Table 89 according to FPTYPE)
			Field 2 picture layer shall be preceded by Field start code as described Annex G

}			

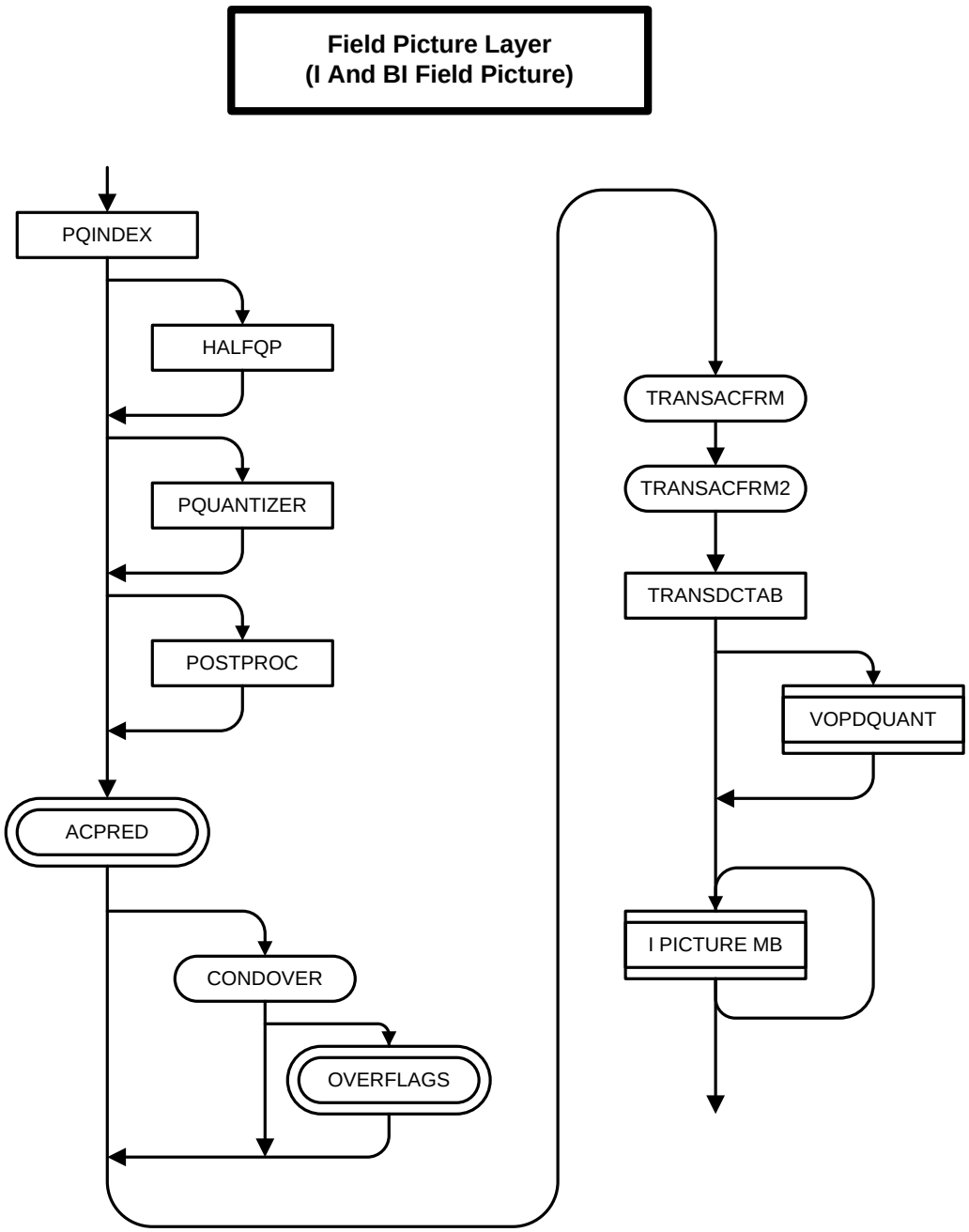


Figure 98: Syntax diagram for the field picture layer bitstream in Interlace I and BI Field pictures

Table 87: Field Interlace I and BI Field Picture Layer bitstream for Advanced Profile

I AND BI INTERLACE FIELDPIC( ) {	Number of bits	Descriptor	Reference
PQINDEX	5	uimsbf	9.1.1.14
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	9.1.1.15
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)

PQUANTIZER	1	uimsbf	9.1.1.16
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	9.1.1.17
}			
ACPRED	Bitplane		9.1.1.19
if (OVERLAP == 1 && PQUANT <= 8) {			OVERLAP 6.2.10. PQUANT computed from PQINDEX as defined in 7.1.1.6
CONDOVER	Variable size	vlclbf	9.1.1.20
if (CONDOVER == 11b) {			
OVERFLAGS	Bitplane		9.1.1.21
}			
}			
TRANSACFRM	Variable size	vlclbf	9.1.1.22
TRANSACFRM2	Variable size	vlclbf	9.1.1.23
TRANSDCTAB	1	uimsbf	9.1.1.24
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 9.1.1.25
}			
for ("all macroblocks") {			'all macroblocks' represents all macroblocks in this BDU.
I INTERLACE FIELD MB ( )			**Table 93**
}			
}			

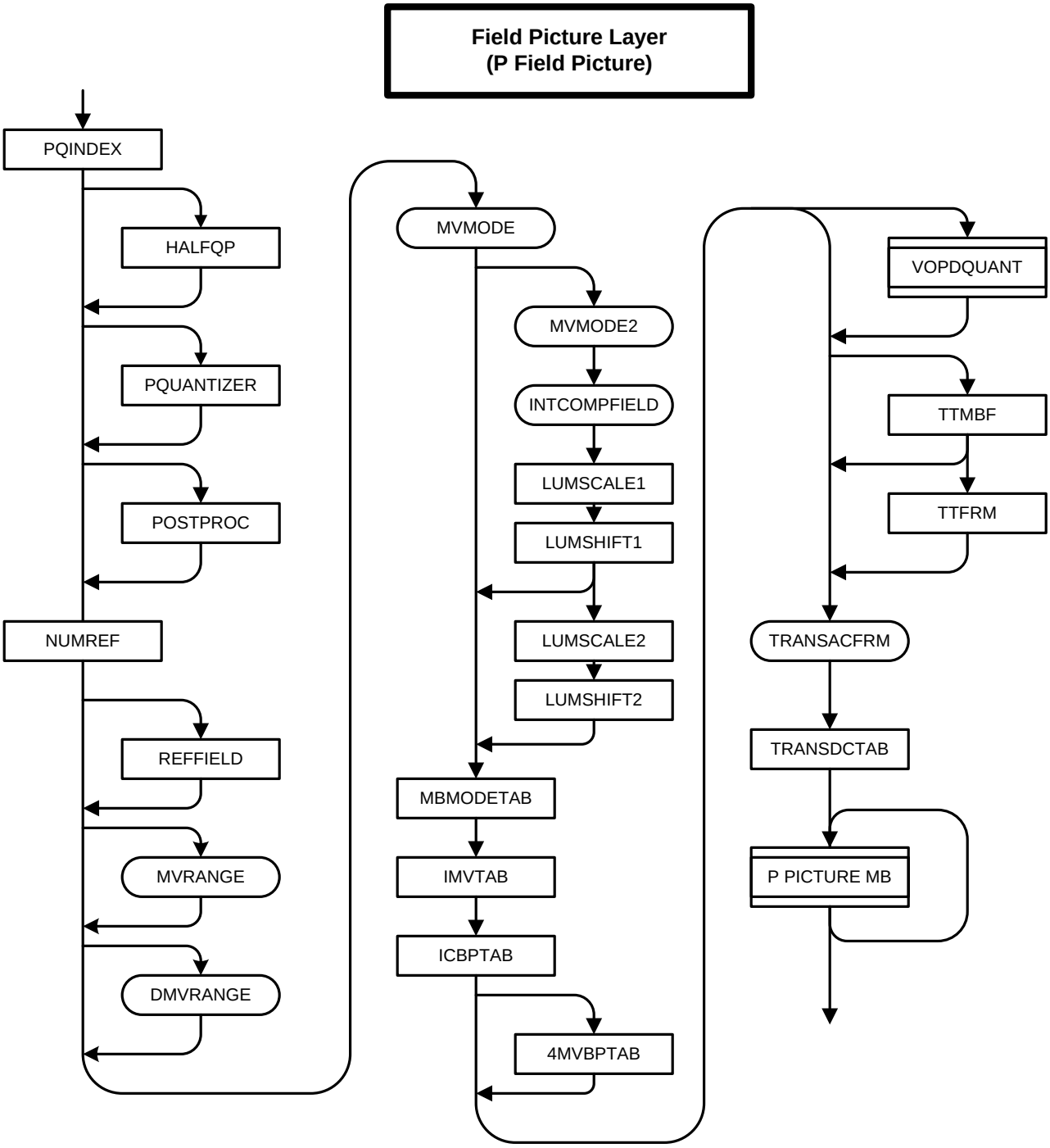


Figure 99: Syntax diagram for the field picture layer bitstream in Interlace P Field pictures

Table 88: Field Interlace P Field Picture layer bitstream for Advanced Profile

P INTERLACE FIELDPIC ( ) {	Number of bits	Descriptor	Reference
PQINDEX	5	uimsbf	9.1.1.14
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	9.1.1.15

}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	9.1.1.16
}			
if (POSTPROCFLAG == 1) {			6.1.5
POSTPROC	2	uimsbf	9.1.1.17
}			
NUMREF	1	uimsbf	9.1.1.44
if (NUMREF == 0) {			
REFFIELD	1	uimsbf	9.1.1.45
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	9.1.1.26
}			
if (EXTENDED_DMV == 1) {			6.2.14
DMVRANGE	Variable size	vlclbf	9.1.1.27
}			
MVMODE	Variable size	vlclbf	9.1.1.46
if (MVMODE == “intensity compensation”) {			
MVMODE2	Variable size	vlclbf	9.1.1.47
INTCOMPFIELD	Variable size	vlclbf	9.1.1.48
LUMSCALE1	6	uimsbf	9.1.1.49
LUMSHIFT1	6	uimsbf	9.1.1.50
if (INTCOMPFIELD == 1b) {			9.1.1.48
LUMSCALE2	6	uimsbf	9.1.1.51
LUMSHIFT2	6	uimsbf	9.1.1.52
}			
}			
MBMODETAB	3	uimsbf	9.1.1.33
IMVTAB	2 or 3	uimsbf	9.1.1.34
ICBPTAB	3	uimsbf	9.1.1.35
if (MVMODE ==”Mixed-MV”) {			9.1.1.46
4MVBPTAB	2	uimsbf	9.1.1.37
}			

if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 9.1.1.25
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	9.1.1.38
if (TTMBF == 1) {			
TTFRM	2	uimsbf	9.1.1.39
}			
}			
TRANSACFRM	Variable size	vlclbf	9.1.1.22
TRANSDCTAB	1	uimsbf	9.1.1.24
for ("all macroblocks") {			'all macroblocks' represents all macroblocks in this BDU.
P INTERLACE FIELD MB ( )			**Table 94**
}			
}			

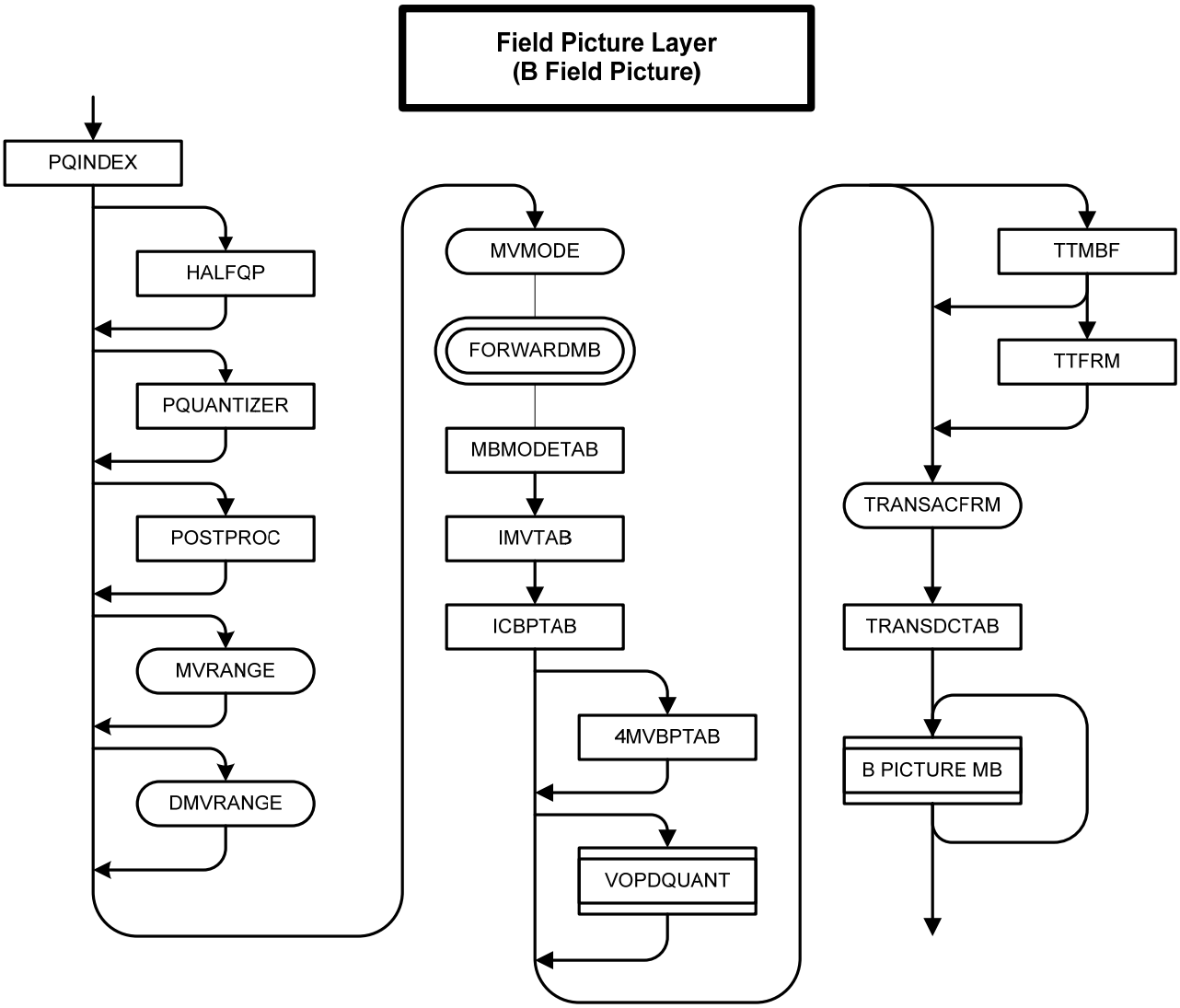


Figure 100: Syntax diagram for the field picture layer bitstream in Interlace B Field pictures

Table 89: Field Interlace B Field Picture layer bitstream for Advanced Profile

B INTERLACE FIELDPIC ( ) {	Number of bits	Descriptor	Reference
PQINDEX	5	uimsbf	9.1.1.14
if (PQINDEX <= 8) {			
HALFQP	1	uimsbf	9.1.1.15
}			
if (QUANTIZER == 01b) {			6.2.11 (Annex J)
PQUANTIZER	1	uimsbf	9.1.1.16
}			
if (POSTPROCFLAG == 1) {			6.1.5

POSTPROC	2	uimsbf	9.1.1.17
}			
if (EXTENDED_MV == 1) {			6.2.7 (Annex J)
MVRANGE	Variable size	vlclbf	9.1.1.26
}			
if (EXTENDED_DMV == 1) {			6.2.14
DMVRANGE	Variable size	vlclbf	9.1.1.27
}			
MVMODE	Variable size	vlclbf	9.1.1.46
FORWARDMB	Bitplane		9.1.1.53
MBMODETAB	3	uimsbf	9.1.1.33
IMVTAB	3	uimsbf	9.1.1.34
ICBPTAB	3	uimsbf	9.1.1.35
if (MVMODE ==”Mixed-MV”) {			9.1.1.46
4MVBPTAB	2	uimsbf	9.1.1.37
}			
if (DQUANT != 0) {			6.2.8 (Annex J)
VOPDQUANT ( )	Variable size	vlclbf	Table 24, 9.1.1.25
}			
if (VSTRANSFORM == 1) {			6.2.9 (Annex J)
TTMBF	1	uimsbf	9.1.1.38
if (TTMBF == 1) {			
TTFRM	2	uimsbf	9.1.1.39
}			
}			
TRANSACFRM	Variable size	vlclbf	9.1.1.22
TRANSDCTAB	1	uimsbf	9.1.1.24
for (“all macroblocks”) {			‘all macroblocks’ represents all macroblocks in this BDU.
B INTERLACE FIELD MB ( )			**Table 95**
}			

}			

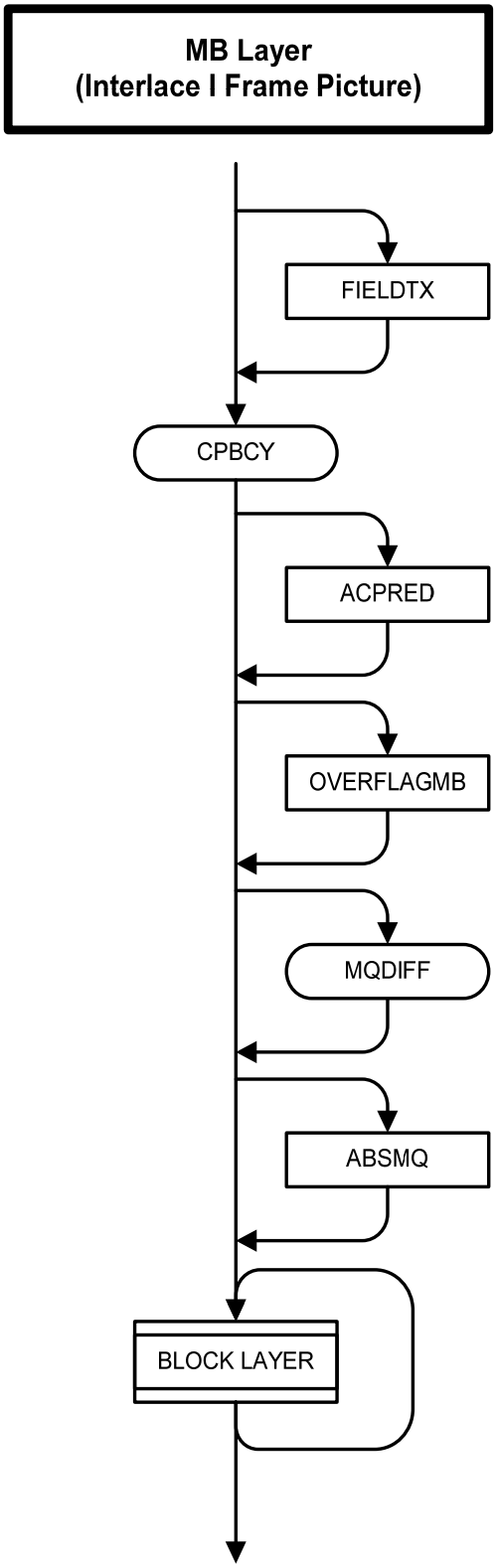


Figure 101: Syntax diagram for macroblock layer bitstream in Interlace Frame I picture

Table 90: Macroblock layer bitstream in Interlaced Frame I Picture

I INTERLACE FRAME MB( ) {	Number of bits	Descriptor	Reference
If (FIELDTX Coding Mode == “Raw”) {			9.1.1.18
FIELDTX	1	uimsbf	9.1.3.1
}			
CBPCY	Variable size	vlclbf	9.1.3.2
If (ACPRED Coding Mode == “Raw”) {			9.1.1.19
ACPRED	1	uimsbf	9.1.3.3
}			
If (CONDOVER == 11b && OVERFLAGS Coding Mode == ”Raw”) {			7.1.1.29, 7.1.1.30, 7.2.2
OVERFLAGMB	1	uimsbf	9.1.3.4
}			
If (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
for (“all blocks in MB”) {			
INTRA BLOCK( )			Table 31
}			
}			

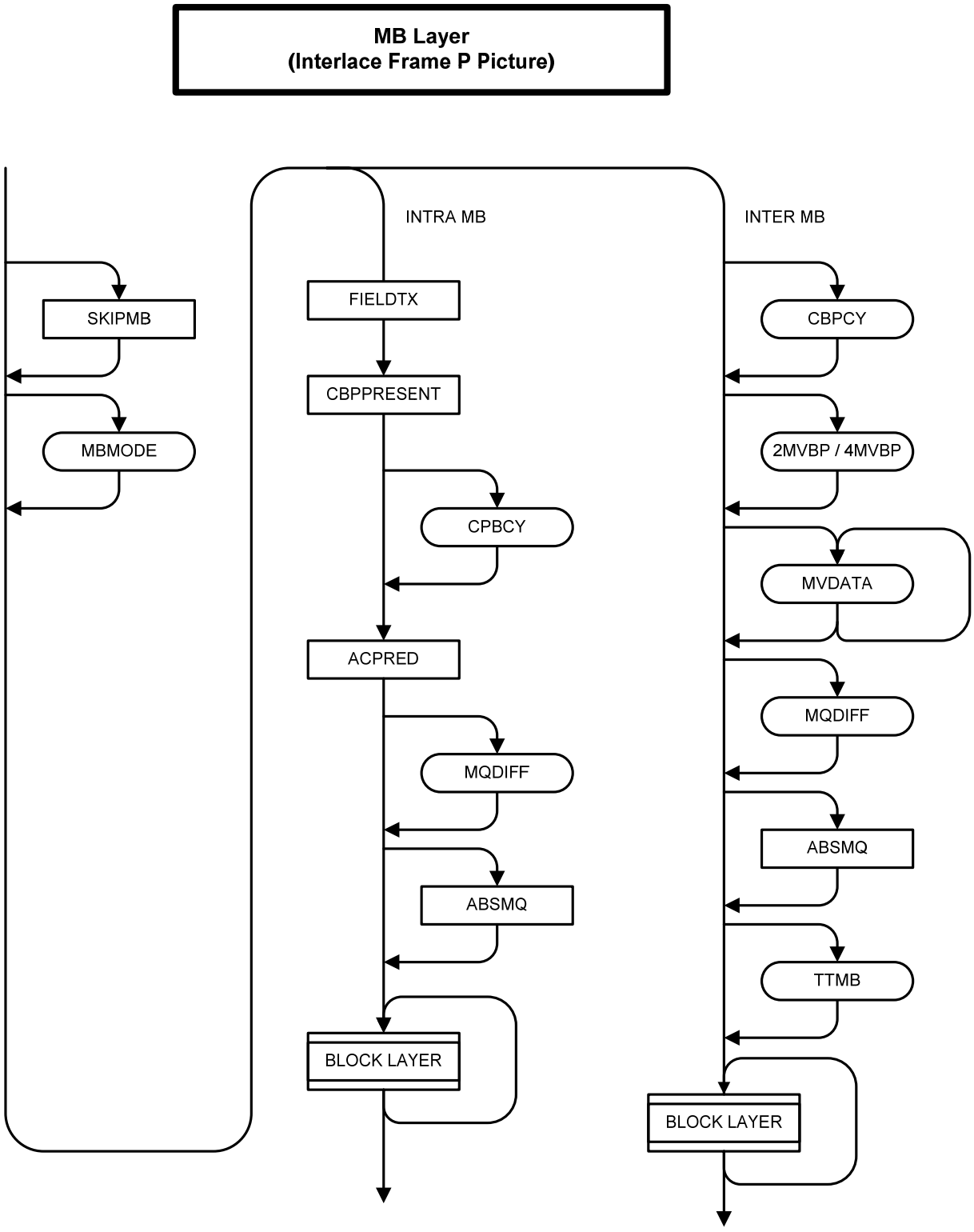


Figure 102: Syntax diagram for macroblock layer bitstream in Interlace Frame P picture

Table 91: Macroblock layer bitstream in Interlaced Frame P Picture

P INTERLACE FRAME MB( ) {	Number of bits	Descriptor	Reference
if (SKIPMB Coding Mode == "Raw") {			9.1.1.32
SKIPMBBIT	1	uimsbf	9.1.3.7
}			
if (!SKIPMBBIT) {			9.1.1.32, 9.1.3.7
MBMODE	Variable size	vlclbf	9.1.3.8
}			
if ("Intra MB") {			Inferred from MBMODE see 9.1.3.8, 10.7.3.4
FIELDTX	1	uimsbf	9.1.3.1
CBPRESENT	1	uimsbf	9.1.3.9
if (CBPPRESENT == 1) {			
CBPCY	Variable size	vlclbf	9.1.3.2
}			
ACPRED	1	uimsbf	9.1.3.3
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == "All Macroblocks") {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
for ("all blocks in MB") {			
INTRA BLOCK( )			Table 31
}			
} /* Intra MB */			
else { /* Inter MB */			
if (!SKIPMBBIT) {			9.1.1.32, 9.1.3.7

if (“CBP is present”) {			Inferred from MBMODE
CBPCY	Variable size	vlclbf	9.1.3.2
}			
if (MVTYPEMB mode == ‘2 Field MV’) {			MVTYPEMB mode is inferred from MBMODE syntax element See 10.7.3.4
2MVBP	Variable size	vlclbf	9.1.3.10
}			
if (MVTYPEMB mode == ‘4-MV’ || MVTYPEMB mode == ‘4 Field MV’) {			MVTYPEMB mode is inferred from MBMODE syntax element See 10.7.3.4
4MVBP	Variable size	vlclbf	9.1.3.11
}			
for (“all motion vectors”) {			Number of motion vectors inferred from 2MVBP or 4MVBP or MVTYPEMB as described in 10.7.3.2 and 10.7.3.4
MVDATA	Variable size	vlclbf	9.1.3.12
}			
if (DQUANTFRM && CBPCY) {			7.1.1.31.1, 9.1.3.2
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
If (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
} /* if (DQPROFILE) */			
} /* if (DQUANTFRM) */			

} /* if (!SKIPMBBIT) */			
if (!TTMBF && CBPCY) {			9.1.1.38, 9.1.3.2
TTMB	Variable size	vlclbf	9.1.3.13
}			
for (“all coded blocks in MB”) {			Number of coded blocks inferred from CBPCY. See 9.1.3.2
INTER BLOCK( )			Table 32
}			
} /* Inter MB */			
}			

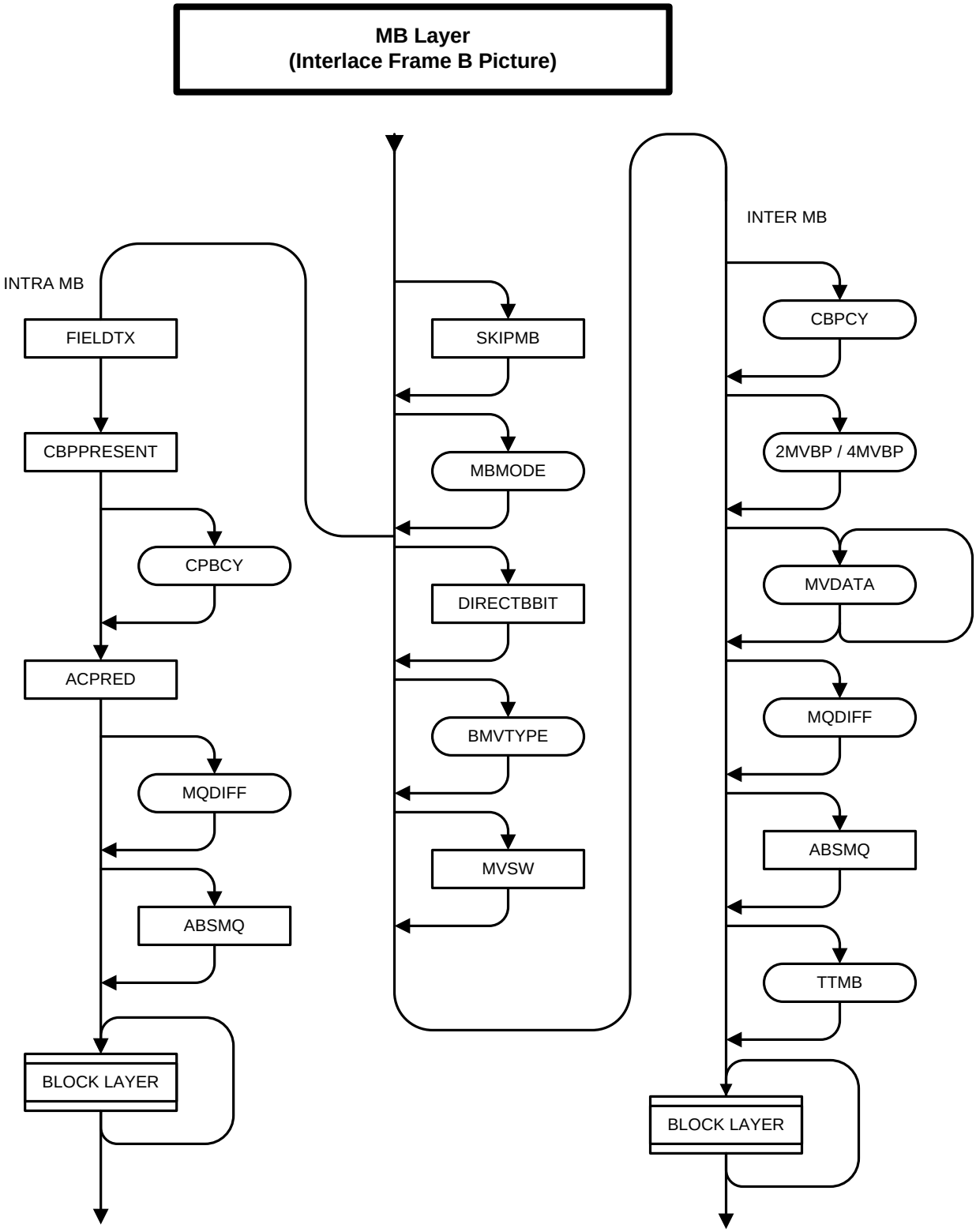


Figure 103: Syntax diagram for macroblock layer bitstream in Interlace Frame B picture

Table 92: Macroblock layer bitstream in Interlaced Frame B Picture

B INTERLACE FRAME MB( ) {	Number of bits	Descriptor	Reference
if (SKIPMB Coding Mode == “Raw”) {			9.1.1.32
SKIPMBBIT	1	uimsbf	9.1.3.7
}			
If (!SKIPMBBIT) {			9.1.1.32, 9.1.3.7
MBMODE	Variable size	vlclbf	9.1.3.8
}			
if (“Intra MB”) {			Inferred from MBMODE see 9.1.3.8, 10.7.3.4
FIELDTX	1	uimsbf	9.1.3.1
CBPPRESENT	1	uimsbf	9.1.3.9
if (CBPPRESENT) {			
CBPCY	Variable size	vlclbf	9.1.3.2
}			
ACPRED	1	uimsbf	9.1.3.3
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
for (“all blocks in MB”) {			
INTRA BLOCK( )			Table 31
}			
} /* Intra MB */			
else { /* Inter MB */			

if (DIRECTMB Coding Mode == “Raw”) {		<	9.1.1.41
DIRECTBBIT	1	uimsbf	9.1.3.14
}			
if (!DIRECTBBIT) {			9.1.1.41, 9.1.3.14
BMVTYPE	Variable size	vlclbf	9.1.3.15
}			
if (‘MVTYPEMB mode == ‘2 Field MV’ && BMVTYPE != Interpolated && !(DIRECTBBIT)) {			MVTYPEMB mode is inferred from MBMODE syntax element. See 10.7.3.4 and9.1.3.16 10.8.6.1
MVSW	1	uimsbf	9.1.3.16, 10.8.6.4
}			
if (SKIPMBBIT) {			
goto End;			
}			
if (“CBP is present”) {			inferred from MBMODE. See 10.7.3.4
CBPCY	Variable size	vlclbf	9.1.3.2
}			
if (!(DIRECTBBIT)) {			
if ((MVTYPEMB mode == “2 Field MV” && BMVTYPE != Interpolated) || (MVTYPEMB mode == “1-MV” && BMVTYPE == Interpolated)) {			MVTYPEMB mode is inferred from MBMODE syntax element. See 10.7.3.4, 9.1.3.10, 10.8.6.1
2MVBP	Variable size	vlclbf	
}			
else if ((“MVTYPEMB mode == 2 Field MV” && MBTYPE == INTERP)) {			MVTYPEMB mode is inferred from MBMODE syntax element See 10.7.3.4, 10.8.6.1
4MVBP	Variable size	vlclbf	9.1.3.11
}			
}			
for (“all motion vectors”) {			Number of MVs inferred from 2MVBP or 4MBVP or MVTYPEMB. See 10.8.6.1.5
MVDATA	Variable size	vlclbf	9.1.3.12

			Note forward MVs are received before backward MVs in the interpolated case
}			
if (DQUANTFRM && CBPCY) {			7.1.1.31.1, 9.1.3.2
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
if (!TTMBF && CBPCY) {			9.1.1.38, 9.1.3.2
TTMB	Variable size	vlclbf	9.1.3.13
}			
for (“all coded blocks in MB”) {			
INTER BLOCK( )			Table 32
}			
} /* Inter MB */			
}			
End:			

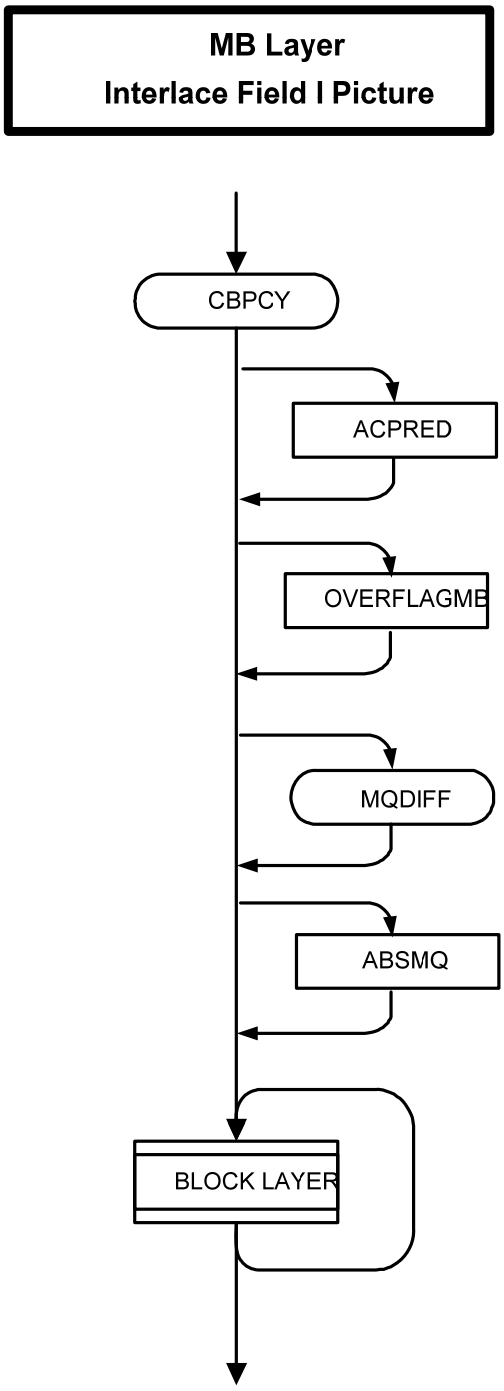


Figure 104: Syntax diagram for macroblock layer bitstream in interlace field I picture

Table 93: Macroblock layer bitstream in Interlaced Field I picture

I INTERLACE FIELD MB( ) {	Number of bits	Descriptor	Reference
CBPCY	Variable size	vlclbf	9.1.3.2
if (ACPRED Coding Mode == "Raw") {			9.1.1.19, See 7.2.2 for

			Raw Mode
ACPRED	1	uimsbf	9.1.3.3
}			
if (CONDOVER == 11b && OVERFLAGS Coding Mode == “Raw”) {			9.1.1.20, 9.1.1.21
OVERFLAGMB	1	uimsbf	9.1.3.4
}			
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
for (“all blocks in MB”) {			
INTRA BLOCK( )			Table 31
}			
}			

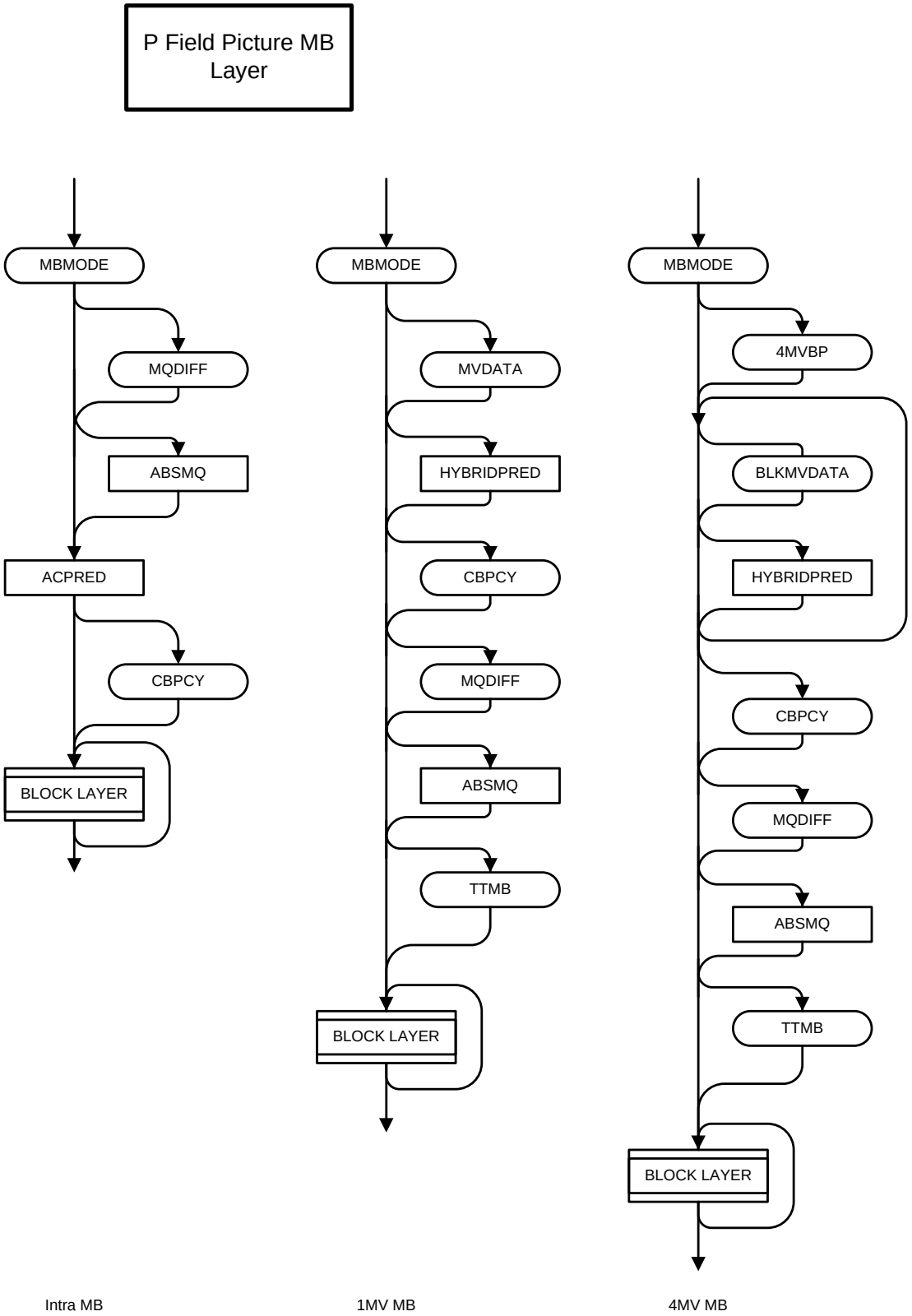


Figure 105: Syntax diagram for macroblock layer bitstream in P field picture

Table 94: Macroblock layer bitstream in Interlaced Field P Picture

P INTERLACE FIELD MB( ) {	Number of bits	Descriptor	Reference
MBMODE	Variable size	vlclbf	9.1.3.8
if (“Intra MB”) {			Inferred from MBMODE see 9.1.3.8, 10.3.5.3
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
ACPRED	1	uimsbf	9.1.3.3
if (“CBP is Present”) {			Inferred from MBMODE, see 10.3.5.3
CBPCY	Variable size	vlclbf	9.1.3.2
}			
for (“all blocks in MB”) {			Inferred from CBPCY
INTRA BLOCK( )			Table 31
}			
} /* Intra MB */			
else { /* Inter MB */			
if (“1-MV MB”) {			Inferred from MBMODE see 10.3.5.3
if (“MV Data is Present”) {			Inferred from MBMODE, see 10.3.5.1.1, 10.3.5.3
MVDATA	Variable size	vlclbf	9.1.3.12
}			

if (“Hybridpred syntax element for MV prediction is Present”) {			Conditions specified in 10.3.5.4.3.5
HYBRIDPRED	1	uimsbf	9.1.3.17
}			
}			
else { /* 4-MV Macroblock */			
4MVBP	Variable Size	vlclbf	9.1.3.11
for (“all Y blocks”) {			
if (“BLKMVDATA is present”) {			Inferred from 4MVBP, see 9.1.3.11, 10.3.5.1.2
BLKMVDATA	Variable Size	vlclbf	9.1.3.18
}			
if (“HYBRIDPRED syntax element for MV prediction is present”) {			Conditions specified in 10.3.5.4.3.5
HYBRIDPRED			9.1.3.17
}			
} /* all Y blocks */			
}			
if (“CBP is Present”) {			Inferred from MBMODE, see 10.3.5.3
CBPCY	Variable size	vlclbf	9.1.3.2
}			
if (DQUANTFRM && CBPCY) {			7.1.1.31.1, 9.1.3.2
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
} /* if (DQPROFILE …*/			
} /* if (DQUANTFRM…*/			
if (!TTMBF && CBPCY) {			9.1.1.38, 9.1.3.2
TTMB	Variable size	vlclbf	9.1.3.13

}			
for (“all coded blocks in MB”) {			Inferred from CBPCY 9.1.3.2, 10.3.5.5
INTER BLOCK( )			Table 32
}			
} /* Inter MB */			
}			

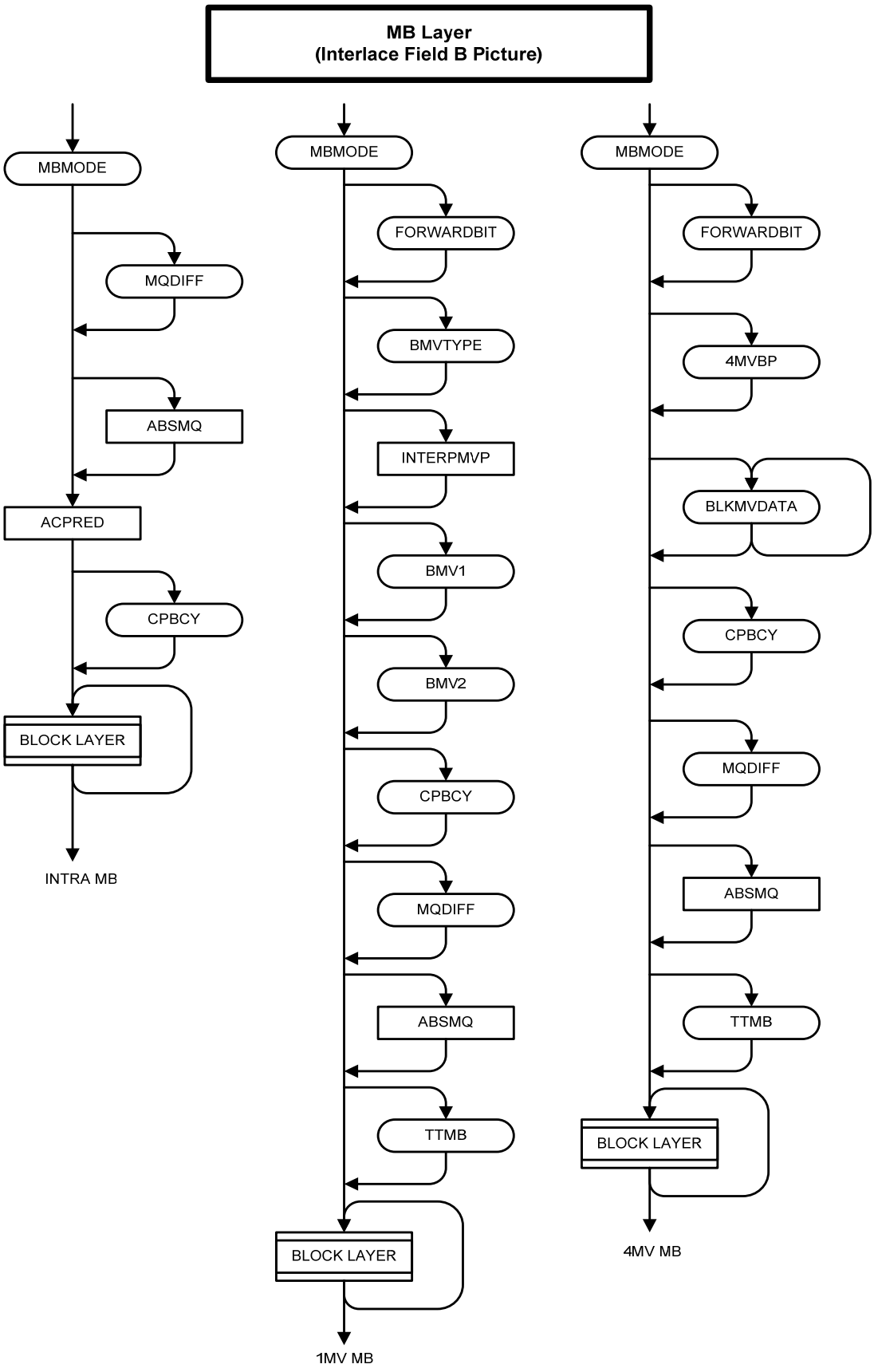


Figure 106: Syntax diagram for macroblock layer bitstream in Field B picture

Table 95: Macroblock layer bitstream in Interlaced Field B Picture

B INTERLACE FIELD MB( ) {	Number of bits	Descriptor	Reference
MBMODE	Variable size	vlclbf	9.1.3.8
if (“Intra MB”) {			Inferred from MBMODE see 9.1.3.8, 10.3.5.3
if (DQUANTFRM) {			7.1.1.31.1
if (DQPROFILE == “all macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
}			
}			
ACPRED	1	uimsbf	9.1.3.3
if (“CBP is present”) {			Inferred from MBMODE see 10.3.5.3
CBPCY	Variable size	vlclbf	9.1.3.2
}			
for (“all blocks in MB”) {			Inferred from CBPCY
INTRA BLOCK( )			Table 31
}			
} /* Intra MB */			
else { /* Inter MB */			
if (FORWARDMB Coding Mode == “Raw”) {			9.1.1.53 See 7.2.2 for Raw Mode
FORWARDBIT	1	uimsbf	9.1.3.19
}			
if (“MB Prediction Mode” != “Forward”) {			Inferred from FORWARDBIT or corresponding bitplane, see 9.1.1.53,

			9.1.3.19
if (“4MVBP is Present”) {			Inferred from MBMODE see 9.1.3.8, 10.3.5.3
BMVTYPE = “Backward”			9.1.3.15 This is inferred because only forward and backward are allowed with 4-MV
}			
else {			
BMVTYPE	Variable Size	vlclbf	9.1.3.15
}			
if (“BMV Type” == “Interpolated”) {			Inferred from FORWARDBIT (9.1.3.19) and BMVTYPE (9.1.3.15)
INTERPMVP	1	uimsbf	9.1.3.20
}			
}			
if (“1-MV MB”) {			Inferred from MBMODE (see 9.1.3.8, 10.3.5.3)
if (“BMVType” != ‘Direct’ && “MVData is Present”) {			For BMVType, see 9.1.3.15, 10.4.6.1. Presence of Block MV Data is Inferred from MBMODE see 10.3.5.3
BMV1	Variable size	vlclbf	9.1.3.21
}			
if (“BMVType” == “Interpolated” && INTERPMVP == 1) {			9.1.3.15, 9.1.3.20
BMV2	Variable size	vlclbf	9.1.3.22 Corresponds to the backward MV
}			
}			
else { /* 4-MV Macroblock */			BMVTYPE can only be forward or backward in 4-MV

			MBs.
4MVBP	Variable Size	vlclbf	9.1.3.11
for (“all Y blocks”) {			
if (“BLKMVDATA is present”) {			Inferred from 4MVBP see 9.1.3.11, 10.3.5.1.2
BLKMVDATA	Variable Size	vlclbf	9.1.3.18
}			
} /* all Y blocks */			
} /* 4-MV Macroblock */			
if (“CBP is Present”) {			Inferred from MBMODE see 10.3.5.3
CBPCY	Variable size	vlclbf	9.1.3.2
}			
if (DQUANTFRM && CBPCY) {			7.1.1.31.1, 9.1.3.2
if (DQPROFILE == “All Macroblocks”) {			7.1.1.31.2
if (DQBILEVEL){			7.1.1.31.5
MQDIFF	1	uimsbf	9.1.3.5
} else {			
MQDIFF	3	uimsbf	9.1.3.5
if (MQDIFF == 7) {			
ABSMQ	5	uimsbf	9.1.3.6
}			
}			
} /* if (DQPROFILE)*/			
} /* if (DQUANTFRM)*/			
if (!TTMBF && CBPCY) {			9.1.1.38, 9.1.3.2
TTMB	Variable size	vlclbf	9.1.3.13
}			
for (“all coded blocks in MB”) {			Inferred from CBPCY 9.1.3.2, 10.3.5.5
INTER BLOCK( )			Table 32
}			
} /* Inter MB */			
}			

9.1.1 Picture layer

Data for each picture shall consist of a picture header followed by data for the macroblock layer. The bitstream elements that make up the interlace frame picture headers for I, P and B picture types are shown in Figure 94, Figure 95 and Figure 96 respectively. The bitstream elements that make up the frame header for interlace field pictures are shown in Figure 97. The bitstream elements that make up the interlace field picture headers for I, P and B pictures are shown in Figure 98, Figure 99 and Figure 100 respectively. The following sections describe the bitstream elements in the interlace frame picture and interlace field picture headers.

9.1.1.1 Frame Coding Mode (FCM) (Variable size)

FCM in interlace frame and interlace field picture headers shall be the same as described in section 7.1.1.15.

9.1.1.2 Picture Type (PTYPE) (Variable size)

PTYPE in interlace frame pictures shall be the same as described in section 7.1.1.4 for progressive pictures.

9.1.1.3 Temporal Reference Frame Counter (TFCNTR) (8 bits)

TFCNTR in interlace frame and interlace field picture headers shall be the same as described in section 7.1.1.16.

9.1.1.4 Top Field First (TFF) (1 bit)

TFF in the frame headers of interlace frame and interlace field pictures shall be the same as described in section 7.1.1.17, except the TFF in the frame header of interlace field pictures shall satisfy an additional constraint: If the current frame coded as two interlace field pictures contains at least one P or B field, and if this P or B field uses one or both fields in another frame as a reference where the reference frame was also coded as a interlace field picture, then the TFF of the current frame and reference frame shall be the same.

9.1.1.5 Repeat First Field (RFF) (1 bit)

RFF in the frame headers of interlace frame and interlace field pictures shall be the same as described in section 7.1.1.18.

9.1.1.6 Repeat Frame Count (RPTFRM) (2 bits)

RPTFRM in the frame headers of interlace frame and interlace field pictures shall be the same as described in section 7.1.1.19.

9.1.1.7 Pan Scan Present Flag (PS_PRESENT) (1 bit)

The PS_PRESENT syntax element shall be the same as described in section 7.1.1.20 for progressive advanced profile pictures.

9.1.1.8 Pan Scan Window Horizontal Offset (PS_HOFFSET) (18 bits)

The PS_HOFFSET syntax element shall be the same as described in section 7.1.1.21 for progressive advanced profile pictures.

9.1.1.9 Pan Scan Window Vertical Offset (PS_VOFFSET) (18 bits)

The PS_VOFFSET syntax element shall be the same as described in section 7.1.1.22 for progressive advanced profile pictures.

9.1.1.10 Pan Scan Window Width (PS_WIDTH) (14 bits)

The PS_WIDTH syntax element shall be the same as described in section 7.1.1.23 for progressive advanced profile pictures.

9.1.1.11 Pan Scan Window Height (PS_HEIGHT) (14 bits)

The PS_HEIGHT syntax element shall be the same as described in section 7.1.1.24 for progressive advanced profile pictures.

9.1.1.12 Rounding Control Bit (RNDCTRL)(1 bit)

RNDCTRL is a 1 bit syntax element that is present in interlace frame and interlace field picture headers (I, P, B). The flag shall be used to indicate the type of rounding used for the current frame. If RNDCTRL == 1, the parameter *RND* which controls rounding shall be set to 1. Otherwise, *RND* shall be set to zero. In interlace frame I and BI pictures, RNDCTRL shall have the value 0

9.1.1.13 UV Sampling Format (UVSAMP)(1 bit)

UVSAMP in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.26.

9.1.1.14 Picture Quantizer Index (PQINDEX) (5 bits)

PQINDEX in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.6.

9.1.1.15 Half QP Step (HALFQP) (1 bit)

HALFQP in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.7.

9.1.1.16 Picture Quantizer Type (PQUANTIZER) (1 bit)

PQUANTIZER in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.8.

9.1.1.17 Post Processing (POSTPROC)(2 bits)

POSTPROC is a 2 bits syntax element that shall be present in all pictures for the advanced profile when the sequence level flag POSTPROCFLAG is set to 1. It shall be the same as defined in section 7.1.1.27 for progressive pictures.

9.1.1.18 Field Transform Flag (FIELDTX)(Variable Size)[I]

FIELDTX is a bitplane coded syntax element that shall be present in Interlace frame I picture headers, and defines whether a macroblock is frame or field coded (and thus the internal organization of the macroblock). FIELDTX == 1 shall indicate that the macroblock is field coded. Otherwise, the macroblock shall be frame coded. FIELDTX may also specify that the frame/field coded information is coded in the raw mode, in which case this information is signaled at the macroblock level (see section 9.1.3.1). Refer to section 7.2 for a description of the bitplane coding. See section 10.5.1 for more details on the use of FIELDTX.

9.1.1.19 AC Prediction (ACPRED)(Variable size)

The ACPRED syntax element shall only be present in interlace field I and interlace frame I pictures and it shall be the same as described in section 7.1.1.28.

9.1.1.20 Conditional Overlap Flag (CONDOVER) (Variable size)

CONDOVER shall be present only in frame / field I pictures and it shall be the same as described in section 7.1.1.29.

9.1.1.21 Conditional Overlap Macroblock Pattern Flags (OVERFLAGS)(Variable size)

OVERFLAGS shall be present only in frame / field I pictures and it shall be the same as described in section 7.1.1.30.

9.1.1.22 Frame-level Transform AC Coding Set Index (TRANSACFRM)(Variable size)

TRANSACFRM in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.11.

9.1.1.23 Frame-level Transform AC Table-2 Index (TRANSACFRM2)(Variable size)

TRANSACFRM2 in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.12.

9.1.1.24 Intra Transform DC Table (TRANSDCTAB)(1 bit)

TRANSDCTAB in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.13.

9.1.1.25 Macroblock Quantization (VOPDQUANT) (Variable size)

VOPDQUANT in interlace frame and interlace field pictures shall be the same as described in section 7.1.1.31.

9.1.1.26 Extended MV Range Flag (MVRANGE) (Variable size)

MVRANGE is a variable-sized syntax element and shall be present only in field / frame P and B picture headers. For interlace field pictures in which MVMODE is Mixed-MV or 1-MV (not half-pel), and for interlace frame pictures, MVRANGE shall be the same as defined in section 7.1.1.9.

For interlace P and B field pictures in which MVMODE is 1-MV Half-pel bicubic or 1-MV Half-pel bilinear, the MVRANGE shall be as defined in Table 96. The value of f represents the fractional portion of the MV range as a mantissa in Table 96. For half-pel mode f shall be only 0 or 5.

For interlace frame B pictures, the value of the MVRANGE syntax element shall be greater or equal to the value of the MVRANGE syntax element of the subsequent (backward reference) anchor P picture. The MVRANGE syntax element of an interlace frame B picture may take any value if the backward reference picture is an I picture or a skipped picture. The value of the MV Range of an interlace B field picture shall be greater or equal to the value of the MV Range of the backward reference field of the same polarity, if the backward reference field is coded as a interlace P Field picture. The MVRANGE syntax element of an interlace B field picture may take any value if the backward reference field of the same polarity is coded as an interlace I field picture.

Table 96: MVRANGE – Motion Vector Range for Interlace Field Pictures Using Half-pel Modes

MVRANGE VLC	MV range in full pixel units (horizontal x vertical)
0b (also default)	[-128, 127.f] x [-64, 63.f]
10b	[-256, 255.f] x [-128, 127.f]
110b	[-1024, 1023.f] x [-256, 255.f]
111b	[-2048,2047.f] x [-512, 511.f]

9.1.1.27 Extended Differential MV Range Flag (DMVRANGE) (Variable size)

DMVRANGE is a variable sized syntax element and shall be present in field / frame P and B pictures if the syntax element EXTENDED_DMV == 1. The DMVRANGE element shall be coded as defined in Table 97. See section 10.3.5.4.2 and it sub-sections for a description of how the DMVRANGE value is used.

Table 97: DMVRANGE VLC Table

Extended Horizontal Differential MV Range	Extended Vertical Differential MV Range	DMVRANGE VLC	DMVRANGE VLC Size
No	No	0b	1
Yes	No	10b	2
No	Yes	110b	3
Yes	Yes	111b	3

9.1.1.28 4 Motion Vector Switch (4MVSWITCH) (1 bit)

The 4MVSWITCH syntax element is a 1-bit field and shall be present in the picture header of interlace frame P pictures. If 4MVSWITCH == 0, the macroblocks in the picture shall have only one motion vector or two motion vectors, depending on whether the macroblock has been frame-coded or field-coded respectively. If 4MVSWITCH == 1, there shall be either 1, 2 or 4 motion vectors per macroblock.

For interlace frame B pictures, 4MVSWITCH shall not be present, and the value of this syntax element shall be set to zero.

Note: There can be 1, 2, or 4 motion vectors per macroblock for interlace frame B pictures. See section 10.7.3 for more details on the use of 4MVSWITCH in decoding.

9.1.1.29 Intensity Compensation (INTCOMP)(1 bit)

INTCOMP is a 1 bit syntax element that shall be present in interlace frame P and B picture headers. INTCOMP is used to indicate whether intensity compensation mode is used in the current frame. If INTCOMP == 1, intensity compensation shall be used. Otherwise, intensity compensation shall not be used. In interlace frame B pictures the value of this bit shall always be set to 0 as this syntax element is not used in B pictures.

9.1.1.30 Luma Scale (LUMSCALE)(6 bits)

The LUMSCALE syntax element shall be present in P interlace frame pictures if the frame header syntax element INTCOMP == 1. LUMSCALE shall be set according to section 8.3.8.

9.1.1.31 Luma Shift (LUMSHIFT)(6 bits)

The LUMSHIFT syntax element shall be present in P interlace frame pictures if the frame header syntax element INTCOMP == 1. LUMSHIFT shall be set according to section 8.3.8.

9.1.1.32 Skipped Macroblock Decoding (SKIPMB)(Variable size)

The SKIPMB syntax element shall only be present in interlace frame P and interlace frame B pictures and shall be the same as described in 7.1.1.37.

9.1.1.33 Macroblock Mode Table (MBMODETAB) (2 or 3 bits)

The MBMODETAB syntax element is a fixed length field that shall be present in interlace frame P, frame B, field P and field B pictures.

For field P and field B pictures, MBMODETAB shall be a 3 bit value that indicates which one of the eight code tables is used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer and shall be as defined in Table 98. There are two sets of eight code tables and the set that is being used shall also depend on whether Mixed-MV is used or not as indicated by the MVMODE flag. These code tables are defined in section 11.4.1 and 11.4.2.

Table 98: **MBMODETAB code-table for interlace field P, B pictures**

MBMODETAB	Macroblock Mode Code Table
000b	Code Table 0 (Table 144 if Mixed-MV; Table 152 otherwise)
001b	Code Table 1(Table 145 if Mixed-MV; Table 153 otherwise)
010b	Code Table 2(Table 146 if Mixed-MV; Table 154 otherwise)
011b	Code Table 3(Table 147 if Mixed-MV; Table 155 if 1-MV)
100b	Code Table 4(Table 148 if Mixed-MV; Table 156 otherwise)
101b	Code Table 5(Table 149 if Mixed-MV; Table 157 otherwise)
110b	Code Table 6 (Table 150 if Mixed-MV; Table 158otherwise)
111b	Code Table 7 (Table 151 if Mixed-MV; Table 159 otherwise)

For frame P and frame B pictures, MBMODETAB shall be a 2 bit value that indicates which one of the four tables in used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer and shall be as defined in

Table 99. There are two sets of four code tables and the set that is being used shall also depend on whether 4-MV is used or not as indicated by the 4MVSWITCH flag. These tables are defined in section 11.4.3 and 11.4.4

Table 99: **MBMODETAB code-table for interlace frame P, B pictures**

MBMODETAB	Macroblock Mode Code Table
00b	Code Table 0 (Table 160 if 4-MV; Table 164 otherwise)
01b	Code Table 1 (Table 161 if 4-MV; Table 165 otherwise)
10b	Code Table 2 (Table 162 if 4-MV; Table 166 otherwise)
11b	Code Table 3 (Table 163 if 4-MV; Table 167 otherwise)

9.1.1.34 Interlace Motion Vector Table (IMVTAB) (2 or 3 bits)

The IMVTAB syntax element is a 2 or 3 bit value present in interlace field/frame P and B pictures. For P and B interlace frame pictures IMVTAB shall be a 2 bit syntax element that indicates which of the four progressive (also called one-reference) MV tables is used to code the MVDATA syntax element in the macroblock layer, and shall be as defined in Table 100. For B interlace field pictures, IMVTAB shall be a 3 bit syntax element that indicates which of eight interlace code tables are used to decode the motion vector data, and shall be as defined in Table 101. For P interlace field pictures in which NUMREF == 0, IMVTAB shall be a 2 bit syntax element that indicates which of four progressive code tables are used to decode the motion vector data, and shall be as defined in Table 100. For P interlace field pictures in which NUMREF == 1, IMVTAB shall be a 3 bit syntax element that indicates which of eight interlace code tables are used to decode the motion vector data, as defined in Table 101. Refer to section 10.3.5 for a description of the motion vector decoding process.

Table 100: **IMVTAB code-table for P interlace field picture with NUMREF == 0, and for P/B interlace frame pictures**

IMVTAB	Motion Vector Table
00b	1-Reference Table 0 (Table 140)
01b	1-Reference Table 1 (Table 141)
10b	1-Reference Table 2 (Table 142)
11b	1-Reference Table 3 (Table 143)

Table 101: **IMVTAB code-table for P interlace field pictures with NUMREF == 1, and for B interlace field pictures**

IMVTAB	Motion Vector Table
000b	2-Reference Table 0 (Table 132)
001b	2-Reference Table 1 (Table 133)
010b	2-Reference Table 2 (Table 134)
011b	2-Reference Table 3 (Table 135)

100b	2-Reference Table 4 (Table 136)
101b	2-Reference Table 5 (Table 137)
110b	2-Reference Table 6 (Table 138)
111b	2-Reference Table 7 (Table 139)

The motion vector tables are defined in section 11.3.

9.1.1.35 Interlace Coded Block Pattern Table (ICBPTAB) (3 bits)

The ICBPTAB syntax element is a 3 bit value present in interlace field P, B pictures and interlace frame P, B pictures. This syntax element shall signal which of eight code tables is used to decode the CBPCY syntax element in intra-coded or inter-coded macroblocks, and shall be as defined in Table 102. The code tables used for decoding the CBPCY syntax element in interlace frame and interlace field pictures are defined in section 11.2. Section 10.3.5.5 describes the decoding of CBPCY syntax element in interlace P and B picture macroblocks.

Table 102: **ICBPTAB code-table**

ICBPTAB	CBPCY Table
000b	Table 0 (Table 124)
001b	Table 1 (Table 125)
010b	Table 2 (Table 126)
011b	Table 3 (Table 127)
100b	Table 4 (Table 128)
101b	Table 5 (Table 129)
110b	Table 6 (Table 130)
111b	Table 7 (Table 131)

9.1.1.36 2-MV Block Pattern Table (2MVBPTAB) (2 bits)

The 2MVBPTAB syntax element is a 2 bit value and shall be present only in interlace frame P and interlace frame B pictures. This syntax element shall signal which one of four tables is used to decode the 2-MV block pattern (2MVBP) syntax element in 2-MV field macroblocks, and shall be as defined in Table 103. The code tables are defined in section 11.1.2.

Table 103: **2MVBP code-table**

2MVBPTAB	2MVBP Table
00b	Table 0 (Table 120)
01b	Table 1 (Table 121)
10b	Table 2 (Table 122)
11b	Table 3 (Table 123)

9.1.1.37 4-MV Block Pattern Table (4MVBPTAB) (2 bits)

The 4MVBPTAB syntax element is a 2 bit value and shall be present only in interlace frame P, B and interlace field P, B pictures. For interlace field P and B pictures, it shall only be present if MVMODE (or MVMODE2, if MVMODE is set to intensity compensation) indicates that the picture is of 'Mixed-MV' type. For interlace frame P picture, it shall be present only if 4MVSWITCH syntax element is set to 1. For interlace frame B pictures, this syntax element shall always present. The 4MVBPTAB syntax element signals which of four tables is used to decode the 4-MV block pattern (4MVBP) syntax element in 4-MV macroblocks, and shall be as defined in Table 104. The code tables are defined in section 11.1.1.

Table 104: **4MVBP code-table**

4MVBPTAB	4MVBP Table
00b	Table 0 (Table 116)
01b	Table 1 (Table 117)
10b	Table 2 (Table 118)
11b	Table 3 (Table 119)

9.1.1.38 Macroblock-level Transform Type Flag (TTMBF) (1 bit)

This syntax element shall be present only in interlace field P, B pictures and interlace frame P, B pictures. It shall be the same as defined in section 7.1.1.40.

9.1.1.39 Frame-level Transform Type (TTFRM) (2 bits)

This syntax element shall be present only in interlace field P, B pictures and interlace frame P, B pictures. It shall be the same as defined in section 7.1.1.41.

9.1.1.40 B Picture Fraction (BFRACTION)(Variable size)

BFRACTION in interlace frame and interlace field picture headers shall be the same as defined in section 7.1.1.14.

9.1.1.41 B Frame Direct Mode Macroblock Bit Syntax Element (DIRECTMB)(Variable size)

The DIRECTMB is a bitplane coded syntax element that shall be present in interlace frame B pictures. It is used to indicate the macroblocks in the B picture that are coded in direct mode. When bitplane coding is used, each macroblock shall be represented by this syntax element. In particular, even those macroblocks that are coded as intra shall be represented. If DIRECTMB == 1 for a non-intra coded macroblock, then the macroblock shall be coded in direct mode. If DIRECTMB == 0 for a non-intra coded macroblock, then the macroblock shall not be coded in direct mode. The value of DIRECTMB shall be ignored for intra-coded macroblock. The DIRECTMB syntax element may also signal that the direct mode is signaled in raw mode, in which case the direct mode shall be signaled at the macroblock level. In this case, the macroblock level bit shall only be present if the macroblock is not coded as intra. Refer to section 8.7 for a description of the bitplane coding method.

9.1.1.42 Field Picture Type (FPTYPE) (3 bits)

FPTYPE is a 3 bit syntax element present in the picture header of interlace field pictures. FPTYPE shall be decoded according to Table 105.

Table 105: Field Picture Type FLC

FPTYPE	First Field Picture Type	Second Field Picture Type
000b	I	I
001b	I	P

010b	P	I
011b	P	P
100b	B	B
101b	B	BI
110b	BI	B
111b	BI	BI

9.1.1.43 P Reference Distance (REFDIST) (Variable size)

REFDIST is a variable sized syntax element and shall be present in interlace field picture headers, if the entry-level flag REFDIST_FLAG == 1, and if the picture type is not one of the following types: B/B, B/BI, BI/B, BI/BI. If the entry level flag REFDIST_FLAG == 0, REFDIST shall be set to the default value of 0. This element defines the number of frames between the current frame and the reference frame. REFDIST shall be set according to Table 106 as a function of the VLC codewords.

Table 106: REFDIST VLC Table

REFDIST	VLC Codeword (Binary)	VLC Size
0	00b	2
1	01b	2
2	10b	2
N	11[(N-3) 1s]0b	N

The last row in Table 106 indicates the codewords used to represent reference frame distances greater than 2. These are coded as (binary) 11 followed by N-3 1s, where N+1 is the reference frame distance. The last bit in the codeword is 0. For example:

N = 3, VLC Codeword = 110b, VLC Size = 3

N = 4, VLC Codeword = 1110b, VLC Size = 4

N = 5, VLC Codeword = 11110b, VLC Size = 5

The value of REFDIST shall be less than, or equal to, 16.

9.1.1.44 Number of Reference Pictures (NUMREF) (1 bit)

NUMREF is a 1 bit syntax element and shall be present only in interlace P field pictures headers. If NUMREF == 0, then the current interlace P field picture shall reference one field. If NUMREF == 1, then the current interlace P field picture shall reference the two temporally closest (in display order) I or P field pictures. Section 10.3.3 defines the use of NUMREF in decoding interlace P field pictures.

9.1.1.45 Reference Field Picture Indicator (REFFIELD) (1 bit)

REFFIELD is a 1 bit syntax element and shall be present in interlace P field picture headers if NUMREF == 0. If REFFIELD == 0, then the temporally closest (in display order) I or P field shall be used as a reference. If REFFIELD == 1, then the second most temporally recent interlace I or P field picture shall be used as reference. Section 10.3.3 defines the use of REFFIELD in decoding interlace P field pictures.

9.1.1.46 Motion Vector Mode (MVMODE) (Variable size)

The MVMODE syntax element is a variable size syntax element that shall be present in interlace field P and B picture headers. For P pictures it shall be the same as the corresponding MVMODE syntax element described for progressive

pictures in section 7.1.1.32. For B pictures, and PQUANT > 12, then MVMODE shall be as defined in Table 107; and for PQUANT <= 12, MVMODE shall be as defined in Table 108.

Note: Intensity compensation is not present in B pictures.

Table 107: B Picture Low rate (PQUANT > 12) MVMODE code table

MVMODE	Mode
1b	1-MV Half-pel bilinear
01b	1-MV
001b	1-MV Half-pel
000b	Mixed-MV

Table 108: B Picture High rate (PQUANT <= 12) MVMODE code table

MVMODE	Mode
1b	1-MV
01b	Mixed-MV
001b	1-MV Half-pel
000b	1-MV Half-pel bilinear

9.1.1.47 Motion Vector Mode 2(MVMODE2) (Variable size)

The MVMODE2 syntax element shall be present in interlace field P picture headers only if MVMODE signals intensity compensation. The syntax shall be identical to the corresponding MVMODE2 syntax element for progressive pictures as defined in section 7.1.1.33.

9.1.1.48 Intensity Compensation Field (INTCOMPFIELD)(Variable size)

INTCOMPFIELD is a variable-sized syntax element that shall be present in interlace field P field picture headers. INTCOMPFIELD shall be used to indicate which reference field undergoes intensity compensation and shall be as defined in Table 109. INTCOMPFIELD shall be present in the bitstream even if NUMREF == 0 (9.1.1.44).

Table 109: INTCOMPFIELD VLC Table

INTCOMPFIELD	Intensity Compensation Applied to:
1b	Both fields
00b	Top field
01b	Bottom field

9.1.1.49 Field Picture Luma Scale 1 (LUMSCALE1)(6 bits)

The LUMSCALE1 syntax element shall be present in the P interlace field picture header if the field picture header syntax element MVMODE (9.1.1.46) signals intensity compensation. If the INTCOMPFIELD element is 1 or 00b then LUMSCALE1 shall be applied to the top field. Otherwise it shall be applied to the bottom field. The effect of LUMSCALE1 on intensity compensation in P interlace field picture headers is defined in section 10.3.8.

9.1.1.50 Field Picture Luma Shift 1 (LUMSHIFT1)(6 bits)

The LUMSHIFT1 syntax element shall be present in the P interlace field picture header if the interlace field picture header syntax element MVMODE (9.1.1.46) signals intensity compensation. If the INTCOMPFIELD element is 1 or 00b then LUMSHIFT1 shall be applied to the top field. Otherwise it shall be applied to the bottom field. The effect of LUMSHIFT1 on intensity compensation in P interlace field pictures is defined in section 10.3.8.

9.1.1.51 Field Picture Luma Scale 2 (LUMSCALE2)(6 bits)

The LUMSCALE2 syntax element shall be present in the P interlace field picture header if the interlace field picture header syntax element MVMODE (9.1.1.46) signals intensity compensation and INTCOMPFIELD == 1. LUMSCALE2 shall be applied to the bottom field. The effect of LUMSCALE2 in P interlace field pictures is defined in section 10.3.8.

9.1.1.52 Field Picture Luma Shift 2 (LUMSHIFT2)(6 bits)

The LUMSHIFT2 syntax element shall be present in the P interlace field picture header if the interlace field picture header syntax element MVMODE (9.1.1.46) signals intensity compensation and INTCOMPFIELD == 1. LUMSHIFT2 shall be applied to the bottom field. The effect of LUMSHIFT2 on intensity compensation in P interlace field pictures is defined in section 10.3.8.

9.1.1.53 B Field Forward Mode Macroblock Bit Syntax Element (FORWARDMB)(Variable size)

The FORWARDMB syntax element shall only be present in interlace field B pictures. The FORWARDMB syntax element uses bitplane coding to indicate the macroblocks in the B field picture that are coded in forward mode. When sent as a bitplane, there is a bit corresponding to each macroblock, even those that are coded as intra. If FORWARDMB == 1, the macroblock is coded in forward mode. If FORWARDMB == 0, the macroblock coding mode shall be derived as described in section 10.4.5.3. FORWARDMB syntax element may also signal that the forward mode is signaled in raw mode in which case the forward mode is signaled at the macroblock level. In this case, the macroblock level bit shall only be present if the macroblock is not coded as intra. Refer to section 8.7 for a description of the bitplane coding method.

9.1.2 Slice Layer

The slice-layer may be present in the bitstream of interlaced coded pictures in Advanced Profile. Refer to Section 7.1.2 for a definition of the Slice layer. For interlaced frame and interlace field pictures, the syntax elements of the slice layer shall be identical to those of progressive pictures.

As in progressive pictures,

1. motion vector predictors, predictors for AC and DC coefficients, and the predictors for quantization parameters shall be reset at the beginning of a new slice, and
2. no in-loop-deblocking and no overlap smoothing shall be allowed across slices.

In other words, the top and bottom macroblock rows of each slice are treated as if they are the top and macroblocks rows of the picture.

For interlace field pictures, two points should be emphasized.

1. If the PIC_HEADER_FLAG == 1 in the slice layer of an interlace field picture, the picture header information that is repeated shall consist of both the frame picture header, and the field picture header of that field.
2. While coding the SLICE_ADDR syntax element, the row address shall not be reset to zero at the beginning of the second field. In other words, the row address of the first macroblock row in the second field is not zero, but is the row address of the last macroblock row in the first field incremented by one.

9.1.3 Macroblock Layer

Data for each macroblock shall consist of a macroblock header followed by the block layer. Figure 101 to Figure 106 define the macroblock layer structure for interlace field I, P, B pictures and interlace frame I, P, B pictures. The elements that make up the macroblock layer are described in the following sections. The picture types in which the following macroblock layer syntax elements occur in are indicated in the square brackets.

9.1.3.1 Field Transform Flag (FIELDTX)(1 bit)[I, P,B]

FIELDTX is a 1-bit syntax element and shall be present in those P and B interlace frame macroblocks which are intra-coded. FIELDTX shall also be present in I interlace frame macroblocks if the raw mode is used to encode the bitplane information. This syntax element indicates whether a macroblock is frame or field coded (basically, the internal organization of the macroblock). FIELDTX == 1 shall indicate that the macroblock is field coded, and FIELDTX == 0 shall indicate that the macroblock is frame coded. See section 10.5.1 for more details on the use of FIELDTX.

Note: In the inter-coded macroblocks of P and B interlace frame, this syntax element is not explicitly signaled, but inferred from MBMODE (see section 10.7.3.4).

9.1.3.2 Coded Block Pattern (CBPCY) (Variable size)[I, P,B]

CBPCY is a variable-length syntax element present in I picture, P picture and B picture macroblock layers. CBPCY shall be coded as defined in section 8.1.2.1 in I picture macroblocks, and shall be coded as defined in section 10.3.5.5 in P picture and B picture macroblocks. In P and B interlace frame picture macroblocks that are intra-coded, CBPCY shall be present only if indicated by the CBPPRESENT syntax element as defined in section 9.1.3.9.

9.1.3.3 AC Prediction Flag (ACPRED)(1 bit)[I, P,B]

The ACPRED syntax element shall be present in all I interlace frame and interlace field picture macroblocks and in all intra macroblocks in field and frame P and B pictures. Its syntax shall be identical to the corresponding ACPRED syntax for progressive pictures shall be as defined in section 7.1.3.2, for I pictures as defined in section 8.1.2.2, and for P pictures as defined in section 8.3.6.1.

9.1.3.4 Conditional Overlap Macroblock Pattern Flag (OVERFLAGMB) (1 bit) [I]

This syntax element shall be present only in I pictures and its syntax shall be identical to the corresponding OVERFLAGMB syntax for progressive pictures as defined in section 7.1.3.3. See Section 8.5.2 for a description.

9.1.3.5 Macroblock Quantizer Differential (MQDIFF)(Variable size)[I,P,B]

MQDIFF shall be present in interlace field I, P, B pictures and interlace frame I, P, B pictures and it shall be the same as defined in section 7.1.3.4.

9.1.3.6 Absolute Macroblock Quantizer Scale (ABSMQ)(5 bits)[I,P,B]

ABSMQ shall be present in interlace field I, P, B pictures and interlace frame I, P, B pictures and it shall be the same as defined in section 7.1.3.5.

9.1.3.7 Skip MB Bit (SKIPMBBIT)(1 bit)[P,B]

SKIPMBBIT is a 1-bit syntax element that shall be present in P and B interlace frame macroblocks, and its syntax shall be identical to the corresponding SKIPMB syntax for progressive pictures defined in section 7.1.3.7.

9.1.3.8 Macroblock Mode (MBMODE)(Variable size)[P,B]

MBMODE is a variable-length syntax element that shall be present in interlace field P, B and interlace frame P, B macroblocks. It shall be as defined in section 10.3.5.3 for interlace field P, B pictures and in section 10.7.3.4 for interlace frame P, B pictures.

9.1.3.9 CBP Present Flag (CBPPRESENT)(1 bit)[P,B]

CBPPRESENT is a 1-bit syntax element that shall be present in those P and B interlace frame macroblocks which are intra-coded. If CBPPRESENT == 1, the CBPCY syntax element shall be present for that macroblock, and decoded. If CBPPRESENT == 0, the CBPCY syntax element shall not be present, and shall be set to zero.

Note: In P and B interlace field macroblocks, the presence of CBPCY syntax element is not explicitly signaled, but inferred from MBMODE (See 10.3.5.3).

9.1.3.10 Two Motion Vector Block Pattern (2MVBP)(Variable size)[P,B]

2MVBP is a variable sized syntax element present in interlace frame P, B picture macroblocks. In interlace frame P macroblocks, this syntax element shall be present if the MBMODE syntax element indicates that the macroblock has 2

field motion vectors. In this case, 2MVBP indicates which of the 2 luma blocks contain non-zero motion vector differentials. In interlace frame B macroblocks, this syntax element shall be present if the MBMODE syntax element indicates that the macroblock contains 1 motion vector, and if the macroblock is an interpolated macroblock. In this case, 2MVBP shall indicate which of the two motion vectors (forward and backward motion vectors) are present. 2MVBP shall be decoded according to the table specified by the 2MVBPTAB syntax element as defined in section 9.1.1.36. The tables used for decoding 2MVBP are listed in Table 120 to Table 123.

9.1.3.11 Four Motion Vector Block Pattern (4MVBP)(Variable size)[P,B]

4MVBP is a variable sized syntax element present in interlace field P, B and interlace frame P, B picture macroblocks. In interlace field P, B and interlace frame P macroblocks, this syntax element shall be present if the MBMODE syntax element indicates that the macroblock has 4 motion vectors. In this case, 4MVBP indicates which of the 4 luma blocks contain non-zero motion vector differentials. In interlace frame B macroblocks, this syntax element shall be present if the MBMODE syntax element indicates that the macroblock contains 2 field motion vectors, and if the macroblock is an interpolated macroblock. In this case, 4MVBP indicates which of the four motion vectors (the top and bottom field forward motion vectors, and the top and bottom field backward motion vectors) are present. 4MVBP shall be decoded according to the table specified by the 4MVBPTAB syntax element as defined in section 9.1.1.37. The tables used for decoding 4MVBP are listed in Table 116 to Table 119.

9.1.3.12 Motion Vector Data (MVDATA)(Variable size)[P,B]

MVDATA is a variable sized syntax element and shall be present in interlace field P, and interlace frame P, B picture macroblocks. This syntax element shall decode to the motion vector(s) for the macroblock according to section 10.3.5.4 for interlace field pictures, and section 10.7.3.6 for interlace frame pictures.

9.1.3.13 MB-level Transform Type (TTMB)(Variable size)[P,B]

TTMB shall be present in interlace field P, B pictures and interlace frame P, B pictures and it shall be the same as defined in section 7.1.3.10.

9.1.3.14 Direct B Frame Coding Mode (DIRECTBBIT)(1 bit)[B]

DIRECTBBIT is a 1-bit syntax element that shall be present only in interlace frame B picture macroblocks, and its syntax shall be identical to the corresponding syntax element in progressive pictures defined in section 7.1.3.12.

9.1.3.15 B Macroblock Motion Prediction Type (BMVTYPE)(Variable size)[B]

BMVTYPE is a variable sized syntax element and shall be present in interlace frame and field B picture macroblocks. For interlace frame B pictures, the BMVTYPE syntax shall be identical to progressive B pictures defined in section 7.1.3.14.

In interlace field B pictures, BMVTYPE shall be sent if the MB mode is not forward (as indicated by the FORWARDMB or FORWARDBIT syntax element) and 4-MV is not being used. In this case, BMVTYPE shall be as defined in Table 110 to signal whether the B MB is backward, direct or interpolated. In the case where the MB mode is not forward and 4-MV is in use, the BMVTYPE is inferred to be backward because only forward and backward modes are allowed with 4-MV.

Table 110: BMVTYPE VLC Table for Interlace Field B Macroblock not encoded in Forward Mode

BMVTYPE	MB Mode is:
0b	Backward
10b	Direct
11b	Interpolated

9.1.3.16 B Frame MV Switch (MVSW)(1 bit)[B]

MVSW is a 1-bit syntax element and shall be present in B frame macroblocks if the MB is in field mode and if the BMVTYPE is forward or backward. If MVSW == 1, then it shall indicate that the MV type and prediction type changes

from forward to backward (or backward to forward) in going from the top to the bottom field. If MVSW == 0, then the prediction type shall not change in going from the top to the bottom field.

9.1.3.17 Hybrid Motion Vector Prediction (HYBRIDPRED)(1 bit)[P,B]

HYBRIDPRED is a 1-bit syntax element per motion vector, and shall be present in interlace field P picture macroblocks. It shall be as defined in section 10.3.5.4.3.5.

9.1.3.18 Block-level Motion Vector Data (BLKMVDATA)(Variable size)[inter]

BLKMVDATA is a variable-sized syntax element that may be present in interlace field P and B pictures only if the macroblock is coded in 4-MV mode. It shall decode to the motion vector information for the block as defined in section 10.3.5.4.

9.1.3.19 Forward B Field Coding Mode (FORWARDBIT)(1 bit)[B]

FORWARDBIT is a 1-bit syntax element that shall be present in interlace B field picture macroblocks if the field level syntax element FORWARDMB indicates that the raw mode is used. If FORWARDBIT == 1, then the macroblock shall be coded using forward mode. If FORWARDBIT == 0, then the macroblock coding mode shall be derived as defined in section 10.4.5.3.

9.1.3.20 Interpolated MV Present (INTERPMVP)(1 bit)[B]

INTERPMVP is a 1-bit syntax element and shall be present in B field macroblocks if the field level syntax element BMVTYPE indicates that the macroblock type is interpolated. If INTERPMVP == 1, then the interpolated MV (BMV2) shall be present. If INTERPMVP == 0, BMV2 shall not be present.

9.1.3.21 B Macroblock Motion Vector 1 (BMV1)(Variable size)[B]

BMV1 is a variable sized syntax element that shall be present in interlace field B picture macroblocks. This syntax element encodes the first motion vector for the macroblock. The decoding procedure for BMV1 shall be identical to the decoding procedure for MVDATA in interlace field P pictures as defined in section 10.3.5.4.1.

9.1.3.22 B Macroblock Motion Vector 2 (BMV2)(Variable size)[B]

BMV2 is a variable sized syntax element that shall be present in interlace field B picture macroblocks if the interpolation mode is used. This syntax element encodes the second motion vector of the macroblock. The decoding procedure for BMV2 shall be identical to the decoding procedure for MVDATA in interlace field P pictures as defined in section 10.3.5.4.1.

9.1.4 Block Layer Syntax Elements

The block layer syntax elements in interlace frame and interlace field pictures shall be identical to the corresponding syntax elements in progressive pictures as described in 7.1.4.

10 Interlace Decoding Process

This section describes the decoding processes required for decoding interlace field I pictures (section 10.1), interlace field BI pictures (section 10.2), interlace field P pictures (section 10.3), interlace field B pictures (section 10.4) interlace frame I pictures (section 10.5), interlace frame BI pictures (section 10.6), interlace frame P pictures (section 10.7), interlace frame B pictures (section 10.8) for advanced profile.

This section also defines the processes that are common to all interlace frame and interlace field coded picture types including the overlapped transform decoding process (section 10.9) and the in-loop deblock filtering process (section 10.10).

The bitplane decoding process and pan-scan decoding process for all interlaced coded pictures is identical to the corresponding processes for progressive pictures that are defined in section 8.7 and section 8.9 respectively.

Unless described below, the decoding processes of interlace field and interlace frame pictures shall be identical to the corresponding processes in progressive pictures.

10.1 Interlace Field I Picture Decoding

The following section describes the process for decoding field I pictures.

10.1.1 Macroblock Layer Decode

Figure 104 shows the elements that make up the I picture macroblock layer. **Figure 8** shows how the frame is composed of macroblocks.

10.1.1.1 Coded Block Pattern (CBPCY)

The coded block pattern shall be the same as advanced profile progressive I pictures as defined in section 8.1.2.1.

10.1.1.2 AC Prediction Flag (ACPRED)

The ACPRED syntax element in the macroblock header is a one-bit syntax element that defines whether AC prediction is used to decode the AC coefficients for all the blocks in the macroblock. Section 8.1.3.7 defines the AC prediction process. If ACPRED == 1, then AC prediction shall be used, otherwise it shall not be used.

10.1.2 Block Layer Decode

The 4 blocks that make up the Y component of the macroblock shall be decoded first followed by the C_b and C_r blocks as shown in Figure 8. Figure 36 shows the process used to reconstruct the 8x8 blocks.

The following sub-sections define the processes for reconstructing intra blocks in interlace field pictures.

10.1.2.1 DC Differential Bitstream Decode

The DC differential decoding process shall be the same as defined in section 8.1.3.1.

10.1.2.2 DC Predictor

The DC Predictor shall be as defined in section 8.1.3.2.

10.1.2.3 DC Inverse-quantization

The DC inverse-quantization process shall be as defined in section 8.1.3.3.

10.1.2.4 AC Coefficient Bitstream Decode

The AC Coefficient decoding process shall be as defined in section 8.1.3.4.

10.1.2.5 Zigzag Scan of AC Coefficients

The zigzag scanning of AC coefficient shall be the same as defined in section 8.1.3.6.

10.1.2.6 AC Prediction

The AC prediction process shall be the same as defined in section 8.1.3.7.

10.1.2.7 Inverse AC Coefficient Quantization

The inverse AC coefficient quantization process shall be the same as defined in section 8.1.3.8.

10.1.2.8 Coefficient Scaling

For DC and AC prediction, the coefficients in the predicted blocks shall be scaled if the macroblocks quantizers differ from those of the current block as defined in section 8.1.3.9. As in progressive pictures, the product DC_p * $DCSTEP_p$, and the product $\overline{AC}_p$ * $STEP_p$ product shall not exceed the signed 12 bit range, i.e., these product values shall be limited to ≥ -2048 && ≤ 2047.

10.1.2.9 Inverse Transform

The inverse transform, overlap smoothing and reconstruction processes shall be identical to the processes defined for advanced profile progressive I frames in section 8.1.3.10.

10.2 Interlace BI Field Decoding

Interlace BI Fields interlace B fields where all macroblocks are intra-coded. The syntax and decoding processes of Interlace BI field shall be identical to that of Interlace I field, but an Interlace BI Field shall not be used as an anchor or reference to predict other pictures.

10.3 Interlace Field P Picture Decoding

Figure 59 shows the steps required to decode and reconstruct blocks in interlace field P pictures. The following sub-sections define the processes for decoding interlace field P pictures.

10.3.1 Handling of Top-Field First (TFF)

In Interlace Field Pictures the TFF syntax element in the frame header shall indicate the temporal order of the two fields that comprise the frame. TFF == 1 shall indicate that the first field in the frame is the top field and the second field is the bottom field. TFF == 0 shall indicate that the first field in the frame is the bottom field and the second field is the top field. The following restrictions and procedures are required for the TFF syntax element and its relationship to the reference field or fields:

1. If the current and reference frame are coded as Interlace Field Pictures (FCM == 11b) then both frames shall have the same TFF value.
2. If the current frame is coded as an Interlace Field Picture and the reference frame is coded as a Progressive Picture or Interlace Frame Picture then for purposes of determining the field order of the reference frame, the TFF value of the reference frame shall be assumed to be the same as the TFF value of the current frame, regardless of the value of TFF for the reference frame.

10.3.2 Out-of-bounds Reference Pixels

When coding the first field in a frame, the top and/or bottom fields from the reference frame may be used as the reference. The out-of-bound pixels for the reference frame shall be generated in a manner which is dependent on the coding type (progressive or interlaced) of the reference frame as shown in Figure 49 and described in section 8.3.2. The boundary of the frame shall be calculated as defined in section 8.3.2.

When coding the second field in a frame, the first field of the current frame may be used as a reference as well as the second field of the reference frame. When the first field of the current frame is used as a reference, out-of-bound pixels shall be generated by replicating the boundary pixels of the first field as shown in Figure 107. The figure shows the

bottom field being coded (denoted by X's) and using the top field as a reference. The figure shows the replication of the top field reference pixels.

When the second field uses the second field of the reference frame as a reference, the pixel replication shall be performed as shown in Figure 49. The out-of-bound pixels for the reference frame shall be generated in a manner which is dependent on the coding type (progressive or interlaced) of the reference frame as shown in Figure 49 and defined in section 8.3.2.

Note: Annex K.2 provides additional information on the internal representation of a frame.

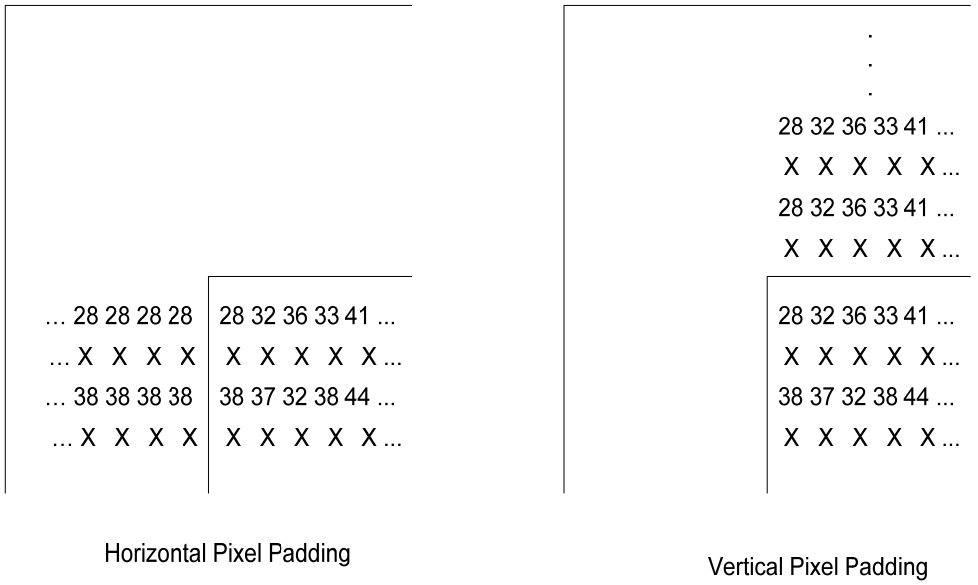


Figure 107: Horizontal and vertical pixel replication for out-of-bounds references in interlace field pictures for the case where the second field uses the first field as reference

10.3.3 Reference Pictures

A P Interlace Field Picture shall reference either one or two previously decoded fields. The NUMREF syntax element in the picture layer is a one bit syntax element that shall indicate whether the current field references one or two previous reference field pictures. If NUMREF == 0, then the current P interlace field picture references one field. In this case, the REFFIELD syntax element shall follow the NUMREF syntax element in the picture layer bitstream. The REFFIELD syntax element is a one bit syntax element that shall indicate which previously decoded field is used as the reference. If REFFIELD == 0, then the temporally closest (in display order) I or P field shall be used as the reference. If REFFIELD == 1, then the second most temporally recent I or P field picture shall be used as the reference. The possible repeat of a field for display (due to RFF == 1) shall be ignored while choosing the closest field for reference.

If NUMREF == 1, then the current P interlace field picture references the two temporally closest (in display order) I or P field pictures.

Figure 109, Figure 110 and Figure 108 show examples of reference field pictures for NUMREF == 0 and NUMREF == 1.

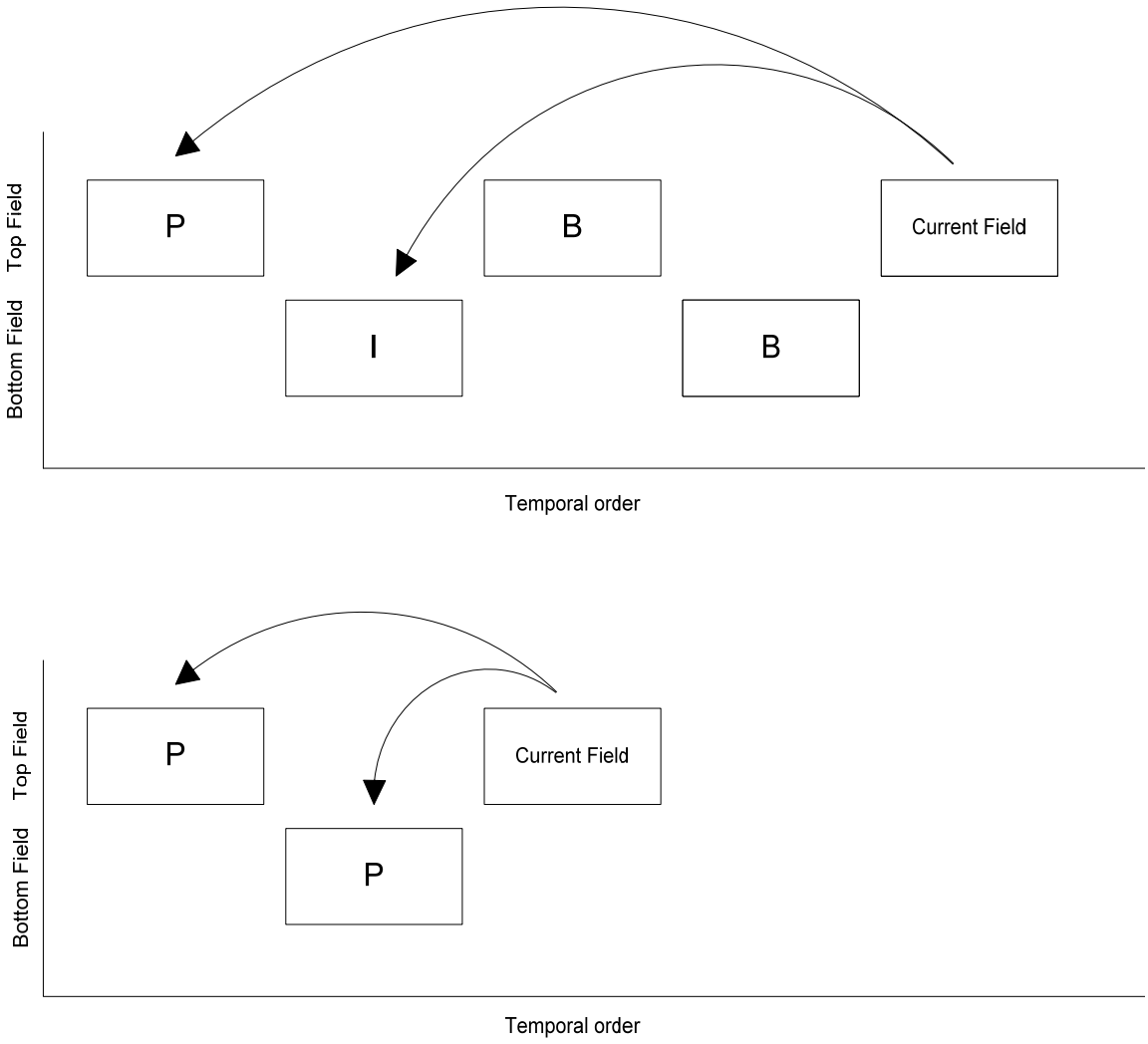


Figure 108: Example of two reference interlace field pictures (NUMREF == 1)

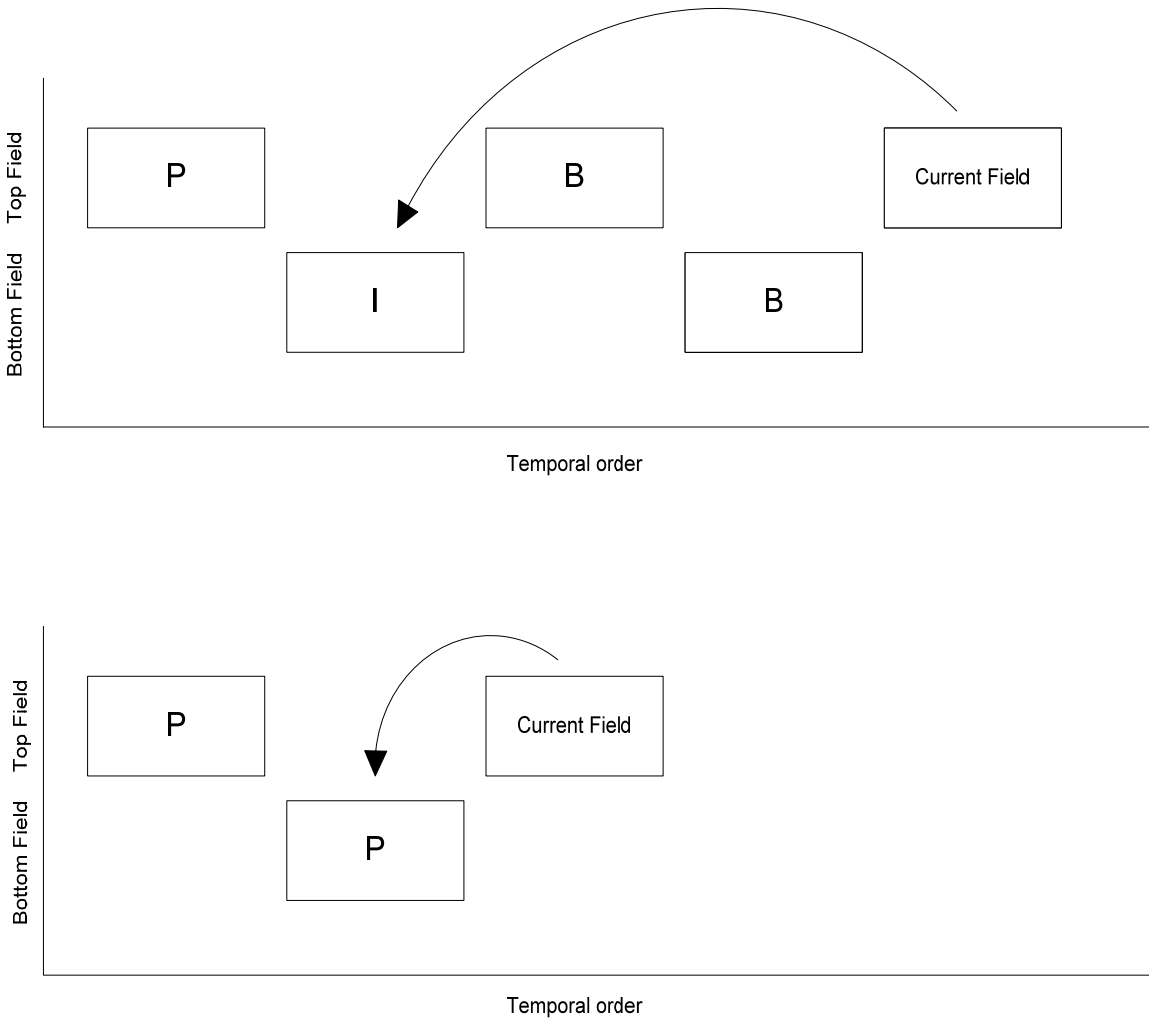


Figure 109: Example of one reference interlace field picture (NUMREF == 0) using temporally most recent reference (REFFIELD == 0)

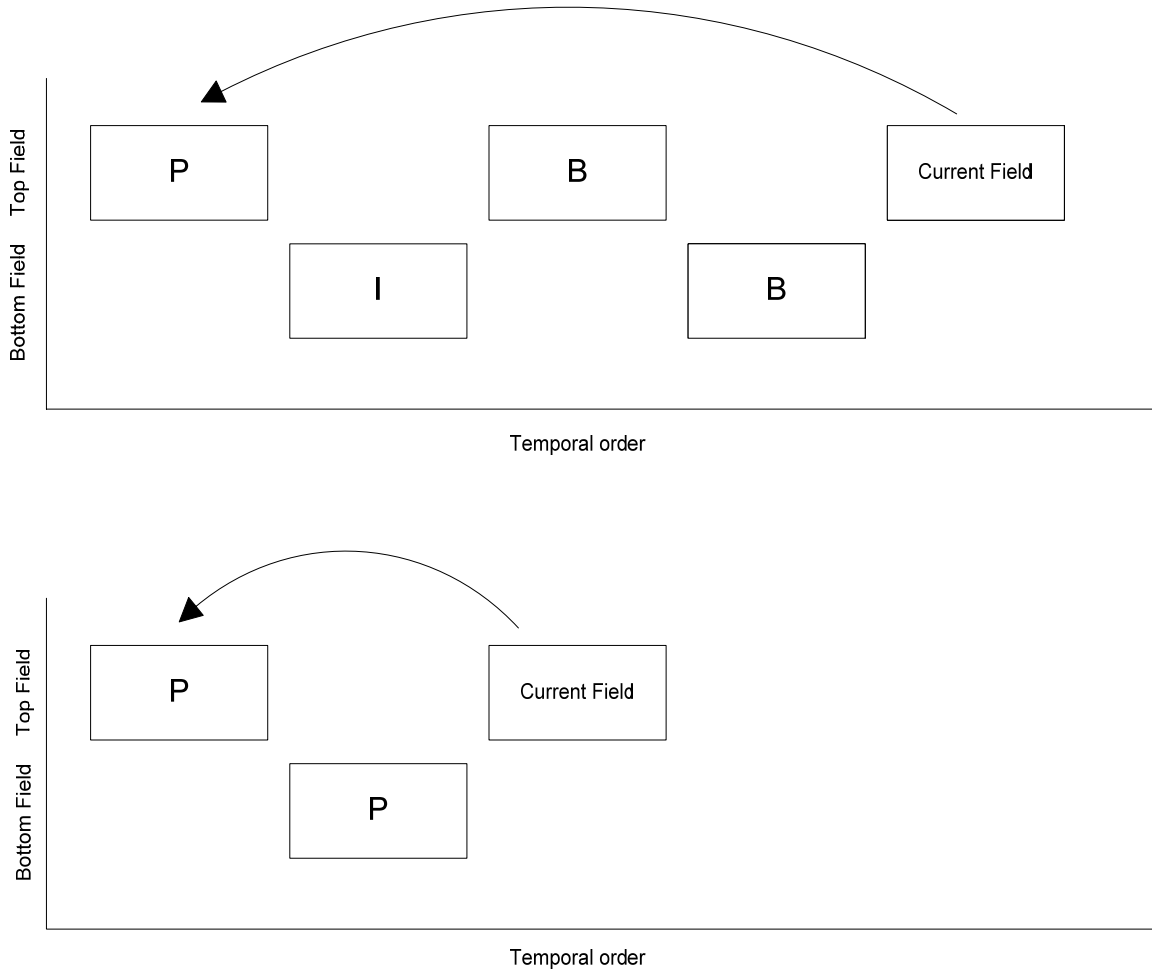


Figure 110: Example of one reference interlace field picture (NUMREF == 0) using temporally second-most recent reference (REFFIELD == 1)

10.3.4 P Picture Types

P pictures shall be one of two types: 1-MV or Mixed-MV. The following sections define each P picture type.

10.3.4.1 1-MV P Picture

In 1-MV P pictures, a single motion vector shall be used to indicate the displacement of the predicted blocks for all 6 blocks in the inter-coded macroblock. The 1-MV mode shall be signaled by the MVMODE and MVMODE2 picture layer syntax elements as defined in section 8.3.4.3.

10.3.4.2 Mixed-MV P Picture

In Mixed-MV P pictures, each inter-coded macroblock shall be encoded as a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luma blocks shall have an associated motion vector. The 1-MV or 4-MV mode for each macroblock shall be indicated by the MBMODE syntax element. The Mixed-MV mode shall be signaled by the MVMODE and MVMODE2 picture layer syntax elements as defined in section 8.3.4.3.

10.3.5 Macroblock Layer Decode

10.3.5.1 Macroblock Types

Macroblocks in P pictures shall be one of 3 possible types: 1-MV, 4-MV, and Intra. The macroblock type is signaled by the MBMODE syntax element in the macroblock layer. The following sections describe each type and how they are signaled.

10.3.5.1.1 1-MV Macroblocks

1-MV macroblocks can occur in 1-MV and Mixed-MV P pictures. A 1-MV macroblock is one where a single motion vector represents the displacement between the current and reference pictures for all 6 blocks in the macroblock. For 1-MV macroblocks, the MBMODE syntax element in the macroblock layer indicates three parameters:

1) That the macroblock type is 1-MV
2) Whether the CBPCY syntax element is present
3) Whether the MVDATA syntax element is present

If the MBMODE syntax element indicates that the CBPCY syntax element is present, then the CBPCY syntax element shall be present in the macroblock layer in the corresponding position. The CBPCY syntax element indicates which of the 6 blocks are coded in the block layer. If the MBMODE syntax element indicates that the CBPCY syntax element is not present, then CBPCY shall be assumed to equal to 0 and no block data shall be present for any of the 6 blocks in the macroblock.

If the MBMODE syntax element indicates that the MVDATA syntax element is present, then the MVDATA syntax element shall be present in the macroblock layer in the corresponding position. The MVDATA syntax element shall encode the motion vector differential. The motion vector differential is combined with the motion vector predictor to reconstruct the motion vector. If the MBMODE syntax element indicates that the MVDATA syntax element is not present, then the motion vector differential shall be assumed to be zero and therefore the motion vector shall be equal to the motion vector predictor.

10.3.5.1.2 4-MV Macroblocks

4-MV macroblocks shall only occur in Mixed-MV P pictures. A 4-MV macroblock is one where each of the 4 luma blocks in a macroblock has an associated motion vector which indicates the displacement between the current and reference pictures for that block. The displacement for the color-difference blocks shall be derived from the 4 luma motion vectors. This procedure shall be identical to the progressive picture case described in section 8.3.5.4.2. The difference between the current and reference blocks is encoded in the block layer.

For 4-MV macroblocks the MBMODE syntax element in the macroblock layer indicates two parameters:

1) That the macroblock type is 4-MV
2) Whether the CBPCY syntax element is present

The CBPCY syntax element indicates which of the 6 blocks are coded in the block layer. If the MBMODE syntax element indicates that the CBPCY syntax element is not present, then CBPCY shall be assumed to equal to 0 and no block data shall be present for any of the 6 blocks in the macroblock.

The 4MVBP syntax element indicates which of the 4 luma blocks contain non-zero motion vector differentials. The 4MVBP syntax element decodes to a value between 0 and 15. This value, when expressed as a binary value, represents a bit syntax element which shall indicate whether the motion vector for the corresponding luma block is present as defined in Figure 111.

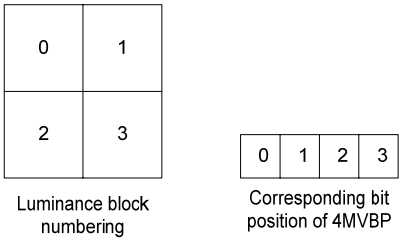


Figure 111: Association of bits in 4MVBP to luma blocks

For each of the 4 bit positions in the 4MVBP, a value of 0 shall indicate that no motion vector differential (BLKMVDATA (9.1.3.18)) is present for that block and the motion vector differential shall be assumed to be 0. A value of 1 indicates that a motion vector differential (BLKMVDATA) shall be present for that block in the corresponding position. For example, if 4MVBP decodes to a value of 1100b, then the bitstream contains BLKMVDATA for blocks 0 and 1 and no BLKMVDATA is present for blocks 2 and 3.

10.3.5.2 Intra Macroblocks

Intra macroblocks can occur in 1-MV or Mixed-MV P pictures. An Intra macroblock is one where all six blocks are coded without referencing any previous picture data. The block encodes the difference between each current block pixel and a constant value of 128.

For Intra macroblocks, the MBMODE syntax element (9.1.3.8) in the macroblock layer indicates two parameters:

1) That the macroblock type is Intra,

2) Whether the CBPCY syntax element is present.

If the MBMODE syntax element indicates that the CBPCY syntax element is present, then the CBPCY syntax element shall be present in the macroblock layer in the corresponding position. The CBPCY syntax element indicates which of the 6 blocks has AC coefficient data coded in the block layer. If the MBMODE syntax element indicates that the CBPCY syntax element is not present, then CBPCY shall be assumed to equal 0 and no AC coefficient data shall be present for any of the 6 blocks in the macroblock.

10.3.5.3 Macroblock Mode (MBMODE)

The MBMODE syntax element indicates the macroblock type (1-MV, 4-MV or Intra) and also the presence of the CBPCY syntax element and MV data, as described above. Depending on whether the MVMODE (9.1.1.46) and MVMODE2 (9.1.1.47) syntax elements indicate Mixed-MV or 1-MV the MBMODE signals the information as follows:

10.3.5.3.1 Macroblock Mode in 1-MV Pictures

When the MBMODE syntax element signals information about the macroblock in 1-MV pictures, the related parameters shall be as defined in Table 111.

Table 111: Macroblock Mode in 1-MV Pictures

MBMODE	MB Type	CBP Present	MVData Present
0	Intra MB	No	NA
1	Intra MB	Yes	NA
2	1-MV MB	No	No
3	1-MV MB	No	Yes
4	1-MV MB	Yes	No
5	1-MV MB	Yes	Yes

10.3.5.3.2 Macroblock Mode in Mixed-MV Pictures

When the MBMODE syntax element signals information about the macroblock in mixed-MV pictures, the related parameters shall be as defined in Table 112.

Table 112: Macroblock Mode in Mixed-MV Pictures

MBMODE	MB Type	CBP Present	MVData

			Present
0	Intra MB	No	NA
1	Intra MB	Yes	NA
2	1-MV MB	No	No
3	1-MV MB	No	Yes
4	1-MV MB	Yes	No
5	1-MV MB	Yes	Yes
6	4-MV MB	No	NA
7	4-MV MB	Yes	NA

One of 8 tables is used to signal the MBMODE. The table is signaled at the picture layer via the MBMODETAB syntax element. Table 144 through Table 151 shall be used for Mixed-MV MB mode. Table 152 through Table 159 shall be used for 1-MV MB mode.

10.3.5.4 Motion Vector Decoding Process

The following sub-sections define the motion vector decoding process for P interlace field picture macroblocks.

Note: In the sections that define the luma motion vector decoding process, all motion vectors are expressed in 1/2 or 1/4 pel units (depending on the value of MVMODE) and conform to the coordinate system shown in Figure 112. In the sections that define the color-difference motion vector derivation process, all motion vectors are expressed in1/4 pel units.

10.3.5.4.1 Interlace Field Picture Coordinate System

In the following sections which describe the motion vector decoding processes the motion vector units are expressed in field picture units. For example, if the vertical component a motion vector indicates that the displacement is +5 then this indicates a displacement of 1 ¼ field picture lines if the motion vector resolution is quarter-pel (MVMODE is Mixed-MV or 1-MV quarter-pel bicubic) or 2 ½ field picture lines if the motion vector resolution is half-pel (MVMODE is 1-MV half-pel bicubic or 1-MV half-pel bilinear). Unless otherwise stated, the motion vector decoding processes in the sections that follow assume that the motion vector is expressed in the resolution that corresponds to the MVMODE for the picture. Figure 112 shows the relationship between the vertical component of the motion vector and the spatial location for both combinations of current and reference field polarities. The figure shows one vertical column of pixels in the current and reference fields. Each circle represents integer pixel positions and the “x”s represent quarter or half pixel positions. The figure shows that a value of 0 indicates no vertical displacement between the current and reference field positions. As the figure shows, if the current and reference fields are of opposite polarities then the 0 vertical vector points to a position halfway between the field lines (a ½ pixel shift) in the reference field. If the current and reference fields are of the same polarity then the 0 vertical vector points to the corresponding field line in the reference field.

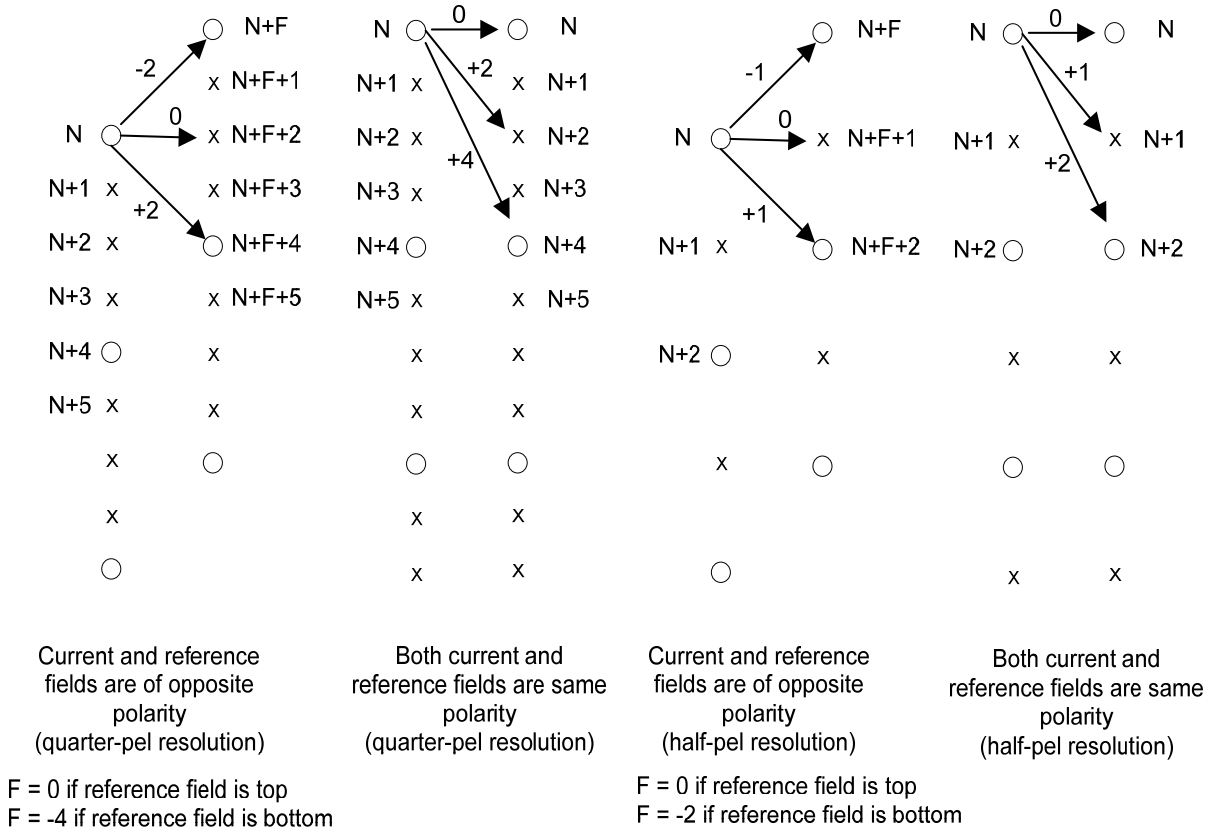


Figure 112: Vertical relationship between motion vectors for current and reference fields

10.3.5.4.2 Decoding Motion Vector Differential

The MVDATA or BLKMVDATA syntax elements encode motion information for the blocks in the macroblock. 1-MV macroblocks shall have a single MVDATA syntax element if MBMODE syntax element indicates that MVDATA is present as defined in 10.3.5.1.1, and 4-MV macroblocks shall have between zero and four BLKMVDATA syntax elements. The following sections describe how to compute the motion vector differential for the one-reference (picture layer syntax element NUMREF == 0) and two-reference (picture layer syntax element NUMREF == 1) cases.

10.3.5.4.2.1 Motion Vector Differentials in One-Reference Interlace Field Pictures

In interlace field pictures that have only one reference field, each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly encodes two parameters: 1) the horizontal motion vector differential component and 2) the vertical motion vector differential component.

The MVDATA or BLKMVDATA syntax elements are each a variable length codeword followed by a fixed length codeword. The value of the codeword determines the size of the fixed length codeword. The IMVTAB syntax element in the picture layer shall specify the table used to decode the variable sized codeword.

The pseudo-code of Figure 113 defines how the motion vector differential shall be decoded. The values **dmv_x** and **dmv_y** are computed in the following pseudo-code. The values are defined as follows:

dmv_x: differential horizontal motion vector component

dmv_y: differential vertical motion vector component

k_x, k_y: fixed length for long motion vectors

k_x and **k_y** depend on the motion vector range as defined by the MVRANGE symbol (section 7.1.1.9) and shall be as defined in Table 75.

extend_x: extended range for horizontal motion vector differential

extend_y: extended range for vertical motion vector differential

extend_x and **extend_y** shall be derived from the DMVRANGE syntax element as follows. If DMVRANGE indicates that extended range for the horizontal component is used, then **extend_x** = 1. Otherwise **extend_x** = 0. Similarly, if DMVRANGE indicates that extended range for the vertical component is used, then **extend_y** = 1 otherwise **extend_y** = 0.

The **offset_table1** and **offset_table2** are arrays and shall be defined as follows:

offset_table1[9] = {0, 1, 2, 4, 8, 16, 32, 64, 128,}
offset_table2[9] = {0, 1, 3, 7, 15, 31, 63, 127, 255}

Figure 113: Pseudo-code for Decoding MV Differentials in One Reference Interlace Field Pictures

```
int index = vlc_decode() // Use the table indicated by IMVTAB in the picture layer
if (index == 71)
{
      dmv_x = get_bits(k_x)
      dmv_y = get_bits(k_y)
}
else
{
          if (extend_x == 1)
      offset_table = offset_table2
          else
      offset_table = offset_table1
       index1 = (index + 1) % 9
          if (index1 != 0)
          {
                val = get_bits (index1 + extend_x)
                sign = 0 - (val & 1)
                dmv_x = sign ^ ((val >> 1) + offset_table[index1])
                dmv_x = dmv_x - sign
          }
          else
                dmv_x = 0

          if (extend_y == 1)
      offset_table = offset_table2
          else
      offset_table = offset_table1
       int index1 = (index + 1) / 9
          if (index1 != 0)
         {
                int val = get_bits (index1 + extend_y)
                sign = 0 - (val & 1)
                dmv_y = sign ^ ((val >> 1) + offset_table[index1])
                dmv_y = dmv_y – sign
          }
          else
                dmv_y = 0
```

```
}
```

10.3.5.4.2.2 Motion Vector Differentials in Two-Reference Interlace Field Pictures

Two-reference Interlace Field Pictures occur in the coding of interlace frames using field pictures. Each frame of the sequence is separated into two fields, and each field is coded using what is essentially the progressive code path. Field pictures often have two reference fields and the coding of field picture motion vectors in this case is described below.

In interlace field pictures that have two reference fields, each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly encodes three parameters: 1) the horizontal motion vector differential component, 2) the vertical motion vector differential component and 3) whether the dominant or non-dominant predictor is used, i.e. which of the two fields is referenced by the motion vector.

The MVDATA or BLKMVDATA syntax element is a variable length codeword followed by a fixed length codeword. The value of the codeword determines the size of the fixed length codeword. The IMVTAB syntax element in the picture layer defines the table used to decode the variable sized codeword.

The pseudo-code of Figure 114 defines how the motion vector differential and dominant/non-dominant predictor information shall be decoded.

The values **predictor_flag, dmv_x** and **dmv_y** are computed in the following pseudo-code. The values are defined as follows:

> **predictor_flag:** binary flag indicating whether the dominant or non-dominant motion vector predictor is used (0 = dominant predictor used, 1 = non-dominant predictor used)
>
> **dmv_x:** differential horizontal motion vector component
>
> **dmv_y:** differential vertical motion vector component
>
> **k_x, k_y:** fixed length for long motion vectors
>
> **extend_x:** extended range for horizontal motion vector differential
>
> **extend_y:** extended range for vertical motion vector differential
>
> **k_x** and **k_y** depend on the motion vector range as defined by the MVRANGE symbol (section 7.1.1.9) and shall be as defined in Table 75.
>
> **extend_x** and **extend_y** shall be derived from the DMVRANGE syntax element. If DMVRANGE indicates that extended range for the horizontal component is used, then **extend_x** = 1. Otherwise **extend_x** = 0. Similarly, if DMVRANGE indicates that extended range for the vertical component is used, then **extend_y** = 1 otherwise **extend_y** = 0.

size_table, offset_table1 and **offset_table2** are arrays and shall be defined as follows:

size_table[16] = {0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7}

offset_table1[9] = {0, 1, 2, 4, 8, 16, 32, 64, 128}
offset_table2[9] = {0, 1, 3, 7, 15, 31, 63, 127, 255}

```
int index = vlc_decode() // Use the table indicated by IMVTAB in the picture layer
int index1, val

if (index == 125)
{
    dmv_x = get_bits(k_x)
    dmv_y = get_bits(k_y)
    predictor_flag = dmv_y & 1
    dmv_y = (dmv_y + predictor_flag) >> 1
```

```
}
else
{
        if (extend_x == 1)
    offset_table = offset_table2
        else
    offset_table = offset_table1
        index1 = (index + 1) % 9
        if (index1 != 0)
        {
            val = get_bits (index1 + extend_x)
            sign = 0 - (val & 1)
            dmv_x = sign ^ ((val >> 1) + offset_table[index1])
            dmv_x = dmv_x - sign
        }
        else
            dmv_x = 0

        if (extend_y == 1)
    offset_table = offset_table2
        else
    offset_table = offset_table1
        index1 = (index + 1) / 9
        if (index1 != 0 && index1 != 1)
        {
            val = get_bits (size_table[index1 + 2 * extend_y])
            sign = 0 - (val & 1)
            dmv_y = sign ^ ((val >> 1) + offset_table[index1 >> 1])
            dmv_y = dmv_y – sign
        }
        else
        {
            dmv_y = 0
        }
        predictor_flag = index1 & 1
}
```

Figure 114: Pseudo-code for Decoding MV Differentials in Two Reference Interlace Field Pictures

10.3.5.4.3 Motion Vector Predictors

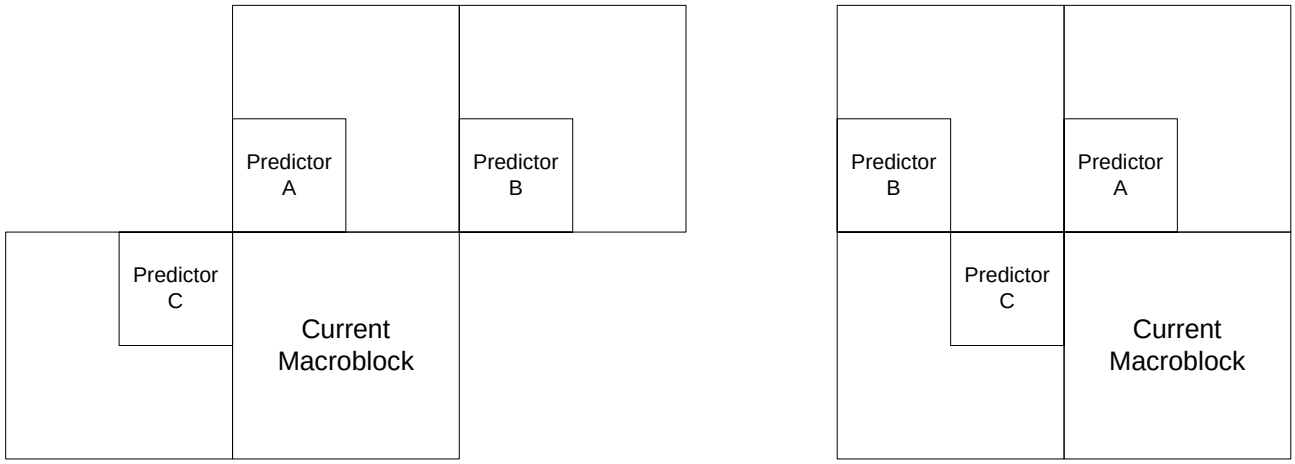
Motion vectors are computed by adding the motion vector differential computed in the previous section to a motion vector predictor. The predictor is computed from three neighboring motion vectors.

The following sections describe how the predictors are calculated for macroblocks in ‘1-MV’ P pictures and ‘Mixed-MV’ P pictures.

10.3.5.4.3.1 Motion Vector Predictors In 1-MV P Pictures

Figure 51 (section 8.3.5.3.1) shows the three motion vectors used to compute the predictor for the current macroblock. The predictors shall be derived as described in section 8.3.5.3.1.

10.3.5.4.3.2 Motion Vector Predictors In Mixed-MV P Pictures



Not last macroblock in macroblock row

Last macroblock in macroblock row

Figure 115: Candidate Motion Vectors for 1-MV Macroblocks in Mixed-MV Interlace Field Pictures

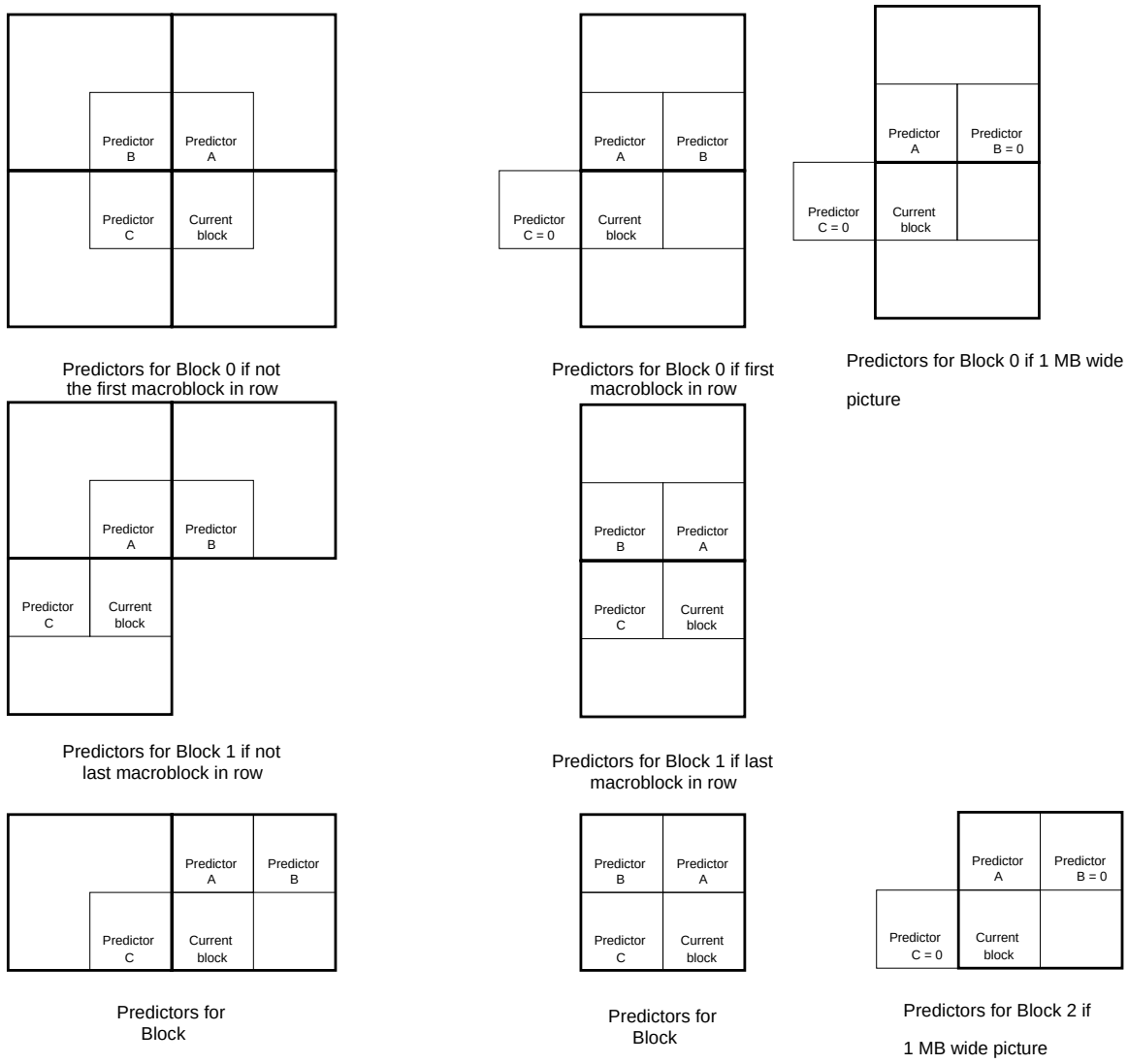


Figure 116: Candidate Motion Vectors for 4-MV Macroblocks in Mixed-MV Interlace Field Pictures

Figure 115 and Figure 116 show the 3 candidate motion vectors for 1-MV and 4-MV macroblocks in Mixed-MV P pictures. The larger rectangles are macroblock boundaries and the smaller rectangles are block boundaries.

For the special case where the frame is one macroblock wide, the predictor shall always be Predictor A (the top predictor).

Figure 115 shows the candidate motion vectors for 1-MV macroblocks. The neighboring macroblocks can be 1-MV or 4-MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4-MV (i.e., predictor A is the motion vector for block 2 in the macroblock above the current and predictor C is the motion vector for block 1 in the macroblock immediately to the left of the current). If any of the neighbors are 1-MV macroblocks, then the motion vector predictors shown in Figure 115 shall be taken to be the vectors for the entire macroblock. As the figure shows, if the macroblock is the last macroblock in the row, then Predictor B shall be from block 2 of the top-left macroblock instead of from block 2 in the top-right macroblock as is the case otherwise.

Figure 116 shows the predictors for each of the 4 luma blocks in a 4-MV macroblock. For the case where the macroblock is the first macroblock in the row, Predictor B for block 0 is handled differently than the remaining blocks in the row. In this case, Predictor B shall be taken from block 3 in the macroblock immediately above the current macroblock instead of from block 3 in the macroblock above and to the left of current macroblock, as is the case otherwise. Similarly, for the case where the macroblock is the last macroblock in the row, Predictor B for block 1 is handled differently. In this case, the predictor shall be taken from block 2 in the macroblock immediately above the

current macroblock instead of from block 2 in the macroblock above and to the left of the current macroblock, as is the case otherwise. If the macroblock is in the first macroblock column, then Predictor C for blocks 0 and 2 shall be set equal to 0.

10.3.5.4.3.3 Dominant and Non-Dominant MV Predictors

In two-reference field P pictures, for each inter-coded macroblock, two motion vector predictors are derived. One is from the dominant field and the other is from the non-dominant field. The dominant field shall be the field containing the majority of the motion vector predictor candidates. In the case of a tie, the motion vector derived from the opposite field shall be the dominant predictor. Intra-coded macroblocks shall not be considered in the calculation of the dominant/non-dominant predictor. If all candidate predictor macroblocks are Intra-coded, then the dominant and non-dominant motion vector predictors shall be set to zero and the dominant predictor shall be from the opposite field.

10.3.5.4.3.4 Calculating the Motion Vector Predictor

If the NUMREF syntax element in the picture header == 0, then the current interlace field picture shall refer to only one previously coded picture. If NUMREF == 1, then the current interlace field picture shall refer to the two most recent field pictures. If NUMREF == 0, a single predictor shall be calculated for each motion vector. If NUMREF == 1, two motion vector predictors shall be calculated. The pseudo-code of Figure 117 and Figure 118 describe how the motion vector predictors are calculated for each case. The variables *fieldpred_x* and *fieldpred_y* represent the horizontal and vertical components of the motion vector predictor.

10.3.5.4.3.4.1 Motion Vector Predictors in One-Reference Interlace Field Pictures

The pseudo-code of Figure 117 shall be used to calculate the motion vector predictors in one-reference interlace field pictures.

```
if (predictorA is not out of bounds) {
    if (predictorC is not out of bounds) {
        if (predictorA is intra) {
            if (predictorB is intra) {
                if (predictorC is intra) {
                    // A, B and C are intra
                    fieldpred_x = 0
                    fieldpred_y = 0
                }
                else {
                    // A and B are intra
                    fieldpred_x = predictorC_x
                    fieldpred_y = predictorC_y
                }
            }
            else {
                // A is intra B is inter
                if (predictorC is intra) {
                    // A and C are intra
                    fieldpred_x = predictorB_x
                    fieldpred_y = predictorB_y
                }
                else {
                    // B and C are inter
                    fieldpred_x =
                        median (0, predictorB_x, predictorC_x)
                    fieldpred_y =
                        median (0, predictorB_y, predictorC_y)
                }
            }
        }
        else {
            // A is inter
            if (predictorB is intra) {
                if (predictorC is intra) {
                    // A is inter B and C are intra
                    fieldpred_x = predictorA_x
                    fieldpred_y = predictorA_y
                }
                else {
                    // A and C are inter and B is intra
                    fieldpred_x =
                        median (0, predictorA_x, predictorC_x)
                    fieldpred_y =
                        median (0, predictorA_y, predictorC_y)
                }
            }
```

```
            else {
                // A and B are inter
                if (predictorC is intra) {
                    // A and B are inter and C is intra
                    fieldpred_x =
                        median (0, predictorA_x, predictorB_x)
                    fieldpred_y =
                        median (0, predictorA_y, predictorB_y)
                }
                else {
                    // A, B and C are inter
                    fieldpred_x =
                        median (predictorA_x, predictorB_x, predictorC_x)
                    fieldpred_y =
                        median (predictorA_x, predictorB_y, predictorC_y)
                }
            }
        }
    }
    else {
        // predictorC is out of bounds
        if (only 1 macroblock per row) {
            if (predictorA is intra) {
                fieldpred_x = 0
                fieldpred_y = 0
            }
            else {
                // Use predictorA
                fieldpred_x = predictorA_x
                fieldpred_y = predictorA_y
            }
        }
        else {
            // Predictor C is out of bounds, use Predictor A and Predictor B
            if (predictorA is intra) {
                if (predictorB is intra) {
                    fieldpred_x = 0
                    fieldpred_y = 0
                }
                else {
                    fieldpred_x = predictorB_x
                    fieldpred_y = predictorB_y
                }
            }
            else {
                if (predictorB is intra) {
```

```
                    fieldpred_x = predictorA_x
                    fieldpred_y = predictorA_y
                }
                else {
                    fieldpred_x =
                        median (predictorA_x, predictorB_x, 0)
                    fieldpred_y =
                        median (predictorA_x, predictorB_y, 0)
                }
            }
        }
    }
}
else {
    // Predictor A is out of bounds
    if (predictorC is out of bounds or predictorC is intra) {
        fieldpred_x = 0
        fieldpred_y = 0
    }
    else {
        // Use predictorC
        fieldpred_x = predictorC_x
        fieldpred_y = predictorC_y
    }
}
```

Figure 117: MV Predictor in One Reference Interlace Field Pictures

10.3.5.4.3.4.2 Motion Vector Predictors in Two-Reference Interlace Field Pictures

In 2-reference pictures (NUMREF == 1) the current field shall reference the two most recent fields. In this case two motion vector predictors shall be computed for each macroblock. One predictor shall be from the reference field of the same polarity and the other shall be from the reference field with the opposite polarity.

Given the 3 motion vector predictor candidates, the pseudo-code of Figure 118 shall define the process for calculating the motion vector predictors. The variables *samefieldpred_x* and *samefieldpred_y* represent the horizontal and vertical components of the motion vector predictor from the same field and *oppositefieldpred_x* and *oppositefieldpred_y* represent the horizontal and vertical components of the motion vector predictor from the opposite field. The variable *dominantpredictor* indicates which field contains the dominant predictor. The value *predictor_flag* decoded from the motion vector differential indicates whether the dominant or non-dominant predictor is used. A predictor candidate that is out-of-bounds shall not be used in the computation of the predictor.

```
samecount = 0;
oppositecount = 0;
if (predictorA is not out of bounds) {
    if (predictorC is not out of bounds) {
        if (predictorA is intra) {
            samefieldpred_x = oppositefieldpred_x = samefieldpredA_x = oppositefieldpredA_x   = 0
            samefieldpred_y = oppositefieldpred_y = samefieldpredA_y = oppositefieldpredA_y   = 0
        }
```

```
if (predictorB is intra) {
    samefieldpred_x = oppositefieldpred_x   = samefieldpredB_x = oppositefieldpredB_x   = 0
    samefieldpred_y = oppositefieldpred_y = samefieldpredB_y = oppositefieldpredB_y   = 0
}
if (predictorC is intra) {
    samefieldpred_x = oppositefieldpred_x = samefieldpredC_x = oppositefieldpredC_x   = 0
    samefieldpred_y = oppositefieldpred_y = samefieldpredC_y = oppositefieldpredC_y   = 0
}
if (predictorA is not intra) {
    if (predictorA is from same field) {
        samecount = samecount + 1
        samefieldpred_x   = samefieldpredA_x = predictorA_x
        samefieldpred_y   = samefieldpredA_y = predictorA_y
        oppositefieldpred_x   = oppositefieldpredA_x = scaleforopposite_x(predictorA_x)
        oppositefieldpred_y   = oppositefieldpredA_y = scaleforopposite_y(predictorA_y)
    }
    else {
        oppositecount = oppositecount + 1
        oppositefieldpred_x   = oppositefieldpredA_x = predictorA_x
        oppositefieldpred_y   = oppositefieldpredA_y = predictorA_y
        samefieldpred_x   = samefieldpredA_x = scaleforsame_x(predictorA_x)
        samefieldpred_y   = samefieldpredA_y = scaleforsame_y(predictorA_y)
    }
}
if (predictorB is not intra) {
    If (predictorB is from same field) {
        samecount = samecount + 1
        samefieldpred_x = samefieldpredB_x = predictorB_x
        samefieldpred_y = samefieldpredB_y = predictorB_y
        oppositefieldpred_x = oppositefieldpredB_x = scaleforopposite_x(predictorB_x)
        oppositefieldpred_y = oppositefieldpredB_y = scaleforopposite_y(predictorB_y)
    }
    else {
        oppositecount   = oppositecount + 1
        oppositefieldpred_x = oppositefieldpredB_x = predictorB_x
        oppositefieldpred_y = oppositefieldpredB _y = predictorB_y
        samefieldpred_x = samefieldpredB_x = scaleforsame_x(predictorB_x)
        samefieldpred_y = samefieldpredB_y = scaleforsame_y(predictorB_y)
    }
}
if (predictorC is not intra) {
    if (predictorC is from same field) {
        samecount = samecount + 1
        samefieldpred_x = samefieldpredC_x = predictorC_x
        samefieldpred_y = samefieldpredC_y = predictorC_y
        oppositefieldpred_x = oppositefieldpredC_x = scaleforopposite_x(predictorC_x)
```

```
                oppositefieldpred_y = oppositefieldpredC_y = scaleforopposite_y(predictorC_y)
            }
            else {
                oppositecount = oppositecount + 1
                oppositefieldpred_x   = oppositefieldpredC_x = predictorC_x
                oppositefieldpred_y = oppositefieldpredC _y = predictorC_y
                samefieldpred_x = samefieldpredC _x = scaleforsame_x(predictorC_x)
                samefieldpred_y = samefieldpredC _y = scaleforsame_y(predictorC_y)
            }
        }
        if ((samecount + oppositecount) > 1) {
            samefieldpred_x =
                median (samefieldpredA_x, samefieldpredB_x, samefieldpredC_x)
            samefieldpred_y =
                median (samefieldpredA_y, samefieldpredA_y, samefieldpredC_y)
            oppositefieldpred_x =
                median (oppositefieldpredA_x, oppositefieldpredB_x, oppositefieldpredC_x)
            oppositefieldpred_y =
                median (oppositefieldpredA_y, oppositefieldpredB_y, oppositefieldpredC_y)
        }

        if (samecount > oppositecount)
            dominantpredictor = samefield
        else
            dominantpredictor = oppositefield
    }
    else {
        // predictorC is out of bounds
        if (only 1 macroblock per row) {
            if (predictorA is intra) {
                samefieldpred_x = oppositefieldpred_x = 0
                samefieldpred_y = oppositefieldpred_y = 0
                dominantpredictor = oppositefield
            }
            else {
                // Use predictorA
                if (predictorA is from same field) {
                    samefieldpred_x = predictorA_x
                    samefieldpred_y = predictorA_y
                    oppositefieldpred_x = scaleforopposite_x(predictorA_x)
                    oppositefieldpred_y = scaleforopposite_y(predictorA_y)
                    dominantpredictor = samefield
                }
                else {
                    oppositefieldpred_x = predictorA_x
                    oppositefieldpred_y = predictorA_y
                    samefieldpred_x = scaleforsame_x(predictorA_x)
```

```
                samefieldpred_y = scaleforsame_y(predictorA_y)
                dominantpredictor = oppositefield
            }
        }
    }
    else {
        // Predictor C is out of bounds, use Predictor and PredictorB
        predictorC_x = 0
        predictorC_y = 0
        if (predictorA is intra) {
            samefieldpred_x = oppositefieldpred_x = samefieldpredA_x = oppositefieldpredA_x   = 0
            samefieldpred_y = oppositefieldpred_y = samefieldpredA_y = oppositefieldpredA_y   = 0
        }
        if (predictorB is intra) {
            samefieldpred_x = oppositefieldpred_x = samefieldpredB_x = oppositefieldpredB_x   = 0
            samefieldpred_y = oppositefieldpred_y = samefieldpredB_y = oppositefieldpredB_y   = 0
        }
        samefieldpred_x = oppositefieldpred_x = samefieldpredC_x = oppositefieldpredC_x = 0
        samefieldpred_y = oppositefieldpred_y = samefieldpredC_y = oppositefieldpredC_y   = 0
        if (predictorA is not intra) {
            if (predictorA is from same field) {
                samecount = samecount + 1
                samefieldpred_x = samefieldpredA_x = predictorA_x
                samefieldpred_y = samefieldpredA_y = predictorA_y
                oppositefieldpred_x = oppositefieldpredA_x = scaleforopposite_x(predictorA_x)
                oppositefieldpred_y = oppositefieldpredA_y = scaleforopposite_y(predictorA_y)
            }
            else {
                oppositecount = oppositecount + 1
                oppositefieldpred_x = oppositefieldpredA_x = predictorA_x
                oppositefieldpred_y = oppositefieldpredA _y = predictorA_y
                samefieldpred_x = samefieldpredA_x = scaleforsame_x(predictorA_x)
                samefieldpred_y = samefieldpredA_y = scaleforsame_y(predictorA_y)
            }
        }
        if (predictorB is not intra) {
            if (predictorB is from same field) {
                samecount   = samecount + 1
                samefieldpred_x = samefieldpredB_x = predictorB_x
                samefieldpred_y = samefieldpredB_y = predictorB_y
                oppositefieldpred_x = oppositefieldpredB_x = scaleforopposite_x(predictorB_x)
                oppositefieldpred_y = oppositefieldpredB_y = scaleforopposite_y(predictorB_y)
            }
            else {
                oppositecount = oppositecount + 1
                oppositefieldpred_x = oppositefieldpredB_x = predictorB_x
```

```
                    oppositefieldpred_y = oppositefieldpredB_y = predictorB_y
                    samefieldpred_x = samefieldpredB_x = scaleforsame_x(predictorB_x)
                    samefieldpred_y = samefieldpredB_y = scaleforsame_y(predictorB_y)
                }
            }
            if ((samecount + oppositecount) > 1) {
                samefieldpred_x =
                    median (samefieldpredA_x, samefieldpredB_x, samefieldpredC_x)
                samefieldpred_y =
                    median (samefieldpredA_y, samefieldpredA_y, samefieldpredC_y)
                oppositefieldpred_x =
                     median (oppositefieldpredA_x, oppositefieldpredB_x, oppositefieldpredC_x)
                oppositefieldpred_y =
                    median (oppositefieldpredA_y, oppositefieldpredB_y, oppositefieldpredC_y)
            }
            if (samecount > oppositecount)
                dominantpredictor = samefield
            else
                dominantpredictor = oppositefield
        }
    }
}
else {
    // Predictor A is out of bounds
    if (predictorC is out of bounds or predictorC is intra) {
        samefieldpred_x = oppositefieldpred_x = 0
        samefieldpred_y = oppositefieldpred_y = 0
        dominantpredictor = oppositefield
    }
    else {
        // Use predictorC
        if (predictorC is from same field) {
            samefieldpred_x = predictorC_x
            samefieldpred_y = predictorC_y
            oppositefieldpred_x = scaleforopposite_x(predictorC_x)
            oppositefieldpred_y = scaleforopposite_y(predictorC_y)
            dominantpredictor = samefield
        }
        else {
            oppositefieldpred_x = predictorC_x
            oppositefieldpred_y = predictorC_y
            samefieldpred_x = scaleforsame_x(predictorC_x)
            samefieldpred_y = scaleforsame_y(predictorC_y)
            dominantpredictor = oppositefield
        }
    }
}
```

Figure 118: MV Predictor in Two Reference Interlace Field Pictures

The scaling operations used in the pseudo-code of Figure 118 to derive the other field's predictor shall be as defined in the pseudo-code of Figure 119.

```
scaleforopposite_x (n)   {
      int scaledvalue
      scaledvalue = (n * SCALEOPP) >> 8
      return scaledvalue
}
scaleforopposite_y (n)   {
      int scaledvalue
      scaledvalue = (n * SCALEOPP) >> 8
      return scaledvalue
}
scaleforsame_x (n)   {
     int scaledvalue
     if (abs (n) > 255)
          scaledvalue = n
     else {
          if (abs (n) < SCALEZONE1_X)
               scaledvalue = (n * SCALESAME1) >> 8
          else {
               if (n < 0)
                    scaledvalue = ((n * SCALESAME2) >> 8) – ZONE1OFFSET_X
               else
                    scaledvalue = ((n * SCALESAME2) >> 8) + ZONE1OFFSET_X
          }
     }

     if (scaledvalue > range_x – 1)
          scaledvalue = range_x – 1
     if (scaledvalue _x < -range_x)
          scaledvalue = -range_x

     return scaledvalue
}
scaleforsame_y (n)   {
     int scaledvalue
     if (abs (n) > 63)
          scaledvalue = n
     else {
          if (abs (n) < SCALEZONE1_Y)
               scaledvalue = (n * SCALESAME1) >> 8
          else {
               if (n < 0)
```

```
                scaledvalue = ((n * SCALESAME2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = ((n * SCALESAME2) >> 8) + ZONE1OFFSET_Y
        }
    }

    if (current field is bottom field and reference field is top field) {
        If (scaledvalue > range_y / 2)
            scaledvalue = range_y / 2
        If (scaledvalue < -(range_y / 2) + 1)
            scaledvalue = -(range_y / 2) + 1
    }
    else {
        If (scaledvalue > (range_y / 2) – 1)
            scaledvalue = (range_y / 2) – 1
        If (scaledvalue < -(range_y / 2)
            scaledvalue = -(range_y / 2)
    }

    return scaledvalue
}
```

Figure 119: Scaling Operation for MV Prediction in Two Reference Interlace Field Pictures

The values range_x and range_y depend on MVRANGE and shall be as specified in Table 75 (section 8.3.5.2.1).

The values of the variables SCALEOPP, SCALESAME1, SCALESAME2, SCALEZONE1_X, SCALEZONE1_Y, ZONE1OFFSET_X and ZONE1OFFSET_Y shall be as defined in Table 113 for the case where the current field is the first field and shall be as defined in Table 114 for the case where the current field is the second field. The reference frame distance is coded in the REFDIST syntax element (9.1.1.43) in the picture header.

Table 113: P Interlace Field Picture MV Predictor Scaling Values when Current Field is First

	Reference Frame Distance (REFDIST)			
	0	1	2	3 or greater
SCALEOPP	128	192	213	224
SCALESAME1	512	341	307	293
SCALESAME2	219	236	242	245
SCALEZONE1_X	32	48	53	56
SCALEZONE1_Y	8	12	13	14
ZONE1OFFSET_X	37	20	14	11
ZONE1OFFSET_Y	10	5	4	3

Table 114: P Interlace Field Picture MV Predictor Scaling Values when Current Field is Second

	Reference Frame Distance (REFDIST)

	0	1	2	3 or greater
SCALEOPP	128	64	43	32
SCALESAME1	512	1024	1536	2048
SCALESAME2	219	204	200	198
SCALEZONE1_X	32	16	11	8
SCALEZONE1_Y	8	4	3	2
ZONE1OFFSET_X	37	52	56	58
ZONE1OFFSET_Y	10	13	14	15

10.3.5.4.3.5 Hybrid Motion Vector Prediction (HYBRIDPRED)

The motion predictor calculated in the previous section is tested relative to the A (top) and C (left) predictors to see if the predictor is explicitly coded in the bitstream. If explicitly coded, then the HYBRIDPRED syntax element will be present to indicate whether to use predictor A or predictor C as the motion vector predictor. The pseudo-code of Figure 120 shall specify the decoding of HYBRIDPRED syntax element, and shall define the hybrid motion vector prediction.

For two-reference field pictures, the pseudo-code of Figure 120 has the following correspondence with the pseudo-code of Figure 118: the values predictor_pre, predictor_post, predictorA, predictorB and predictorC all represent fields of the same polarity or opposite polarity as the current field as determined by the values of predictor_flag and dominantpolarity and the procedure described in section 10.3.5.4.4.1.

For example, if the predictor_flag and dominantpredictor indicate that the opposite field predictor is used, then:

predictor_pre_x = oppositefieldpred_x

predictor_pre_y = oppositefieldpred_y

predictorA_x = oppositefieldpredA_x

predictorA_y = oppositefieldpredA_y

predictorB_x = oppositefieldpredB_x

predictorB_y = oppositefieldpredB_y

predictorC_x = oppositefieldpredC_x

predictorC_y = oppositefieldpredC_y

Likewise, if the predictor_flag and dominantpredictor indicate that the same field predictor is used then:

predictor_pre_x = samefieldpred_x

predictor_pre_y = samefieldpred_y

predictorA_x = samefieldpredA_x

predictorA_y = samefieldpredA_y

predictorB_x = samefieldpredB_x

predictorB_y = samefieldpredB_y

predictorC_x = samefieldpredC_x

predictorC_y = samefieldpredC_y

where the values of oppositefieldpred and samefieldpred are calculated as described in section 10.3.5.4.3.4.2.

For one reference field pictures, the pseudo-code of Figure 120 has the following correspondence with the pseudo-code of Figure 117:

predictor_pre_x = fieldpred_x

predictor_pre_y = fieldpred_y

The variables are defined as follows in the pseudo-code:

```
predictor_pre_x: The horizontal motion vector predictor as calculated in the above section
predictor_pre_y: The vertical motion vector predictor as calculated in the above section
predictor_post_x: The horizontal motion vector predictor after checking for hybrid motion vector prediction
predictor_post_y: The vertical motion vector predictor after checking for hybrid motion vector prediction
hybridmv_thresh: The threshold for determining whether hybrid motion vector prediction is used.

if (MVMODE is Mixed-MV or 1-MV quarter pel)
    hybridmv_thresh = 32
else // MVMODE is 1-MV half pel bicubic or 1-MV half pel bilinear)
    hybridmv_thresh = 16

if ((predictorA is out of bounds) || (predictorC is out of bounds) || (predictorA is intra) || (predictorC is intra)) {
    predictor_post_x = predictor_pre_x
    predictor_post_y = predictor_pre_y
}
else {
    int sumA = abs(predictor_pre_x – predictorA_x) + abs(predictor_pre_y – predictorA_y)
    int sumC = abs(predictor_pre_x – predictorC_x) + abs(predictor_pre_y – predictorC_y)
    if (sumA > hybridmv_thresh) {
        // read next bit to see which predictor candidate to use
        if (get_bits(1) == 1) { // HYBRIDPRED field
            // use top predictor (predictorA)
            predictor_post_x = predictorA_x
            predictor_post_y = predictorA_y
        }
        else {
            // use left predictor (predictorC)
            predictor_post_x = predictorC_x
            predictor_post_y = predictorC_y
        }
    }
    else if (sumC > hybridmv_thresh){
        if (get_bits(1) == 1) {
            // use top predictor (predictorA)
            predictor_post_x = predictorA_x
            predictor_post_y = predictorA_y
        }
```

```
            else {
                // use left predictor (predictorC)
                predictor_post_x = predictorC_x
                predictor_post_y = predictorC_y
            }
    }
    else {
        predictor_post_x = predictor_pre_x
        predictor_post_y = predictor_pre_y
     }
}
```

Figure 120: Hybrid MV Prediction in Interlace Field Pictures

10.3.5.4.4 Reconstructing Motion Vectors

The following sections define how to reconstruct the luma and color-difference motion vectors for 1-MV and 4-MV macroblocks.

10.3.5.4.4.1 Luma Motion Vector Reconstruction

In all cases (1-MV and 4-MV macroblocks) the luma motion vector shall be reconstructed by adding the differential to the predictor as follows, where mv_x and mv_y are the horizontal and vertical components of luma motion vector:

For NUMREF == 0 (one reference interlace field picture) and reference is from the same field polarity:

mv_x = (dmv_x + predictor_post_x) smod (range_x)

mv_y = (dmv_y + predictor_post_y) smod (range_y)

For NUMREF == 0 (one reference interlace field picture) and reference is from the opposite field polarity:

mv_x = (dmv_x + predictor_post_x) smod (range_x)

If current field is top field

mv_y = (dmv_y + predictor_post_y) smod (range_y)

If current field is bottom field

mv_y = ((dmv_y + predictor_post_y - 1) smod (range_y)) + 1

For NUMREF == 1 (two reference interlace field picture) and reference is from the same field polarity:

mv_x = (dmv_x + predictor_post_x) smod (range_x)

mv_y = (dmv_y + predictor_post_y) smod (range_y / 2)

For NUMREF == 1 (two reference interlace field picture) and reference is from opposite field polarity:

mv_x = (dmv_x + predictor_post_x) smod (range_x)

If current field is top field

mv_y = (dmv_y + predictor_post_y) smod (range_y / 2)

If current field is bottom field

mv_y = ((dmv_y + predictor_post_y - 1) smod (range_y / 2)) + 1

range_x and range_y depend on MVRANGE and shall be as specified in Table 75.

If the picture uses two reference pictures (NUMREF == 1), then the *predictor_flag* derived after decoding the motion vector differential shall be combined with the value of *dominantpredictor* derived from motion vector prediction to determine which field is used as reference. The pseudo-code of Figure 121 shall describe how the reference field is determined:

```
if (predictor_flag == 0) {
    if (dominantpredictor == samefield)
        reference is from same field as current field
    else
        reference is from opposite field as current field
}
else {
    // predictor_flag == 1
    if (dominantpredictor == samefield)
        reference is from opposite field as current field
    else
        reference is from same field as current field
}
```

Figure 121: Pseudo-code for determining Reference Field in Two Reference Interlace Field Pictures

1-MV Macroblock

In 1-MV macroblocks there is a single motion vector for the 4 blocks that make up the luma component of the macroblock (see section 9.1.3.12).

If the MBMODE syntax element indicates that no MV data is present in the macroblock layer, then dmv_x = 0 and dmv_y = 0 (thus mv_x = predictor_post_x and mv_y = predictor_post_y).

4-MV Macroblock

Each of the Inter-coded luma blocks in a macroblock has its own motion vector. Therefore there are 4 luma motion vectors in each 4-MV macroblock.

If the 4MVBP syntax element indicates that no motion vector information is present for a block, then dmv_x = 0 and dmv_y = 0 for that block (thus mv_x = predictor_post_x and mv_y = predictor_post_y for that block).

10.3.5.4.4.2 Color-difference Motion Vector Reconstruction

The color-difference motion vectors shall be derived from the luma motion vectors. The following sections describe how to reconstruct the color-difference motion vectors for 1-MV and 4-MV macroblocks. The color-difference motion vectors shall be reconstructed in two steps.

As a first step, the nominal color-difference motion vector shall be obtained by combining and scaling the luma motion vectors appropriately. The scaling shall be performed in such a way that half-pixel offsets are preferred over quarter pixel locations.

In the second step, the 1-bit FASTUVMC syntax element (6.2.6) shall be used to determine if further rounding of color-difference motion vectors is necessary. If FASTUVMC == 0, no rounding shall be performed in the second step. If FASTUVMC == 1, the color-difference motion vectors that are at quarter pel offsets shall be rounded to the nearest half and full pel positions as defined in section 8.3.5.4.5.

Bilinear filtering only shall be used for all color-difference interpolation.

In the sub-sections below cmv_x and cmv_y denote the color-difference motion vector components and lmv_x and lmv_y denote the luma motion vector components. The vertical motion vector components, lmv_y and cmv_y, shall conform to the coordinate system shown in Figure 112. The luma motion vectors used to derive the color-difference motion vectors (the motion vectors used in the median, rounding and shifting operations) are expressed in quarter pel units in all cases.

10.3.5.4.4.2.1 First-stage Color-difference Motion Vector Reconstruction – 1-MV Color-difference Motion Vector Case:

In a 1-MV macroblock, the color-difference motion vectors shall be derived from the luma motion vectors as follows:

cmv_x = (lmv_x + round[1mv_x & 3]) >> 1

cmv_y = (lmv_y + round[1mv_y & 3]) >> 1

Where:

round[0] = 0, round[1] = 0, round[2] = 0, round[3] = 1

The color-difference reference field is same as the luma reference field.

10.3.5.4.4.2.2 First-stage Color-difference Motion Vector Reconstruction – 4-MV Color-difference Motion Vector Case:

The pseudo-code of Figure 122 shall define how the color-difference motion vectors are derived from the motion information in the 4 luma blocks in 4-MV macroblocks for one-reference P pictures. In this section, ix and iy are temporary variables.

```
// lmv0_x, lmv0_y is the motion vector for block 0
// lmv1_x, lmv1_y is the motion vector for block 1
// lmv2_x, lmv2_y is the motion vector for block 2
// lmv3_x, lmv3_y is the motion vector for block 3
ix = median4(lmv0_x, lmv1_x, lmv2_x, lmv3_x)
iy = median4(lmv0_y, lmv1_y, lmv2_y, lmv3_y)
cmv_x = (ix + round[ix & 3]) >> 1
cmv_y = (iy + round[iy & 3]) >> 1
Where:
round[0] = 0, round[1] = 0, round[2] = 0 and round[3] = 1.
color-difference reference field == luma reference field.
```

Figure 122: Color-difference MV Derivation in One Reference Interlace Field Pictures

The pseudo-code of Figure 123 shall define how the color-difference motion vectors are derived from the motion information in the 4 luma blocks in 4-MV macroblocks for two-reference P pictures. The variable p is the polarity of the reference field (p=0 if current and reference field are the same, p=1 if current and reference field are opposite).

```
if (all 4 luma block motion vectors are of reference polarity p)
{
    // lmv0_x, lmv0_y is the motion vector for block 0
    // lmv1_x, lmv1_y is the motion vector for block 1
    // lmv2_x, lmv2_y is the motion vector for block 2
    // lmv3_x, lmv3_y is the motion vector for block 3
    ix = median4(lmv0_x, lmv1_x, lmv2_x, lmv3_x)
```

```
        iy = median4(lmv0_y, lmv1_y, lmv2_y, lmv3_y)
        The color-difference reference field is p.
}
else if (3 of the luma block motion vectors are of reference polarity p)
{
        // lmv0_x, lmv0_y,
        // lmv1_x, lmv1_y,
        // lmv2_x, lmv2_y are the 3 motion vectors from the same field
        ix = median3(lmv0_x, lmv1_x, lmv2_x)
        iy = median3(lmv0_y, lmv1_y, lmv2_y)
            The color-difference reference field is p.
}
else
{
        // Use the 2 motion vectors from the field that has the same polarity as the current field.
        // lmv0_x, lmv0_y,
        // lmv1_x, lmv1_y   are the motion vectors that have the same polarity as the current field
        ix = (lmv0_x + lmv1_x) / 2
        iy = (lmv0_y + lmv1_y) / 2
                The color-difference reference field is the field that has the same polarity as the current field.
}
cmv_x = (ix + round[ix & 3]) >> 1
cmv_y = (iy + round[iy & 3]) >> 1
Where:
round[0] = 0, round[1] = 0, round[2] = 0 and round[3] = 1.
```

Figure 123: Color-difference MV Derivation in Two Reference Interlace Field Pictures

10.3.5.4.4.2.3 Second Stage Color-difference Rounding

The color difference rounding shall be as defined in Section 8.3.5.4.5.

10.3.5.5 Coded Block Pattern

The CBPCY syntax element in the intra and inter-coded macroblock layer indicates the transform coefficient status for each block in the macroblock. The CBPCY element decodes to a 6-bit field which indicates whether coefficients are present for the corresponding block. Table 70 defines the correspondence between the bit positions in CBPCY and the block number. For intra-coded macroblocks, a value of 0 in a particular bit position shall indicate that the corresponding block does not contain any non-zero AC coefficients. A value of 1 shall indicate that at least one non-zero AC coefficient is present. The DC coefficient shall still be present for each block in all cases. For inter-coded macroblocks, a value of 0 in a particular bit position shall indicate that the corresponding block does not contain any non-zero coefficients. A value of 1 shall indicate that at least one non-zero coefficient is present. For cases where the bit is 0, no coefficient data shall be coded for that block.

10.3.6 Block Layer Decode

10.3.6.1 Intra Coded Block Decode

The decoding process for Intra coded blocks shall be the same as described in section 8.3.6.1, except that unlike progressive P frames, interlace field P pictures shall not contain Intra blocks within 4-MV-coded macroblocks, so the section describing the decoding process for those blocks shall be ignored.

10.3.6.2 Inter Coded Block Decode

Figure 59 shows the steps for reconstructing Inter blocks. For illustration the figure shows the reconstruction of a block whose 8x8 error signal is coded with two 8x4 Transforms. The 8x8 error block can also be transformed with either two 4x8 Transforms, four 4x4 Transforms, or one 8x8 Transform. The steps required to reconstruct an inter-coded block shall be: 1) transform type selection, 2) sub-block pattern decode, 3) coefficient decode, 4) zigzag scan of coefficients, 5) inverse quantization, 6) inverse Transform, and 7) obtain predicted block via motion compensation and add predicted and error blocks. The following sections describe these steps.

10.3.6.2.1 Transform Type Selection

The transform type selection process shall be identical to the corresponding selection process for progressive pictures as described in section 8.3.6.2.1.

10.3.6.2.2 Subblock Pattern Decode

The subblock pattern decode shall be identical to the corresponding decode process for progressive pictures as described in section 8.3.6.2.2.

10.3.6.2.3 Coefficient Bitstream Decode

The process of transform coefficient decoding shall be as described in Section 8.3.6.2.3.

10.3.6.2.4 Zigzag Scan of Coefficients

The zigzag scanning of coefficients shall be the same as defined in section 8.3.6.2.5.

10.3.6.2.5 Inverse Quantization

The non-zero quantized coefficients reconstructed as described in the sections above shall be inverse quantized as described in section 8.1.3.8.

10.3.6.2.6 Inverse Transform

The inverse transform shall be identical to the corresponding process for progressive P pictures, and shall be as described in section 8.3.6.4.

10.3.6.2.7 Motion Compensation

The motion compensation process shall be the same as described in section 8.3.6.5.

10.3.7 Rounding Control

The rounding control process shall be the same as described in section 8.3.7.

10.3.8 Intensity Compensation

If the MVMODE syntax element indicates that intensity compensation information is present in the bitstream then the INTCOMPFIELD syntax element (9.1.1.48) shall be present and shall indicate which reference fields undergo intensity compensation.

If INTCOMPFIELD indicates that both reference fields undergo intensity compensation then the LUMSCALE1 (9.1.1.49), LUMSHIFT1 (9.1.1.50), LUMSCALE2 (9.1.1.51) and LUMSHIFT2 (9.1.1.52) syntax elements shall be present in the bitstream. The first two elements shall be used to control the intensity compensation of the top (even lines) reference field and the last two elements shall be used to control the intensity compensation of the bottom (odd lines) reference field. For example, if the bottom field is the second field in the frame and is the field currently being decoded and INTCOMPFIELD indicates that both reference fields are to have intensity compensation performed, then LUMSHIFT1 and LUMSCALE1 shall control intensity compensation of the top field of the current frame and LUMSHIFT2 and LUMSCALE2 shall control intensity compensation of the bottom field of the reference frame. The

process for applying LUMSHIFT and LUMSCALE to achieve intensity compensation for a field shall be the same as for progressive frames as described in section 8.3.8.

If INTCOMPFIELD indicates that only the top reference field is to undergo intensity compensation then the LUMSCALE1 and LUMSHIFT1 syntax elements shall be present in the bitstream and shall control how intensity compensation is performed for the field. For example, if the bottom field is the second field in the frame and is the field currently being decoded and INTCOMPFIELD indicates that the top reference field is to have intensity compensation performed then LUMSHIFT1 and LUMSCALE1 control intensity compensation of the top field of the current frame. If the top field is the first field in the frame and is the field currently being decoded and INTCOMPFIELD indicates that the top reference fields is to have intensity compensation performed, then LUMSHIFT1 and LUMSCALE1 control intensity compensation of the top field of the reference frame.

If INTCOMPFIELD indicates that only the bottom reference field is to undergo intensity compensation, then the LUMSCALE1 and LUMSHIFT1 syntax elements shall be present in the bitstream and shall control how intensity compensation is performed for the field. For example, if the bottom field is the second field in the frame and is the field currently being decoded and INTCOMPFIELD indicates that the bottom reference fields is to have intensity compensation performed, then LUMSHIFT1 and LUMSCALE1 control intensity compensation of the bottom field of the reference frame. If the top field is the second field in the frame and is the field currently being decoded and INTCOMPFIELD indicates that the bottom reference fields is to have intensity compensation performed, then LUMSHIFT1 and LUMSCALE1 control intensity compensation of the top field of the current frame.

If intensity compensation is performed on a reference field, then after decoding the field, the post-compensated pixel values shall be retained and shall be used when decoding the next field. If the next field indicates that the field that was intensity compensated by the previous field is to have intensity compensation performed again then the post-compensated field shall be used. Therefore, when a reference field has intensity compensation performed twice, the result of the first intensity compensation operation shall be used as input for the second intensity compensation.

The pixel replication for out-of-bound motion compensation shall be performed after one or both fields of a reference frame have been intensity compensated. The vertical replication method shall be controlled by the coding type of the reference frame. For example, if the reference frame was coded as a progressive frame then it will use the progressive vertical replication rule – namely that the samples in the top row are replicated upward and the samples in the bottom row are replicated downward. If the coding type of the reference was interlace field or interlace frame then the interlace replication rule shall be used – namely that the samples in the top line of the frame (the frame being made up of the two interlaced fields) are replicated upward every other line and the samples in the second-to-top line of the frame are replicated upward every other line. The samples in the last line are replicated downward every other line and the samples in the second-to-last last line are replicated downward every other line.

10.4 Interlace Field B Picture Decoding

The following sections define the process required to decode B interlace Field pictures. B field syntax shall be the same as P field syntax (as defined in 10.3), except as described in this section. When using B frames, both forward and backward frames are needed for motion compensation. In field mode coding, the first (B) field may be used as reference for the second B field being decoded. For example, if in a B interlace field picture, the first field to be decoded is the top field, then the bottom field of that picture shall use the top field as the reference for motion compensation. For B fields, the number of reference fields (see NUMREF) shall always be set to 2. Therefore, B fields always use 4 reference fields in all (top and bottom forward, top and bottom backward) to predict the current MB.

B field coding has the following properties:

1) The first B field shall reference the first and second fields from the previous and next anchor frames (see Figure 124).

2) The second B field shall reference the first B field from the current frame (e.g. the top B field in Figure 124) as the field of “opposite polarity” and the second field of the previous anchor frame as the field of “same polarity”, plus the first and second fields of the next anchor frame. If TFF == 1 then the first field is the top field and the second field is the bottom field. This is the case illustrated in Figure 124. If TFF == 0 then the bottom field is the first field and the top field is the second field.

3) B interlace field picture macroblocks shall use one of four prediction types: forward, backward, direct and interpolated (see section 10.4.5.1). The “forward/not forward” (0/1) decision (per MB) can be bitplane coded at the picture level.

Note: This is different from progressive B frames, where the frame level bitplane codes “direct/not direct”.

4) MV prediction shall follow the same logic as field P pictures (see section 10.3.5.4), except that separate forward and backward contexts shall be retained. The “holes” in the forward and backward prediction contexts are filled differently as compared to the progressive case. After decoding a backward motion vector, the forward buffer’s MV shall be filled with the predicted motion vector if the macroblock was coded in backward mode.

5) Only forward and backward modes shall be used with 4-MV; Interpolated and direct modes shall not be used.

6) The MB mode (comprising 1-MV/4-MV/Intra, skipped MB, CBP present, 4-MV block pattern) joint coding, MV tables and MV architecture) shall be the same as defined for P interlace field pictures (see section 10.3.5).

7) Hybrid MV prediction (see section 10.3.5.4.3.5) shall not be used in interlace field B pictures.

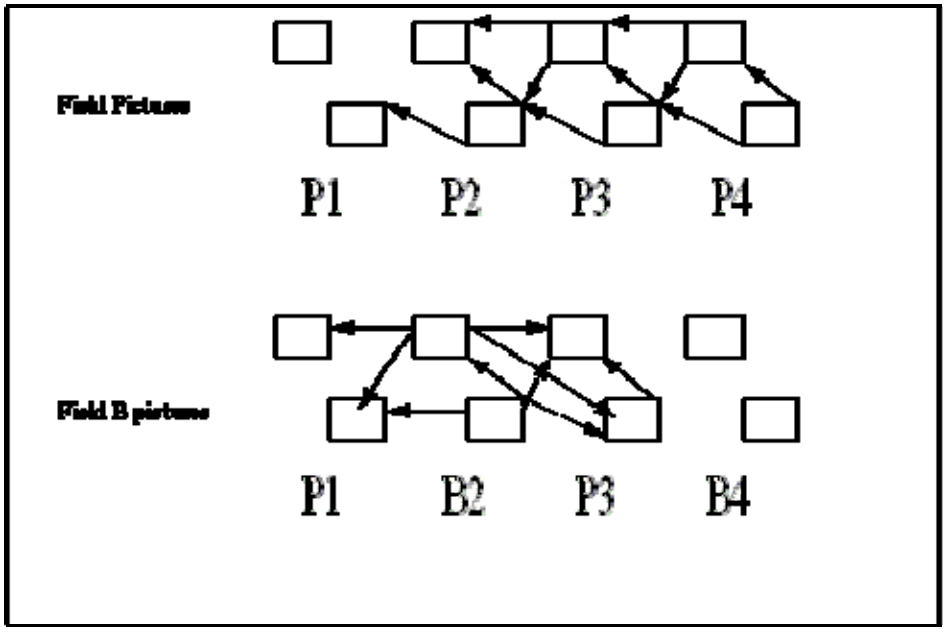


Figure 124: B field references

10.4.1 Handling of TFF

In Interlace Field Pictures the TFF syntax element in the frame header shall indicate the temporal order of the two fields that comprise the frame. TFF == 1 shall indicate that the first field in the frame is the top field (even lines of the frame) and the second field is the bottom field (odd lines of the frame). TFF == 0 shall indicate that the first field in the frame is the bottom field and the second field is the top field. The following restrictions and procedures are required for the TFF syntax element and its relationship to the reference field or fields:

1. If the current frame is coded as an Interlace Field Picture, then the backward reference frame shall be coded as an Interlace Field Picture and both shall have the same TFF value. If the forward reference frame is coded as an Interlace Field Picture then the TFF of the forward reference shall be the same as the TFF of the current frame.
2. If the current frame is coded as an Interlace Field Picture and the forward reference frame is coded as a Progressive Picture or Interlace Frame Picture, then for purposes of determining the field order of the forward reference frame, the TFF value of the reference frame shall be assumed to be the same as the TFF value of the current frame, regardless of the value of TFF for the forward reference frame.

10.4.2 Out-of-bounds Reference Pixels

Out-of-bounds reference pixels shall be the same as interlace field P pictures (see 10.3.2).

10.4.3 Reference Pictures

A B Interlace Field Picture shall only reference a total of four fields as shown in Figure 124. Therefore, the NUMREF and REFFIELD syntax elements shall not be present in B field pictures.

10.4.4 B Picture Types

B pictures shall be one of two types: 1-MV or Mixed-MV. The following sections describe each B picture type.

10.4.4.1 1-MV B Picture

In 1-MV B pictures, either zero, one or two motion vectors shall be used to indicate the displacement of the predicted (inter-coded) blocks, depending on the prediction type, BMVTYPE of that macroblock. When BMVTYPE (9.1.3.15) is equal to Direct, the forward and backward motion vectors shall be inferred and no further motion vectors shall be explicitly signaled. When the BMVTYPE is Interpolated, two motion vectors, i.e. forward and backward, shall be decoded. In the forward and backward cases, only one motion vector shall be decoded, as with 1-MV P interlace field pictures (10.3.5.1.1). The 1-MV mode shall be signaled by the MVMODE picture layer syntax elements as described in section 8.3.4.3.

Note: The MVMODE2 syntax element is not present in B interlace field pictures, as intensity compensation is not applicable.

10.4.4.2 Mixed-MV B Picture

In Mixed-MV B pictures, each inter-coded macroblock shall be encoded as either a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luma blocks has a motion vector associated with it. Additionally, 4-MV macroblocks shall only be associated with forward or backward prediction types (BMVTYPE) in B interlace field pictures. The 1-MV or 4-MV mode for each macroblock shall be indicated by the MBMODE syntax element at every macroblock. The Mixed-MV mode shall be signaled by the MVMODE picture layer syntax elements as described in section 8.3.4.3.

10.4.5 B Macroblock Layer Decode

10.4.5.1 Macroblock Types

Macroblocks in B interlace field pictures shall be one of 3 possible types: 1-MV, 4-MV, and Intra. Additionally, inter-coded macroblocks shall be one of four prediction types: forward, backward, direct or interpolated (see 9.1.3.15). The macroblock prediction type is signaled by the MBMODE syntax element in the macroblock layer. The prediction type is signaled by a combination of the frame-level bitplane FORWARDMB which signals forward/non-forward for each macroblock, and the macroblock level BMVYPTE syntax in case the prediction type is non-forward.

10.4.5.1.1 1-MV Macroblocks

1-MV macroblocks may occur in 1-MV and Mixed-MV B pictures. 1-MV macroblocks shall be signaled identically to P interlace field pictures, i.e. using the MBMODE syntax element in the macroblock layer. The CBPCY syntax element shall be also identical to P interlace field pictures, both syntactically and semantically.

If the MBMODE syntax element indicates that the MVData is present, then the BMV1 syntax element (which is analogous to MVDATA syntax element in P pictures) shall be present in the macroblock layer in the corresponding position. The BMV1 syntax element encodes the motion vector differential. The motion vector differential is combined with the motion vector predictor to reconstruct the motion vector. If the MBMODE syntax element indicates that the BMV1 syntax element is not present, then the motion vector differential shall be zero and therefore the motion vector is equal to the motion vector predictor.

Additionally in B interlace field pictures, if the macroblock type is 1-MV and the prediction type of the macroblock is decoded as INTERPOLATED, then the INTERPMVP syntax element shall be used to signal whether or not the second motion vector differential, BMV2 is present. If it is, then BMV2 shall be decoded immediately following BMV1. If BMV2 is not present, the motion vector differential for BMV2 shall be set to zero, and therefore the motion vector is equal to the motion vector predictor.

When the prediction type is interpolated, BMV1 corresponds to the forward and BMV2 corresponds to the backward MV. Note: This is unlike progressive B pictures.

10.4.5.1.2 4-MV Macroblocks

4-MV macroblocks shall only occur in Mixed-MV B pictures. They shall be treated identically to P interlace field pictures, with the additional restriction that they shall only be associated with the prediction types (BMVTYPE) of forward or backward.

10.4.5.1.3 Intra Macroblocks

These may occur in B pictures, and shall be identical to those in P interlace field pictures (see section 10.3.5.2).

10.4.5.2 Macroblock Mode

The MBMODE syntax and semantics shall be identical to P interlace field pictures (see section 10.3.5.3).

10.4.5.3 Prediction Type Decoding (BMVTYPE)

The prediction type (see 9.1.3.15) shall be decoded according to the following rules:

- If the picture level bitplane FORWARDMB indicates that a macroblock is of forward type, then the prediction type for that macroblock shall be set to forward.
- If FORWARDMB element (9.1.1.53) is coded as 'Raw', then an additional bit at the macroblock level, FORWARDBIT (9.1.3.19), shall be used to decide whether the prediction type is forward or not.
- If the prediction type is non-forward, and if the macroblock uses 4-MV, as signaled by the MBMODE syntax element (only possible in a Mixed-MV B picture), then the prediction type shall be backward, because only forward and backward types may be associated with 4-MV mode.
- Else, the BMVTYPE syntax element (9.1.3.15) shall be explicitly decoded by the VLC specified in Table 110.

10.4.5.4 Non-zero interpolated MV (INTERPMVP)

In case the prediction type is interpolated, an additional syntax element INTERPMVP (9.1.3.20) shall be present in the bitstream, and that element shall signal whether or not the second motion vector differential, BMV2 is present.

10.4.5.5 B Frame Modes

Inter-coded macroblocks in B fields shall be one of four modes: *backward*, *forward*, *direct* or *interpolated*. The forward mode shall be the same as conventional P picture prediction. In the forward mode, the macroblock references its temporally previous anchor fields. Likewise, backward mode macroblocks shall reference their temporally subsequent (in display order) anchor frame.

Direct mode and interpolated mode macroblocks use both the anchors for prediction. Since there are two reference images for these modes, there are two motion vectors for each macroblock. The direct mode implicitly derives these motion vectors by appropriately scaling and bounding the motion vectors of the co-located macroblock in the temporally subsequent (in display order) anchor frame.

The direct and interpolated modes use two motion vectors to predict from the reference (anchor) frames. Both the direct and interpolated motion modes use round-up averaging for combining the pixel values of the two interpolated references into one set of macroblock pixels:

Average pixel value = (Forward interpolated value + Backward interpolated value + 1) >> 1

The motion compensation interpolation processes (e.g. bicubic, bilinear, quarter or half pel) shall be signaled and implemented exactly the same way as with P fields (see section 10.3.6.2.7) .

10.4.5.6 Decoding Direct Mode Motion Vectors

The calculation of direct mode motion vectors is similar to the methods used with progressive B pictures (section 8.4.5.4).

The motion vectors from the previously decoded (i.e. temporally future) anchor frame (I/I, I/P, P/I or P/P interlace field picture pair) shall be used to compute the direct mode MVs for the current B field. Of these, the motion vectors corresponding to the top field shall be used to compute the direct mode motion vectors in the top B field, and those corresponding to the bottom field shall be used to compute the motion vectors of the bottom B field. For example, in the B picture, the direct mode motion vectors for MB (x, y) in field z (z = top/bottom) uses the scaled MVs from MB (x, y) of the previously decoded I or P field z (i.e. co-located MB in the field of the same polarity).

- If the co-located macroblock from the anchor picture is intra-coded, i.e. when the previously decoded z field is I, or if it is P but MB (x, y) is intra-coded, then for purposes of direct mode scaling: this motion vector shall be set to (0, 0), and the mode shall be 1-MV, and the reference field polarity shall be the same as the current field.
- If 1-MV mode was used for the co-located anchor macroblock, then that MV shall be used in the direct mode scaling calculations.
- If 4-MV mode was used for the co-located anchor macroblock, then the logic described in the pseudo-code, SelectDirectModeMVFromColocatedMB (Figure 125), shall be used to derive the MV needed for direct mode scaling calculations.

If the co-located macroblock is inter-coded, the pseudo-code of Figure 125 (SelectDirectModeMVFromColocatedMB) shall be used to pick MVs, from the co-located macroblock. The obtained motion vectors are used in the scaling operation that derives the forward and backward motion vectors.

SelectDirectModeMVFromColocatedMB considers the reference field polarities of the 4-MVs and favors the dominant polarity. Thus, if the number of MVs (out of 4) that point to a field of the same polarity as the current field outnumber those that point to a field of the opposite polarity, the median4, median3, or the arithmetic mean of 2 (integer division by 2 with truncation towards zero) shall be used if there are 4, 3, or 2 same-polarity reference field MVs respectively. Otherwise if the MVs that point to field of the opposite polarity as the current field outnumber those that point to field of the same polarity, similar operations shall be used to get a representative MV to use from the opposite-polarity reference field MVs.

```
MotionVector SelectDirectModeMVFromColocatedMB (int SingleMV, int OppFieldCount, int
SameFieldCount)
{
    MotionVector SelectedMV

    if (the co-located MB used 1-MV)
        then SelectedMV = SingleMV
    else    // 4 MVs to pick from
    {
        Count the number of same field and opposite field MVs

        if (OppFieldCount > SameFieldCount)
            then use only the opposite field MVs in next step
            // i.e. opposite is the chosen polarity
        else
            use only the same field MVs in the next step
            // i.e. same is the chosen polarity

        Count the number of MVs of the chosen polarity
        if (Number of Chosen MVs == 3)
            // MVa, MVb and MVc are the three chosen MVs
            SelectedMV = Median3 (MVa, MVb, MVc )
        else if (Number of Chosen MVs == 2)
            //MVa and MVb are the two chosen MVs
```

```
            SelectedMV = (MVa + MVb)/2    //Integer division by 2 with truncation towards zero of the chosen
MVs
        else // all 4 are of the chosen polarity
            // MVa, MVb, MVc and MVd are the four chosen MVs
            SelectedMV = Median4 (MVa, MVb, MVc, MVd)
    }

    return (SelectedMV)
}
```

Figure 125: Selection of Direct Mode MVs in Interlace Field Pictures

With the MV obtained above, the scaling logic, Scale_Direct_MV, shall then be applied as shown in the pseudo-code of Figure 126.

```
// (IN MV_X, IN MV_Y) set to (x, y) components of SelectedMV (Figure 125) if co-located MB is inter-coded
// (IN MV_X, IN MV_Y) set to (0,0) for the case when co-located MB is intra-coded.
// variables labeled "IN" are inputs to the function, and
// the variables labled "OUT" are the scaled MVs calculated by the function Scale_Direct_MV
Scale_Direct_MV (IN MV_X, IN MV_Y, OUT MV_X_F , OUT MV_Y_F, OUT MV_X_B, OUT MV_Y_B)

{
    if (Backward reference frame is half-pel MV resolution) {
        if (Current frame is half-pel MV resolution) {
            MV_X_F = (MV_X * 2 * ScaleFactor + 255) >> 9;
            MV_Y_F = (MV_Y * 2 * ScaleFactor + 255) >> 9;
            MV_X_B = (MV_X * 2 * (ScaleFactor - 256) + 255) >> 9;
            MV_Y_B = (MV_Y * 2 * (ScaleFactor - 256) + 255) >> 9;
        }
        else { // Current frame is quarter-pel MV resolution
            MV_X_F = (MV_X * 2 * ScaleFactor + 128) >> 8;
            MV_Y_F = (MV_Y * 2 * ScaleFactor + 128) >> 8;
            MV_X_B = (MV_X * 2 * (ScaleFactor - 256) + 128) >> 8;
            MV_Y_B = (MV_Y * 2 * (ScaleFactor - 256) + 128) >> 8;
        }
    }
    else {   // Backward reference frame is quarter-pel MV resolution
        if (Current frame is half-pel MV resolution) {
            MV_X_F = (MV_X * ScaleFactor + 255) >> 9;
            MV_Y_F = (MV_Y * ScaleFactor + 255) >> 9;
            MV_X_B = (MV_X * (ScaleFactor - 256) + 255) >> 9;
            MV_Y_B = (MV_Y * (ScaleFactor - 256) + 255) >> 9;
        }
        else { // Current frame is quarter-pel MV resolution
            MV_X_F = (MV_X * ScaleFactor + 128) >> 8;
            MV_Y_F = (MV_Y * ScaleFactor + 128) >> 8;
            MV_X_B = (MV_X * (ScaleFactor - 256) + 128) >> 8;
```

```
            MV_Y_B = (MV_Y * (ScaleFactor - 256) + 128) >> 8;
        }
    }
}
```

Figure 126: Scaling of Direct Mode MVs in Interlace Field Pictures

In the pseudo-code of Figure 126, ScaleFactor shall be computed in exactly the same way as described in section 8.4.5.4. Scale_Direct_MV uses the same logic as was used for progressive B pictures in section 8.4.5.4 (but without pull-back) to obtain the forward and backward pointing MVs.

The polarity of the reference fields used for the forward and backward predictors in direct mode shall be the same as the reference polarity field used by the co-located macroblock in the reference anchor field. If the co-located macroblock was coded as intra, then the reference field used is of the same polarity as the field being decoded.

Examples:

1. If the current B field is a top field and the co-located macroblock in the reference anchor field referenced the bottom field then its reference polarity is opposite. Therefore, the forward and backward predictors reference the bottom field in the forward and backward reference fields.
2. If the current B field is a bottom field and the co-located macroblock in the reference anchor field referenced the bottom field then its reference polarity is same. Therefore, the forward and backward predictors reference the bottom field in the forward and backward reference fields.
3. If the current B field is a bottom field and the co-located macroblock in the reference anchor field is coded as intra then its reference polarity is same. Therefore, the forward and backward predictors reference the bottom field in the forward and backward reference fields.

There shall be no computation of direct mode MVs for macroblocks (MB's) that are not using the direct mode prediction (e.g. forward or backward), unlike for progressive B frame.

10.4.5.7 Motion Vector Decoding Process

The following sections describe the motion vector decoding process for B interlace field picture macroblocks. This is very similar to the motion vector decoding process used for P interlace field pictures (10.3.5.4) with the following additional information.

10.4.5.7.1 Interlace Field Picture Coordinate System

This shall be identical to P interlace field pictures as defined in 10.3.5.4.1. Likewise, the motion vector resolution is either half-pel or quarter-pel depending on the MVMODE.

10.4.5.7.2 Decoding Motion Vector Differential

This shall be identical to P interlace field pictures as defined in 10.3.5.4.2. BMV1 and BMV2 syntax elements shall be used instead of MVDATA in the 1-MV case, but they are treated exactly the same way. Also, when the prediction type (BMVTYPE) is interpolated, BMV1 shall correspond to the forward and BMV2 shall correspond to the backward motion vector residual.

10.4.6 MV Prediction in B fields

Motion compensation shall be performed in both the forward and backward directions for B fields. The following sub-sections define how to perform motion vector prediction in B fields in the forward and backward directions. Separate prediction contexts are used for forward and backward mode MVs. The forward prediction context shall be used to predict forward MVs and the backward prediction context shall be used for the prediction of backward MVs.

10.4.6.1 Populating the forward and backward prediction contexts

The forward and backward motion vector contexts shall be used to predict forward and backward motion vectors respectively. For Interpolated macroblocks, the forward prediction buffer shall be used to predict the forward MV, i.e. BMV1, and the backward buffer to predict the backward MV, i.e. BMV2. When the MB is of type Direct or

Interpolated, then the forward MV component shall be buffered in the forward buffer and the backward MV component shall be buffered in the backward buffer. The actual prediction logic in each case shall be identical to that described for the two reference P field case in section 10.3.5.4.3.

When the MB is of type Forward, the forward MV shall be buffered after it is decoded in the forward prediction buffer. Then the backward component is predicted from the backward buffer and used to fill in the corresponding position in the backward buffer.

Note: This differs from the progressive case, where the backward component of the direct mode (MV_X_B, MV_Y_B) is used.

When the MB is of type Backward, the backward MV shall be buffered after it is decoded in the backward prediction buffer. Then the forward component is predicted from the forward buffer and used to fill in the corresponding position in the forward buffer.

Note: This differs from the progressive case, where the forward component of the direct mode (MV_X_F, MV_Y_F) is used.

This procedure is illustrated with an example. Let MB:(12,13) be decoded and it is forward predicted. The forward MV is computed and is inserted in the buffer of forward MVs. In the backward MV buffer, the MV prediction logic is used to fill in the position (12, 13).

When filling a buffer with the predicted motion vector, the MV type shall be set to 1-MV prior to the prediction process and the motion vector shall be set to be the dominant MV predictor.

For intra coded macroblocks, the “intra motion vector” shall be used to fill in both forward and backward motion prediction planes.

Note: Any consistent representation of “intra motion vector” can be chosen by the decoder implementation. e.g. If the MVs are being stored in a 2-byte short array, then the “intra motion vector” could be represented as a unique large constant that is filled into the MV array to indicate that the MB was coded as intra.

10.4.6.2 Forward MV Prediction in B fields

Forward MV prediction for both B fields shall be identical to P field MV prediction as defined in section 10.3.5.4.3, except for the method of deriving the reference frame distance. The forward reference frame distance shall be computed from the BFRACTION syntax element in the B interlace field picture header and from the REFDIST syntax element in the backward reference frame header. The forward reference frame distance shall be computed as:

Forward Reference Frame Distance (FRFD) = ((Numerator * FrameReciprocal * REFDIST) >> 8)

The Numerator and FrameReciprocal values shall be derived from the value of BFRACTION as defined in section 8.4.5.4. The REFDIST value used above shall be derived from the corresponding syntax element in the backward reference frame header as defined in section 9.1.1.43.

10.4.6.3 Backward MV Prediction in B fields

Backward MV prediction for the second B field in the frame shall be identical to P field MV prediction as defined in section 10.3.5.4.3.

Backward MV prediction for the first B field in the frame shall be identical to P field MV prediction with exception that the MV scaling is different to that defined in section 10.3.5.4.3.4 and Figure 119. For this case, scaleforopposite_x, scaleforopposite_y, scaleforsame_x and scaleforsame_y shall be defined in the pseudo-code of Figure 127.

```
scaleforopposite_x (n)   {
    int scaledvalue
    if (abs (n) > 255)
        scaledvalue = n
    else {
        if (abs (n) < SCALEZONE1_X)
            scaledvalue = (n * SCALEOPP1) >> 8
```

```
        else {
            if (n < 0)
                scaledvalue = ((n * SCALEOPP2) >> 8) – ZONE1OFFSET_X
            else
                scaledvalue = ((n * SCALEOPP2) >> 8) + ZONE1OFFSET_X
        }
    }
    if (scaledvalue > range_x – 1)
        scaledvalue = range_x – 1
    if (scaledvalue _x < -range_x)
        scaledvalue = -range_x

    return scaledvalue
}
scaleforopposite_y (n)   {
    int scaledvalue
    if abs (n) > 63)
        scaledvalue = n
    else {
        if (abs (n) < SCALEZONE1_Y)
            scaledvalue = (n * SCALEOPP1) >> 8
        else {
            if (n < 0)
                scaledvalue = ((n * SCALEOPP2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = ((n * SCALEOPP2) >> 8) + ZONE1OFFSET_Y
        }
    }
    if (current field is bottom field and reference field is top field) {
        if (scaledvalue > range_y / 2)
            scaledvalue = range_y / 2
        if (scaledvalue < -(range_y / 2) + 1)
            scaledvalue = -(range_y / 2) + 1
    }
    else {
        if (scaledvalue > (range_y / 2) – 1)
            scaledvalue = (range_y / 2) – 1
        if (scaledvalue < -(range_y / 2)
            scaledvalue = -(range_y / 2)
    }

    return scaledvalue
}

scaleforsame_x (n)   {
    int scaledvalue
    scaledvalue = (n * SCALESAME) >> 8
```

```
    return scaledvalue
}

scaleforsame_y (n)   {
    int scaledvalue
    scaledvalue = ((n * SCALESAME) >> 8)
    return scaledvalue
}
```

Figure 127: Backward MV Predictor Scaling for the First Field in Interlace Field B Pictures

The values range_x and range_y depend on MVRANGE and shall be specified as in Table 75.

For the case where the current field is the first field, the values of SCALESAME, SCALEOPP1, SCALEOPP2, SCALEZONE1_X, SCALEZONE1_Y, ZONE1OFFSET_X and ZONE1OFFSET_Y shall be as defined in Table 115.

The backward reference frame distance (BRFD) shall be computed from the REFDIST syntax element in the backward reference frame header and from the forward reference frame distance (FRFD). The backward reference frame distance shall be computed as:

Backward Reference Frame Distance (BRFD) = REFDIST – FRFD - 1

If BRFD < 0 then BRFD = 0.

The REFDIST syntax element shall be decoded according to section 9.1.1.43 and the forward reference frame distance (FRFD) shall be computed as described in section 10.4.6.2.

Table 115: B Interlace Field Picture Backward MV Predictor Scaling Values for when Current Field is First

	Backward Reference Frame Distance (BRFD)			
	0	1	2	3 or greater
SCALESAME	171	205	219	228
SCALEOPP1	384	320	299	288
SCALEOPP2	230	239	244	246
SCALEZONE1_X	43	51	55	57
SCALEZONE1_Y	11	13	14	14
ZONE1OFFSET_X	26	17	12	10
ZONE1OFFSET_Y	7	4	3	3

10.4.6.3.1 Hybrid Motion Vectors

Hybrid motion vectors shall not be allowed in B interlace field pictures.

10.4.6.3.2 Reconstructing Motion Vectors

Reconstructing motion vectors shall be identical to those of a P interlace field picture, for both luma and color-difference motion vector computation as defined in section 10.3.5.4.4.

10.4.6.4 Coded Block Pattern

The CBPCY syntax element shall be identical in syntax and semantics to that used with P interlace field pictures as defined in section 10.3.5.5.

10.4.7 Block Layer Decode

The block layer decoding process for B interlace field pictures, except for motion compensation which involves bi-directional prediction, shall be identical to that used in P interlace field pictures as defined in section 10.3.6.

10.5 Interlace Frame I Picture Decoding

The following sub-sections define the process for decoding interlace frame I pictures.

10.5.1 Macroblock Layer Decode

Figure 101 shows the elements that make up the intra MB layer. Each macroblock shall be either frame or field coded as indicated by FIELDTX (9.1.1.18, 9.1.3.1) which indicates the internal organization of a macroblock. FIELDTX == 1 indicates that the macroblock is field coded and FIELDTX == 0 indicates that the macroblock is frame coded. For frame coded macroblocks, the luma blocks shall be interlaced with each field occurring alternatively. For field coded macroblocks, before the permutation the top two luma blocks shall contain only the lines from top field while the bottom two luma blocks shall contain only lines from the bottom field. After the permutation, the luma blocks shall be interlaced with each lines occurring alternatively. The C_b/C_r blocks shall always remain interlaced (i.e. containing alternating lines from each field) for both field coded and frame coded macroblocks.

10.5.2 Block Decode

This section describes the process used to reconstruct the blocks which is very similar to advanced profile progressive I picture's block decoding as defined in section 8.1.3. Figure 128 shows the process used to reconstruct the 8x8 blocks.

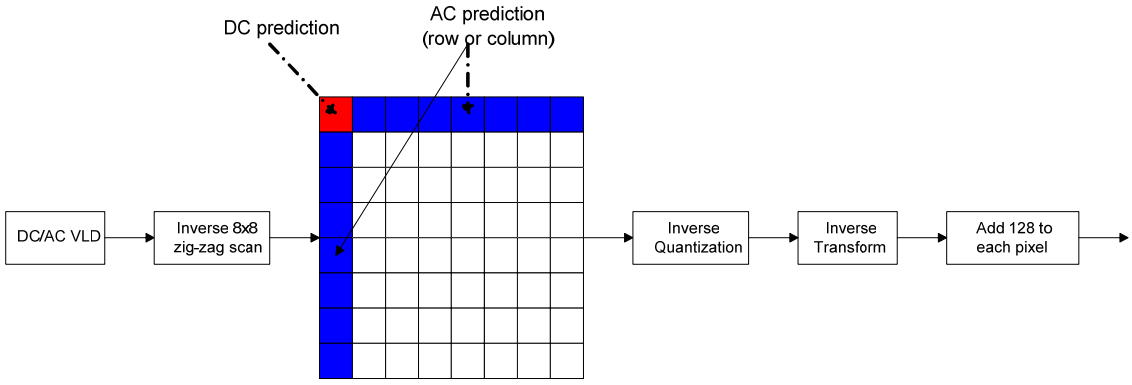


Figure 128: Intra Block Decode

The DC coefficients shall be coded differentially using neighboring block's DC coefficients as defined in section 8.1.3. The quantized DC value for the current block shall be obtained by adding the DC predictor to the DC differential. The process of DC inverse quantization and DC differential decoding shall be the same as advanced profile I picture.

The ACPRED flag (9.1.1.19, 9.1.3.3) for each macroblock shall indicate whether some of AC coefficients are coded differentially. If the AC coefficients are differentially coded, then the AC coefficients for the current block shall be obtained by adding the AC predictor (either the quantized AC coefficients of the first row of the top block or the first column of the left block) to the AC differential. The process of decoding AC (possibly differential) coefficient coding shall be the same as advanced profile progressive I picture with the exception of the selection of the zigzag scan pattern and the prediction direction.

The zigzag scan used for each block shall be selected as follows:

- If ACPRED is FALSE, then each block shall use the Intra Mode 8x8 scan shown in Table 242.
- If ACPRED is TRUE, then for each block, the prediction direction shall be computed using the algorithm described in Figure 58 and
    - the Intra Mode 8x8 scan shall be used when use_ac_prediction is FALSE,
    - the Intra Horizontal Scan (Table 234) shall be used if use_ac_prediction is TRUE and the prediction direction is TOP, and
    - the Intra Vertical Scan (Table 235) shall be used if use_ac_prediction is TRUE and the prediction direction is LEFT.

After reconstruction of the TRANSFORM coefficients, the resulting 8 × 8 blocks shall be processed by a separable two-dimensional inverse transform of size 8 by 8 as defined in section 8.1.3.10. The inverse transform output has a dynamic range of 10 bits. Subsequent to the inverse transform, the process of overlap smoothing shall be carried out if signaled. Finally, the constant value of 128 shall be added to the reconstructed and possibly overlap smoothed intra block. This result shall be clamped to the range [0 255] and forms the reconstruction prior to loop filtering. In addition, the decoded luma blocks shall be permuted if the current macroblock is field coded. The decoder color-difference blocks shall not be permuted.

10.5.2.1 DC Predictor

The DC predictor shall be obtained from one of the previously decoded adjacent blocks. Figure 38 shows the current block and the candidate predictors from the adjacent blocks. The values A, B and C represent the quantized DC values for the top, top-left and left adjacent blocks respectively. The FIELDTX flag shall not affect the DC/AC prediction process. For example, the adjacent blocks for block 0 of the current macroblock are always the block 3, block 2, block 1 of the top-left, top, and left macroblocks, respectively.

The adjacent blocks A, B, C are considered missing if they are outside the picture boundary or if the blocks are not intra coded (the last provision is for intra blocks in Interlace frame P or B pictures only). As defined in section 7.1.2, with respect to prediction, the first row of macroblocks in the slice shall be considered to be the first row of macroblocks in the picture.

The prediction direction shall be calculated the same way as shown in Figure 58.

In addition, the DC predictor shall be scaled in the same way as advanced profile progressive I picture (section 8.1.3.9) if the MQUANT of the predictor blocks are different from that of the current block.

10.5.2.2 AC Prediction

If AC prediction is turned on for the current block, then the AC coefficients on either the top row or the left column can be differentially encoded. The decision for the prediction direction is based on the DC predictor. There are three cases, DC is predicted from the left block, the top block, or not predicted.

- If DC is predicted from the top block, then the top row of the current block shall be differentially coded.
- If DC is predicted from the left block, then the left column of the current block shall be differentially coded.
- If DC is not predicted, then the AC coefficients shall not be differentially coded.

The AC coefficients in the predicted row or column shall be added to the corresponding decoded AC coefficients in the current block to produce the fully reconstructed quantized Transform coefficient block. In addition, if the macroblock quantizers of the predictor blocks are different from that of the current block, the AC predictor shall be scaled in the same way as advanced profile progressive I picture (section 8.1.3.9). As in progressive pictures, the product $DC_p * DCSTEP_p$, and the product $\overline{AC}_p * STEP_p$ product shall not exceed the signed 12 bit range, i.e., these product values shall be limited to ≥ -2048 && ≤ 2047.

10.6 Interlace BI Frame Decoding

Interlace BI Frames are B frames where all macroblocks are intra coded. The syntax and decoding processes of Interlace BI frames shall be identical to that of I, but Interlace BI Frames shall not be used as an anchor or reference frame to predict other frames.

10.7 Interlace Frame P Picture Decoding

The following sub-sections define the process for decoding interlace frame P pictures.

10.7.1 Skipped Frames

In the advanced profile, a skipped frame shall be signaled by the PTYPE syntax element in the picture header. If a frame is signaled as skipped, then it shall be treated as if it were a P frame which was identical to the reference frame.

Therefore, the reconstruction of the skipped frame can be treated conceptually as copying the reference frame pixel data and setting the buffered motion vectors (for subsequent direct mode vector computation) to (0,0).

10.7.2 Out-of-bounds Reference Pixels

The previously decoded anchor frame shall be used as the reference for motion-compensated predictive coding of the current frame P picture. The motion vectors used to locate the predicted blocks in the reference frame can include pixel locations that are outside the boundary of the reference frame. In this case the boundary pixels shall be replicated to form out-of-bounds reference pixels as described in section 8.3.2.

10.7.3 Macroblock Layer Decode

In interlace frame P pictures, each inter-coded macroblock shall be motion compensated in frame mode using either 1 or 4 motion vector(s), or in field mode using either 2 or 4 motion vectors. Frame motion compensation treats a macroblock as a whole entity while field motion compensation treats a macroblock as composed of two separate fields. A macroblock that is inter-coded shall not contain any intra blocks. In addition, the residual after motion compensation may be coded in frame transform mode or field transform mode, where these modes are the same as the modes for interlace frame I picture defined in section 10.5.1. More specifically, the luma component of the residual data values shall be re-arranged according to fields if it is coded in field transform mode and shall remain unchanged if it is coded in frame transform mode. The color-difference component shall remain the same (i.e., without re-arrangement) in both cases.

A macroblock may also be coded as intra. In this case, the decoding process shall be the same as I macroblock decoding in interlace frame I picture as defined in section 10.5.1.

The motion compensation may be restricted to not include 4 (both field/frame) motion vectors and this is signaled through 4MVSWITCH (9.1.1.28). The type of motion compensation / residual coding shall be jointly indicated for each macroblock through MBMODE and SKIPMB. MBMODE employs different set of tables according to 4MVSWITCH. The motion vectors shall be in quarter pixel units. The luma interpolation for deriving subpixel motion shall be bicubic and the color-difference interpolation for deriving subpixel motion shall be bilinear as defined in sections 8.3.6.5.2 and 8.3.6.5.1.

Macroblocks in interlace frame P pictures shall be one of 5 types: 1-MV, 2 Field MV, 4 Frame MV, 4 Field MV, and Intra. The first four types of macroblock shall be inter-coded while the last type shall be intra-coded. The macroblock type shall be signaled by the MBMODE syntax element in the macroblock layer along with the skip bit. MBMODE shall jointly encode macroblock types along with various pieces of information regarding the macroblock for different types of macroblock.

10.7.3.1 Inter Macroblock Types

The following sub-sections define four types of motion compensation:

10.7.3.1.1 1-MV Macroblock

In 1-MV macroblocks, the displacement of the four luma blocks shall be represented by a single motion vector. A corresponding color-difference motion vector shall be derived to represent the displacements of each of the two 8x8 color-difference blocks.

Note: 1-MV macroblocks can also be called 1 Frame MV macroblocks as they are motion compensated in frame mode.

10.7.3.1.2 2 Field MV Macroblock

"2 Field MV" is a macroblock with two motion vectors where one motion vector describes the displacement of the top field of the MB, and the other motion vector describes the displacement of the bottom field of the MB. In 2 Field MV macroblocks, the displacement of each field of the luma blocks shall be defined by a different motion vector (see Figure 129). The top field motion vector shall describe the displacement of the even lines of the luma blocks while the bottom field motion vector shall describe the displacement of the odd lines of the luma blocks. Using the top field motion vector, a corresponding top field color-difference motion vector shall be derived that describes the displacement of the even lines of the color-difference blocks. Similarly, a bottom field color-difference motion vector shall be derived from the bottom field motion vector that describes the displacements of the odd lines of the color-difference blocks.

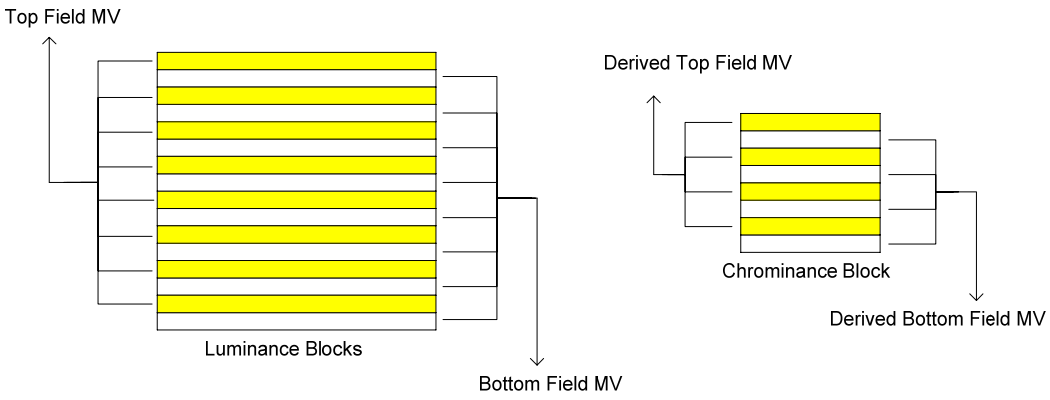


Figure 129: Two Field MV Macroblock

10.7.3.1.3 4 Frame MV Macroblock

"4 Frame MV" is a macroblock with four motion vectors where each of the motion vectors describes the displacement of one of the four luma blocks in the MB. In 4 Frame MV macroblocks, each one of the four luma block's displacement shall be defined by a different motion vector (see Figure 130). Similarly, each color-difference block shall be motion compensated using four derived color-difference motion vector that describes the displacement of the four 4x4 subblocks. The color-difference motion vector of each 4x4 subblock shall be derived as described in section 10.7.3.7, from the motion vectors of the spatially co-located luma block.

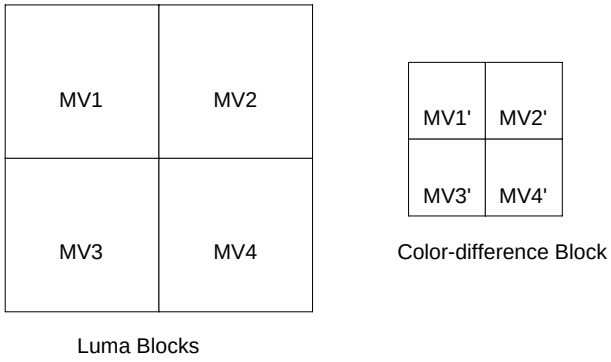


Figure 130: 4 Frame MV Macroblock

10.7.3.1.4 4 Field MV Macroblock

"4 Field MV" is a macroblock with four motion vectors, where two motion vectors describe the displacement of the top field of the MB, and the other two motion vectors describe the displacement of the bottom field of the MB. In 4 Field MV macroblocks, the displacement of each field in the luma blocks shall be defined by two different motion vectors (see Figure 131). The even lines of the luma blocks are subdivided vertically to form two eight by eight regions. The displacement of the left region shall be described by the top left field block motion vector and the displacement of the right region shall be described by the top right field block motion vector. Similarly, the odd lines in the luma blocks are subdivided vertically to form two eight by eight regions. The displacement of the left region shall be described by the bottom left field block motion vector and the displacement of the right region shall be described by the bottom right field block motion vector. Similarly, each color-difference block is partitioned into four regions in the same way as the luma blocks and each region shall be motion compensated using a derived field color-difference motion vector.

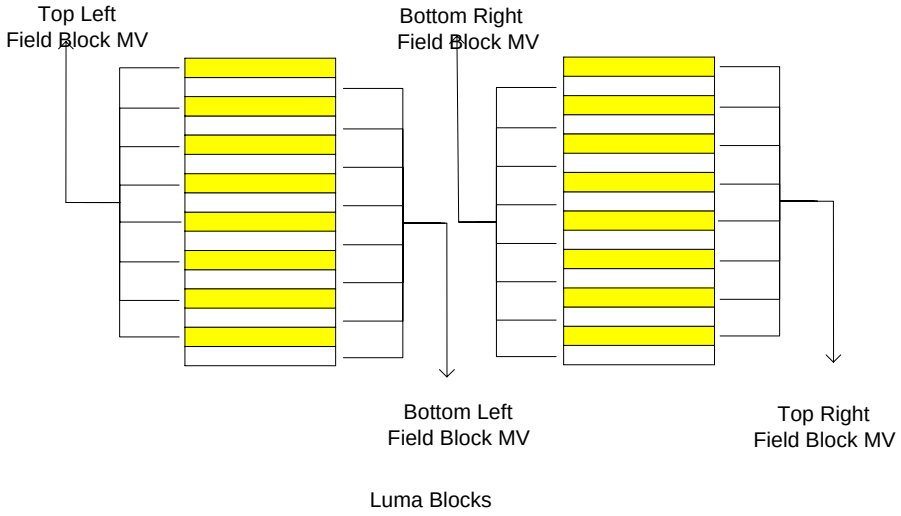


Figure 131: 4 Field MV Macroblock – Luma Block

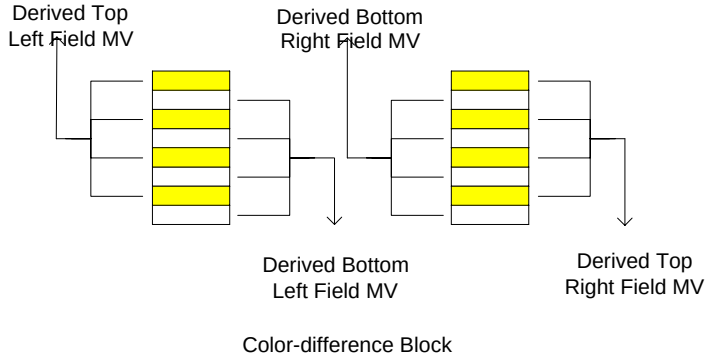


Figure 132: 4 Field MV Macroblock – Color-difference Block

10.7.3.2 4MVBP and 2MVBP

For 4 frame MV macroblocks, the 4MVBP shall be present to signal whether each of the four frame block motion vectors exist. The ordering shall be as follows:

Bit 3 = top left frame mv.

Bit 2 = top right frame mv.

Bit 1 = bottom left frame mv.

Bit 0 = bottom right frame mv.

For 4 field MV macroblocks, the 4MVBP shall be present to signal whether each of the four field block motion vectors exist. The ordering shall be as follows:

Bit 3 = top left field mv.

Bit 2 = top right field mv.

Bit 1 = bottom left field mv.

Bit 0 = bottom right field mv.

For 2 field MV macroblocks, the 2MVBP shall be present to signal whether each of the 2 field motion vectors exist. The ordering shall be as follows:

Bit 1 = top field mv.

Bit 0 = bottom field mv.

10.7.3.3 Skipped Macroblock Signaling

The SKIPMB syntax element (9.1.1.32) shall indicate the skip condition for a macroblock.

If SKIPMB == 1, then the current macroblock shall be skipped and there shall be no other information sent after the SKIPMB field. The skip condition implies that the current macroblock is 1-MV with zero differential motion vector (i.e. the macroblock is motion compensated using its 1-MV motion predictor) and there are no coded blocks (CBP == 0).

If the SKIPMB field is not 1, then the MBMODE field shall be decoded to indicate the type of macroblock, types of transform for an inter-coded macroblock, and the presence of differential motion vector for the 1-MV macroblock, as defined in section 10.7.3.4.

10.7.3.4 Macroblock Mode Signaling

MBMODE jointly specifies

a) the type of macroblock: either one of the 4 inter macroblock types (MVTYPEMB): 1-MV, 4 Frame MV, 2 Field MV, 4 Field MV, or the Intra macroblock type,
b) types of transform for inter-coded macroblock: one of field or frame or zero coded blocks (i.e. CBP == 0), and
c) whether there is differential motion vector for the 1-MV macroblock.

MBMODE shall take one of 15 possible values as defined below.

Let <MVP> denote the signaling of whether the nonzero 1-MV differential motion vector is present or absent in a 1-MV macroblock.

Let <Field/Frame transform> denote the signaling of whether the residual of the macroblock is a) frame transform coded, b) field transform coded, or c) zero coded blocks (i.e. CBP == 0).

Then the MBMODE shall signal the following information jointly:

MBMODE = { <1-MV, MVP, Field/Frame transform>, <2 Field MV, Field/Frame transform>, <4 Frame MV, Field/Frame transform>, <4 Field MV, Field/Frame transform>, <Intra>};

The case where <1-MV, MVP=0, CBP=0> is not signaled by MBMODE, but shall be signaled by the skip condition.

For inter-coded macroblocks, the CBPCY syntax element shall not be decoded when the <Field/Frame Transform> in MBMODE indicates no coded blocks.

If the <Field/Frame transform> in MBMODE indicates field or frame transform, then CBPCY shall be decoded. The decoded <Field/frame Transform> is used to set the flag FIELDTX. If it indicates that the macroblock is field transform coded, FIELDTX shall be set to one. If it indicates that the macroblock is frame transform coded, FIELDTX shall be set to zero. If it indicates a zero-coded block, FIELDTX shall be set to the same type as the motion vector, i.e., FIELDTX is set to 1 if the motion vector is a FIELD MV (2 Field MV or 4 Field MV) and set to 0 if the motion vector is a FRAME MV (1-MV or 4 Frame MV).

For non 1-MV inter-coded macroblocks, an additional field is sent to indicate which of the differential motion vectors is non-zero. In the case of 2 Field MV macroblocks, the 2MVBP field shall be sent to indicate which of the two motion vectors contain nonzero differential motion vectors. Similarly, the 4MVBP field shall be sent to indicate which of the four motion vectors contain nonzero differential motion vectors.

For intra-coded macroblocks, the Field/Frame transform and zero coded blocks shall be coded in separate fields.

10.7.3.5 Motion Vector Predictors

The process of computing the motion vector predictor(s) for the current macroblock consists of two steps:

First, three candidate motion vectors for the current macroblock are gathered from its neighboring macroblocks. Figure 133 shows the neighboring macroblock from which the candidate motion vectors are selected. The order of the collection of candidate motion vectors is important and it shall start from A, to B, and ends at C. A predictor candidate shall be considered non-existent if either the corresponding block is outside the frame boundary, or if the corresponding block is part of a different slice. Thus, motion vector prediction shall not be performed across slice boundaries.

Second, the motion vector predictor(s) for the current macroblock shall be computed from the set of candidate motion vectors.

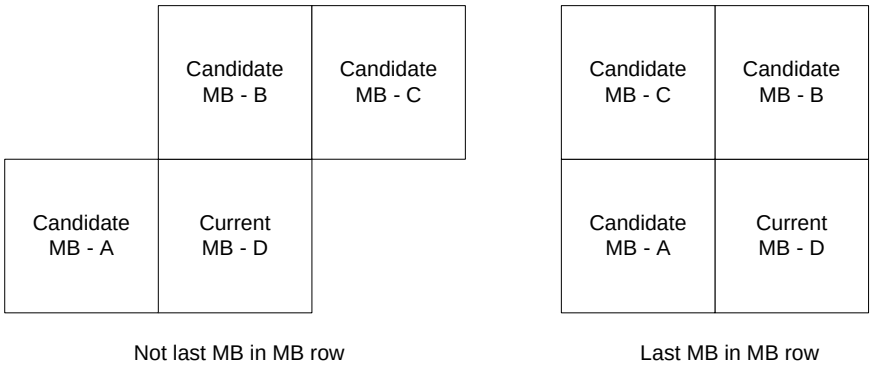


Figure 133: Candidate (Spatial) Neighboring Macroblocks for Interlace Frame Picture

The following sections describe how the candidate motion vectors are collected for different types of macroblock and how the motion vector predictor(s) is computed. For Figure 134 through Figure 144, the values "A", "B" and "C" are the candidate motion vectors defined in 10.7.3.5.

10.7.3.5.1 1-MV Candidate Motion Vectors Derivation

The pseudo-code of Figure 134 shall be used to collect the (possibly) three candidate motion vectors for 1-MV:

```
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of candidate motion vectors.
    } else if (A is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of A and add the
        resulting MV to the set of candidate motion vectors.
    } else if (A is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
        block field MV of A and add the resulting MV to the set of candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of candidate motion vectors.
```

```
    } else if (B is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vectors.
    } else if (B is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top left block field MV and bottom left
        block field MV of B and add the resulting MV to the set of candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of C and add the
        resulting MV to the set of candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Average (as defined in section 10.7.3.5.5) the top left block field MV and bottom left
            block field MV of C and add the resulting MV to the
            set of candidate motion vectors.
        } else { // C is top left MB
            Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
            block field MV of C and add the resulting MV to the
            set of candidate motion vectors.
        }
    }
}
```

Figure 134: Candidate Motion Vector Derivation for '1-MV' in Interlace Frame

10.7.3.5.2 4 Frame MV Candidate Motion Vectors Derivation

In this case, the candidate motion vectors from the neighboring blocks, for each of the four frame block motion vectors in the current macroblock, shall be collected.

The pseudo-code of Figure 135 shall be used to collect the (possibly) three candidate motion vectors for the top left frame block MV:

```
// Top Left Block MV
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
```

```
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of A and add the
        resulting MV to the set of candidate motion vectors.
    } else if (A is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
        block field MV of A and add the resulting MV to the set
       of candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vectors.
    } else if (B is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top left block field MV and bottom left
        block field MV of B and add the resulting MV to the set
  of candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of C and add the
        resulting MV to the set of candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Average (as defined in section 10.7.3.5.5) the top left block field MV and bottom left
            block field MV of C and add the resulting MV to the
```

```
            set of candidate motion vectors.
        } else { // C is top left MB
            Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
            block field MV of C and add the resulting MV to the
            set of candidate motion vectors.
        }
    }
}
```

Figure 135: Candidate MV Derivation for Top Left block in '4 Frame MV' in Interlace Frame

The pseudo-code of Figure 136 shall be used to collect the (possibly) three candidate motion vectors for the top right frame block MV:

```
// Top Right Block MV
Add the top left block MV of the current MB to the set of
candidate motion vectors.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom right block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vectors.
    } else if (B is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
        block field MV of B and add the resulting MV to the set
            of candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of C and add the
```

```
            resulting MV to the set of candidate motion vectors.
        } else if (C is 4 Field MV) {
            if (C is top right MB) {
                Average (as defined in section 10.7.3.5.5) the top left block field MV and bottom left
                block field MV of C and add the resulting MV to the
                set of candidate motion vectors.
            } else { // C is top left MB
                Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
                block field MV of C and add the resulting MV to the
                set of candidate motion vectors.
            }
        }
}
```

Figure 136: Candidate MV Derivation for Top Right block in '4 Frame MV' in Interlace Frame

The pseudo-code of Figure 137 shall be used to collect the (possibly) three candidate motion vectors for the bottom left frame block MV:

```
// Bottom Left Block MV
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the bottom right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Average (as defined in section 10.7.3.5.5) the two field motion vectors of A and add the
        resulting MV to the set of candidate motion vectors.
    } else if (A is 4 Field MV) {
        Average (as defined in section 10.7.3.5.5) the top right block field MV and bottom right
        block field MV of A and add the resulting MV to the set
            of candidate motion vectors.
    }
}

Add the top left block MV of the current MB to the set of
candidate motion vectors.

Add the top right block MV of the current MB to the set of candidate motion vectors.
```

Figure 137: Candidate MV Derivation for Bottom Left block in '4 Frame MV' in Interlace Frame

The pseudo-code of Figure 138 shall be used to collect the three candidate motion vectors for the bottom right frame block MV:

```
// Bottom Right Block MV
Add the bottom left block MV of the current MB to the set of candidate motion vectors.
```

Add the top left block MV of the current MB to the set of candidate motion vectors.

Add the top right block MV of the current MB to the set of candidate motion vectors.

Figure 138: Candidate MV Derivation for Bottom Left block in '4 Frame MV' in Interlace Frame

10.7.3.5.3 2 Field MV Candidate Motion Vectors Derivation

In this case, the candidate motion vectors from the neighboring blocks, for each of the two field motion vectors in the current macroblock, shall be collected.

The pseudo-code of Figure 139 shall be used to collect the (possibly) three candidate motion vectors for the top field MV:

```
// Top Field MV
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Add the top field MV of A to the set of candidate motion
        vectors.
    } else if (A is 4 Field MV) {
        Add the top right field block MV of A to the set of
        candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of candidate motion
        vectors.
    } else if (B is 4 Field MV) {
        Add the top left field block MV of B to the set of
        candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
```

```
            Add MV of C to the set of candidate motion vector.
        } else if (C is 4 Frame MV) {
            if (C is top right MB) {
                Add the bottom left block MV of C to the set of
                candidate motion vectors.
            } else { // C is top left MB
                Add the bottom right block MV of C to the set of
                candidate motion vectors.
            }
        } else if (C is 2 Field MV) {
            Add the top field MV of C to the set of candidate motion
            vector.
        } else if (C is 4 Field MV) {
            if (C is top right MB) {
                Add the top left field block MV of C to the set of
                candidate motion vectors.
            } else { // C is top left MB
                Add the top right field block MV of C to the set of
                candidate motion vectors.
            }
        }
}
```

Figure 139: Candidate MV Derivation for Top Field MV in '2 Field MV' in Interlace Frame

The pseudo-code of Figure 140 shall be used to collect the (possibly) three candidate motion vectors for the bottom field MV:

```
// Bottom Field MV
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the bottom right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Add the bottom field MV of A to the set of candidate
       motion vectors.
    } else if (A is 4 Field MV) {
        Add the bottom right field block MV of A to the set of
        candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
```

```
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the bottom field MV of B to the set of candidate
  motion vectors.
    } else if (B is 4 Field MV) {
        Add the bottom left field block MV of B to the set of
        candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
         Add the bottom field MV of C to the set of
         candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Add the bottom left field block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right field block MV of C to the set
            of candidate motion vectors.
        }
    }
}
```

Figure 140: Candidate MV Derivation for Bottom Field MV in '2 Field MV' in Interlace Frame

10.7.3.5.4 4 Field MV Candidate Motion Vectors Derivation

In this case, the candidate motion vectors from the neighboring blocks, for each of the four field blocks in the current macroblock, shall be collected.

The pseudo-code of Figure 141 shall be used to collect the (possibly) three candidate motion vectors for the top left field block MV:

```
// Top Left Field Block MV
```

```
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Add the top field MV of A to the set of
        candidate motion vectors.
    } else if (A is 4 Field MV) {
        Add the top right field block MV of A to the set of
        candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of
        candidate motion vectors.
    } else if (B is 4 Field MV) {
        Add the top left field block MV of B to the set of
        candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Add the top field MV of C to the set of
        candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Add the top left field block MV of C to the set of
```

```
            candidate motion vectors.
        } else { // C is top left MB
            Add the top right field block MV of C to the set of
            candidate motion vectors.
        }
    }
}
```

Figure 141: Candidate MV Derivation for Top Left MV in '4 Field MV' in Interlace Frame

The pseudo-code of Figure 142 shall be used to collect the (possibly) three candidate motion vectors for the top right field block MV:

```
// Top Right Field Block MV
Add the top left field block MV of the current MB to the set of
candidate motion vectors.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom right block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of
        candidate motion vectors.
    } else if (B is 4 Field MV) {
        Add the top right field block MV of B to the set of
        candidate motion vectors.
    }
}

if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Add the top field MV of C to the set of
        candidate motion vectors.
    } else if (C is 4 Field MV) {
```

```
        if (C is top right MB) {
            Add the top left field block MV of C to the set of
          candidate motion vectors.
        } else { // C is top left MB
            Add the top right field block MV of C to the set of
          candidate motion vectors.
        }
    }
}
```

Figure 142: Candidate MV Derivation for Top Right MV in '4 Field MV' in Interlace Frame

The pseudo-code of Figure 143 shall be used to collect the (possibly) three candidate motion vectors for the bottom left field block MV:

```
// Bottom Left Field Block MV
if (A exists and A is not intra coded) {
    if (A is 1 MV) {
        Add MV of A to the set of candidate motion vectors.
    } else if (A is 4 Frame MV) {
        Add the bottom right block MV of A to the set of
        candidate motion vectors.
    } else if (A is 2 Field MV) {
        Add the bottom field MV of A to the set of
        candidate motion vectors.
    } else if (A is 4 Field MV) {
        Add the bottom right field block MV of A to the set of
        candidate motion vectors.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the bottom field MV of B to the set of
        candidate motion vectors.
    } else if (B is 4 Field MV) {
        Add the bottom left field block MV of B to the set of
        candidate motion vectors.
    }
}
```

```
if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Add the bottom field MV of C to the set of
        candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Add the bottom left field block MV of C to the set of
          candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right field block MV of C to the set
of candidate motion vectors.
        }
    }
}
```

Figure 143: Candidate MV Derivation for Bottom Left MV in '4 Field MV' in Interlace Frame

The pseudo-code of Figure 144 shall be used to collect the (possibly) three candidate motion vectors for the bottom right field block MV:

```
// Bottom Right Field Block MV
Add the bottom left field block MV of the current MB to the set of candidate motion vectors.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vectors.
    } else if (B is 4 Frame MV) {
        Add the bottom right block MV of B to the set of
        candidate motion vectors.
    } else if (B is 2 Field MV) {
        Add the bottom field MV of B to the set of
        candidate motion vectors.
    } else if (B is 4 Field MV) {
        Add the bottom right field block MV of B to the set of
        candidate motion vectors.
    }
}
```

```
if (C exists and C is not intra coded) {
    if (C is 1 MV) {
        Add MV of C to the set of candidate motion vectors.
    } else if (C is 4 Frame MV) {
        if (C is top right MB) {
            Add the bottom left block MV of C to the set of
            candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vectors.
        }
    } else if (C is 2 Field MV) {
        Add the bottom field MV of C to the set of
        candidate motion vectors.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Add the bottom left field block MV of C to the set of
             candidate motion vectors.
        } else { // C is top left MB
            Add the bottom right field block MV of C to the set
            of candidate motion vectors.
        }
    }
}
```

Figure 144: Candidate MV Derivation for Bottom Left MV in '4 Field MV' in Interlace Frame

10.7.3.5.5 Average Field Motion Vectors

Given two field motion vectors (MVX_1, MVY_1) and (MVX_2, MVY_2), the average operation used to form a candidate motion vector (MVX_A, MVY_A) shall be:

$MVX_A = (MVX_1 + MVX_2 + 1) >> 1;$

$MVY_A = (MVY_1 + MVY_2 + 1) >> 1;$

10.7.3.5.6 Computing Frame MV predictor(s) from Candidate Motion Vectors

This section defines how the MV predictor is computed for frame MVs given the set of candidate motion vectors. The operation shall be the same for computing the predictor for 1-MV or for each one of the four frame block MVs (4-MV Frame).

Let TotalValidMV denote the total number of motion vector(s) in the set of candidate motion vectors (TotalValidMV = 0, 1, 2, or 3).

Let ValidMV array denote the motion vector in the set of candidate motion vectors.

The pseudo-code of Figure 145 defines how the MV predictor (PMV_x, PMV_y) shall be computed.

```
if (TotalValidMV >= 2) {
    // Note that if there are only two valid MVs, then the
    // third ValidMV is set to be (0, 0)
```

```
        PMVx = median3 (ValidMVx [0], ValidMVx [1], ValidMVx [2]);
        PMVy = median3 (ValidMVy [0], ValidMVy [1], ValidMVy [2]);
} else if (TotalValidMV is 1) {
        PMVx = ValidMVx [0];
        PMVy = ValidMVy [0];
} else {
        PMVx = 0;
        PMVy = 0;
}
```

Figure 145: Computation of Frame MV Predictors from Candidate Motion Vectors

10.7.3.5.7 Computing Field MV predictor(s) from Candidate Motion Vectors

This section defines how the MV predictor(s) are computed for field MVs given the set of candidate motion vectors. The operation shall be the same for computing the predictor for each of the two field MVs or for each one of the four field block MVs (4-MV Frame).

First, the candidate motion vectors are separated into two sets, where one set contains only motion vectors that point to the same field as the current field and the other set contains motion vectors that point to the opposite field. Assuming that the motion vectors are represented in quarter pixel units, then the pseudo-code of Figure 146 shall be used on its y component to verify whether a candidate motion vector points to the same field:

```
if (ValidMVy & 4) {
    ValidMV points to the opposite field.
} else {
    ValidMV points to the same field.
}
```

Figure 146: Classifying Candidate Motion Vectors as Same Field or Opposite Field

Let SameFieldMV and OppFieldMV denote the two sets and let NumSameFieldMV and NumOppFieldMV denote the number of motion vectors that belongs to each set. The motion vectors in each set are ordered starting with predictor A if it exists, followed by predictor B if it exists, and then predictor C if it exists. For example, if the set of SameFieldMV contains only predictor B and predictor C, then SameFieldMV[0] equals to predictor B. The pseudo-code of Figure 147 shall define how the MV predictor (PMV_x, PMV_y) is computed:

```
if (TotalValidMV == 3) {
    if (NumSameFieldMV == 3 || NumOppFieldMV == 3) {
        PMVx = median3 (ValidMVx [0], ValidMVx [1], ValidMVx [2]);
        PMVy = median3 (ValidMVy [0], ValidMVy [1], ValidMVy [2]);
    } else if (NumSameFieldMV >= NumOppFieldMV) {
        PMVx = SameFieldMVx [0];
        PMVy = SameFieldMVy [0];
    } else {
        PMVx = OppFieldMVx [0];
        PMVy = OppFieldMVy [0];
    }
} else if (TotalValidMV == 2) {
    if (NumSameFieldMV >= NumOppFieldMV) {
```

```
        PMVx = SameFieldMVx [0];
        PMVy = SameFieldMVy [0];
    } else {
        PMVx = OppFieldMVx [0];
        PMVy = OppFieldMVy [0];
    }
} else if (TotalValidMV == 1) {
    PMVx = ValidMVx [0];
    PMVy = ValidMVy [0];
} else {
    PMVx = 0;
    PMVy = 0;
}
```

Figure 147: Computation of Field MV Predictors from Candidate Motion Vectors

10.7.3.6 Decoding Motion Vector Differential

The MVDATA syntax elements contain motion vector differential information for the macroblock. Depending on the type of motion compensation and motion vector block pattern signaled at each macroblock, there shall be 0 to 4 MVDATA syntax elements per macroblock. More specifically,

- For 1-MV macroblocks, there shall be either 0 or 1 MVDATA syntax element present depending on the MVP field in MBMODE.
- For 2 Field MV macroblocks, there shall be 0, 1, or 2 MVDATA syntax element(s) present depending on 2MVBP.
- For 4 Frame / Field MV macroblocks, there shall be 0, 1, 2, 3, or 4 MVDATA syntax element(s) present depending on 4MVBP.

The motion vector differential shall be decoded the same way as one reference field motion vector differential for field P picture described in section 10.3.5.4.2.1 with no halfpel mode.

10.7.3.7 Reconstructing Motion Vectors

Given the motion vector differential dmv, the luma motion vector shall be reconstructed by adding the differential to the predictor as follows:

mv_x = (dmv_x + PMVx) smod range_x

mv_y = (dmv_y + PMVy) smod range_y

The smod operation ensures that the reconstructed vectors are valid. (A smod b) lies within –b and b – 1. range_x and range_y depend on MVRANGE and are specified in Table 75.

Given a luma frame or field motion vector, a corresponding color-difference frame or field motion vector is derived to compensate a portion of the C_b/C_r block (it could be the entire portion). The FASTUVMC syntax element shall be ignored in interlace frame P and B pictures. The pseudo-code of Figure 148 shall define how a color-difference motion vector CMV is derived from a luma motion vector LMV.

```
Int s_RndTbl [] = {0, 0, 0, 1};
Int s_RndTblField [] = {0, 0, 1, 2, 4, 4, 5, 6, 2, 2, 3, 8, 6, 6, 7, 12};
CMVX = (LMVX + s_RndTbl[LMVX & 3]) >> 1;
if (LMV is a field motion vector) {
    CMVY = (LMVY >> 4)*8 + s_RndTblField [LMVY & 0xF];
} else {
    CMVY = (LMVY + s_RndTbl[LMVY & 3]) >> 1;
```

```
}
```

Figure 148: Reconstruction of MV in Interlace Frame Picture

10.7.4 Block Layer Decode

If the current macroblock is intra-coded, then the block layer shall be equivalent to decoding of a macroblock in interlace frame I pictures as defined in section 10.5.2.

If the current macroblock is inter-coded, then the block layer consists of decoding the residual after motion compensation and the process shall be the same as defined in section 10.3.6.2.

10.7.4.1 Motion Compensation

The motion compensation process for frame compensated macroblock shall be exactly the same as defined in section 8.3.6.5.

The motion compensation process for field compensated macroblock shall also be the same but with the filtering process applied to the same field lines. The following process is used to determine the starting location in the reference frame. Let (x_0, y_0) denote that frame coordinate of the top left pixel in the current macroblock and let (x_s, y_s) denote the top left pixel coordinate of the area inside the macroblock that is to be motion compensated. For the case of 2 field MV, (x_s, y_s) equals to (x_0, y_0) for the top field MV, and (x_s, y_s) equals to (x_0, y_0+1) for the bottom field MV. For the case of 4 field MV case, (x_s, y_s) equals to (x_0, y_0), (x_0+8, y_0), (x_0, y_0+1), or (x_0+8, y_0+1) for the top-left field MV, top-right field MV, bottom-left field MV, or bottom-right field MV, respectively. Given a motion vector (mv_x, mv_y) in quarter-pel unit and its (x_s, y_s) in integer-pel unit, the starting location (x_r, y_r) in the reference frame in quarter-pel unit shall be computed as.

$$(x_r, y_r) = (x_s * 4 + mv\_x, y_s * 4 + mv\_y);$$

To determine the starting field line, the vertical component of the starting location shall be truncated to the nearest integer. The field used for motion compensation is determined based on the truncated value of the vertical start position (even value = top, odd value = bottom). Thus, the set of lines used for the motion compensation process shall be the lines with the same polarity as the starting field line (i.e. every other lines extending from the starting field line.) Once the starting field line is determined, the vertical subpixel part of the starting location shall be interpreted as between the field lines starting at the starting field line while the horizontal subpixel part of the starting location shall be interpreted as between each horizontally adjacent sample on the field lines. The interpolation process shall then be performed as described in section 8.3.6.5 using the field lines. For example, if the starting location is (11.25, 1.75) then the starting field location shall be line 1 and the set of lines use for motion compensations shall be 1, 3, 5, …, etc. The vertical subpixel interpolation of .75 and horizontal subpixel interpolation of .25 shall be performed using these lines.

10.8 Interlace Frame B Picture Decoding

Interlace frame B picture decoding shall be the same as interlace P picture decoding, except as described in this section.

The additional elements are summarized here:

1) BFRACTION (9.1.1.40) at the picture level that tells us how to scale the direct mode MVs;

2) DIRECTMB (9.1.1.41) bitplane coding that is sent at the picture layer for direct/non-direct MB's;

3) DIRECTBBIT bit (9.1.3.14) at the MB level in the case where the direct mode bitplane is coded raw;

4) BMVTYPE (9.1.3.15) at the MB level that indicates if the MB is forward, backward or interpolated;

5) One bit MVSW (9.1.3.16)at the MB level, if the MB is coded in field mode and if BMVTYPE is forward or backward, to indicate if mode is switched from forward to backward (or backward to forward) in going from the top to the bottom field's MV; and

6) The syntax element 4MVSWITCH (9.1.1.28) is not present in interlace B picture headers. Instead, it is implicitly set to 0. However, 4MVBPTAB syntax element shall always be present in the frame header.

As with B frame MV decoding in progressive and field coding, two buffers are maintained, one each for forward and backward motion, and the rule "forward predicts forward, backward predicts backward" ahall be used. MV prediction

follows similar logic as interlace frame P pictures (refer to 10.7.3.5), but separate forward and backward contexts shall be retained. The holes in the buffer, i.e., the forward buffer location for a MB coded as backward, shall be filled with what would have been the predicted MV (see section 10.8.6.7.1).

10.8.1 Skipped Anchor Frames

If an anchor frame is coded as skipped then it shall be treated as a P frame which is identical to its reference frame. Therefore, the reconstruction of that frame can be treated conceptually as a copy of the reference frame. In this case, both anchor frames shall be identical for the intervening B frames. For example, if the frames are coded as follows in display order:

I0 B1 P2 B3 P4 B5 S6 (I0 P2 B1 P4 B3 S6 B5 in coding order) where S6 is the skipped frame

then this is treated as:

I0 B1 P2 B3 P4 B5 P4

because the skipped frame (S6) is treated as being identical to its reference (P4).

The motion vectors for the skipped anchor frame shall be set to zero.

10.8.2 Out-of-bounds Reference Pixels

This shall be identical to P interlace frame pictures as defined in section 10.7.2.

10.8.3 BFRACTION

The semantics of BFRACTION shall be identical to that of the corresponding syntax element in progressive B pictures as defined in 7.1.1.14.

10.8.4 Bitplane coding of direct mode

This shall be identical to progressive B pictures (refer to 8.4.5.1). The syntax element DIRECTMB (9.1.1.41) indicates if individual macroblocks at the frame level are coded in direct mode.

10.8.5 4MVSWITCH and 4MVBPTAB

The 4MVSWITCH syntax element (see section 9.1.1.28) shall not be present in interlace frame B pictures, and is always set to 0 in the decoder, i.e. no 4-MV. 4MVBPTAB syntax element (see section 9.1.1.37) shall always be present.

10.8.6 B Macroblock Layer Decode

At the MB level, the B frame syntax is also similar to P frame MB (refer to Figure 102 and Figure 103). Macroblocks in interlace frame B pictures shall be one of 3 types: 1-MV, 2 Field MV, or Intra. 4 Frame MV and 4 Field MV modes shall not be allowed. These modes shall be joint coded the same way as in interlace frame P pictures, with the MBMODE syntax element as defined in section 10.7.3.4. Each MB may also predicted as forward, backward, direct or interpolated (using DIRECTMB and BMVTYPE syntax elements). If a MB is of type 1-MV and either forward or backward, then it uses a single MV. If it is of type 1-MV and direct or interpolated, it uses 2-MVs. If it is of type 2 Field MV and either forward or backward predicted, then it uses 2-MVs. If it is of type 2 Field MV and direct or interpolated, then it uses 4-MVs.

Frame motion compensation treats a macroblock as a whole entity while field motion compensation treats a macroblock as composed of two separate fields. Additionally, when field motion compensation is used, the prediction type (BMVTYPE) may be different for the top and bottom fields. When this kind of field level switching of prediction type is used, the switching shall be restricted to be from forward to backward or vice-versa, i.e. direct and interpolated modes are not allowed to switch to a different prediction type at the field level.

I MBs in B frames shall also be the same as those in P frames as defined in 10.7.3.

10.8.6.1 Inter Macroblock Types

Only the 1-MV and 2 field MV prediction types shall apply to B pictures. The following sections describe four types of motion compensation:

10.8.6.1.1 1-MV Macroblock

In a 1-MV macroblock, the displacement of the luma blocks shall be represented by a single motion vector when the prediction type is forward or backward, and by two motion vectors when the type is direct or interpolated. Corresponding color-difference motion vectors shall be derived in each case as defined in 10.7.3.7. In the case of interpolated and direct prediction, the motion compensated pixels from forward and backward reference pictures shall be averaged to form the final prediction as follows:

Average pixel value = (Forward interpolated value + Backward interpolated value + 1) >> 1

10.8.6.1.2 2 Field MV Macroblock

In 2 Field MV macroblock, the displacement of each field of the luma blocks shall be described by a different motion vector (see section 10.7.3.1.2). Additionally, the prediction type is allowed to switch from forward to backward or vice-versa in going from the top to the bottom field, thus allowing the top field to be motion compensated from one reference picture and the bottom field to be motion compensated from the other one. In the case of interpolated and direct prediction, the motion compensated pixels from forward and backward reference pictures shall be averaged as described in section 10.8.6.1.1 to form the final prediction.

10.8.6.1.3 4 Frame MV Macroblock

This shall not be present in frame interlaced B pictures.

10.8.6.1.4 4 Field MV Macroblock

This shall not be present in frame interlaced B pictures.

10.8.6.1.5 Interpretation of 2MVBP, 4MVBP and order of motion vectors in B pictures

In a 1-MV macroblock, the 2MVBP syntax element (9.1.3.10) shall be used with interpolated mode to indicate which of the two MV differentials are present. Bit 1 corresponds to the forward and bit 0 corresponds to the backward motion vector.

In a 2 field MV macroblock, the 2MVBP syntax element shall be used with forward and backward mode, to indicate for which of the two fields, motion vector differential is present. Bit 1 corresponds to the top field and bit 0 corresponds to the bottom field motion vector. The same top/bottom signaling shall be used when MVSW syntax is used to switch from forward prediction for the top field to backward prediction for the bottom field, or vice-versa.

In a 2 field MV macroblock, the 4MVBP syntax element (9.1.3.11) shall be used with interpolated mode, to indicate which of the four MV differentials are present. Bit 3 corresponds to the top field forward motion vector, and bit 2 corresponds to the top field backward motion vector, and bit 1 corresponds to the bottom field forward and bit 0 to the bottom field backward motion vector.

In all cases, the bits of 2MVBP and 4MVBP being set to '1' shall signify that the corresponding motion vector differential is present, and if it is set to '0' the corresponding motion vector is equal to the predicted motion vector, i.e. there is no difference to add to the prediction. Also, in all cases the actual order of decoded motion vectors shall be sent in the same order as the bits in 2MVBP or 4MVBP, e.g. in 2 field MV macroblock using interpolated mode, the first motion vector to be received by the decoder is the top field forward, and the last (i.e. 4^{th}) motion vector to be received is the bottom field backward motion vector.

10.8.6.2 Skipped Macroblock Signaling

Macroblock skipping shall be signaled the same way as with P frames as defined in section 10.7.3.3. Skipped macroblocks shall only be of the 1-MV frame type, i.e. field motion compensation is not allowed. The motion vector shall be coded with zero differential motion vector (i.e. the macroblock is motion compensated using its 1-MV motion predictor) and there shall be no coded blocks (CBP == 0). If a macroblock is skipped, then only the BMVTYPE and DIRECTBBIT (if the bitplane coding is set to the raw mode) information shall be sent for that macroblock, so that the motion vector(s) is correctly predicted as forward, backward, direct or interpolated.

10.8.6.3 Macroblock Mode Signaling

The allowed macroblock modes are defined in 10.8.6.1, and the signaling of these allowed modes shall be identical to P frames as defined in section 10.7.3.4.

10.8.6.4 Prediction Type Decoding (BMVTYPE and MVSW)

The prediction type shall be decoded according to the following rules. If the picture level bitplane DIRECTMB (see section 9.1.1.41) indicates that a macroblock is of direct type, then the prediction type for that macroblock shall be set to direct. If DIRECTMB element is coded as raw, then an additional bit at the macroblock level, DIRECTBBIT, shall be used to decide whether the prediction type is direct or not.

If the prediction type is non-direct, then the BMVTYPE syntax element (see section 9.1.3.15) shall be decoded. This shall be identical to that used with progressive B pictures as defined in section 7.1.3.14. If the macroblock mode is '2-MV field coded' and if the BMVTYPE is either forward or backward, then the MVSW bit (see section 9.1.3.16) shall also be decoded to check whether or not the prediction type changes (i.e. flip from forward to backward or vice-versa) in going from the top to the bottom field for that macroblock.

10.8.6.5 B Frame Prediction Modes

Inter-coded macroblocks in B frames shall be one of four prediction modes: backward, forward, direct and interpolated. Additionally, field coded MBs shall have the ability to switch prediction modes from backward to forward, or forward to backward at the field level.

A macroblock in the forward mode macroblock shall be interpolated from its temporally previous anchor frame, whereas a macroblock in the backward mode shall be interpolated from the temporally subsequent (in display order) anchor frame.

The direct and interpolated modes use two motion vectors to predict from the two reference (anchor) frames. Both the direct and interpolated motion modes shall use round-up averaging for combining the pixel values of the two interpolated references into one set of macroblock pixels according to the following:

Average pixel value = (Forward interpolated value + Backward interpolated value + 1) >> 1

10.8.6.6 Deriving Direct Mode Motion Vectors

The calculation of direct mode motion vectors is similar to the corresponding computation in progressive B pictures as defined in section 8.4.5.4. First, motion vectors from the temporally subsequent anchor (I or P) interlace frame picture shall be buffered. Specifically, two MVs shall be buffered for each MB in the anchor frame. Therefore, in all the number of MVs buffered shall be equal to 2 × NumberOfMBs.

Nominally, one MV (denoted by MVT) shall be buffered for the top half of the MB or top field, and one MV (denoted by MVB) shall be buffered for the bottom half or bottom field. The rules for buffering shall be based on the macroblock coding mode of the co-located MB in the anchor frame as listed below, and are illustrated in Figure 149:

1. If the co-located MB in the anchor is coded as an Intra MB, MVT and MVB shall be set to zero.
2. If the co-located MB in the anchor is coded with 1-MV, MVT and MVB shall be set to that motion vector.
3. If the co-located MB in the anchor is field coded with 2-MVs, MVT and MVB shall be set to the top and bottom MVs respectively.
4. If the co-located MB in the anchor is coded with 4-MVs (regardless of field or frame coding mode), MVT and MVB shall be set to the motion vectors of the top left and bottom left blocks. In other words, if the 4-MVs are respectively MV1, MV2, MV3 and MV4, MVT shall be set to MV1 and MVB to MV3.

All buffered MVs shall be set to (0, 0) when the co-located MB in the temporally subsequent anchor is coded as Intra. This shall also hold true when the anchor is an I frame.

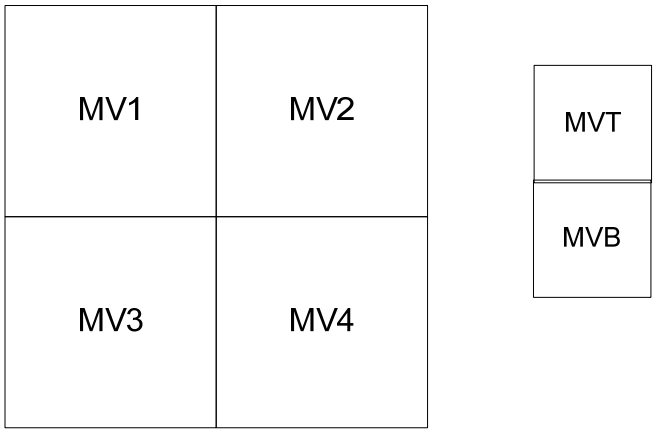


Figure 149: Buffering P frame MVs to use in B's direct mode: motion vectors (MV1, MV2, MV3 and MV4) corresponding to blocks in the co-located MB of the anchor frame are shown on left; buffered MVs (MVT and MVB) are shown on right. In general, MVT == MV1 and MVB == MV3.

Subsequently, the forward and backward pointing direct mode motion vectors corresponding to the top and bottom fields of the direct mode MB shall be calculated by applying the pseudo-code of Figure 150 to MVT and MVB:

```
Scale_Direct_MV (IN MV_X, IN MV_Y, OUT MV_X_F , OUT MV_Y_F, OUT MV_X_B, OUT MV_Y_B)
{
    MV_X_F = (MV_X * ScaleFactor + 128) >> 8;
    MV_Y_F = (MV_Y * ScaleFactor + 128) >> 8;
    MV_X_B = (MV_X * (ScaleFactor - 256) + 128) >> 8;
    MV_Y_B = (MV_Y * (ScaleFactor - 256) + 128) >> 8;
}
```

Figure 150: Deriving Direct Mode MVs in Interlace B Frames

In the pseudo-code of Figure 150, ScaleFactor shall be computed as defined in section 8.4.5.4. Scale_Direct_MV shall use the same logic as used for progressive B pictures in 8.4.5.4 (but without pull-back) to obtain the forward and backward pointing MVs.

10.8.6.7 Motion Vector Prediction in Interlace Frame B Pictures

MV prediction for interlace frame B pictures shall follow exactly the same rules as with interlace frame P pictures (10.7.3.5). Separate prediction contexts shall be used for forward and backward mode MVs. The forward prediction context shall be used to predict forward MVs and the backward prediction context shall be used for the prediction of backward MVs.

10.8.6.7.1 Populating the forward and backward prediction contexts

Since the causal neighbor MBs to the left and top of the current MB can use different coding modes, there is no assurance that both forward and backward MVs are available for predicting the MV or MVs of the current MB. The B frame MV predictor rules define the prediction of both forward and backward MVs, based on the causal entries in two buffers, the backward MV buffer and the forward MV buffer. Each buffer shall be of size 2 ×NumberOfMBs. In other words, each buffer accommodates one motion vector for the top field and one motion vector for the bottom field, for each macroblock in the frame. The population of these buffers shall proceed during the B frame decoding process, as follows:

1. If the current MB is coded in the Intra mode, the corresponding forward and backward MV buffer entries (both top and bottom fields) shall be set to zero.

2. If the current MB is coded using the Interpolated mode, the corresponding forward and backward MV buffer entries shall be set to the forward and backward motion vectors respectively.
3. If the current MB is coded using the Direct mode, the corresponding forward and backward MV buffer entries shall be set to the Direct mode forward and backward motion vectors respectively.
4. If the current MB is coded using the Forward mode, the corresponding forward MV buffer entries shall be set to the forward motion vector(s). For this MB, the population of the backward MV buffer can be viewed as a two step process:
    a. First, 2 Field prediction shall be used to select the candidate predictors, and to compute the actual prediction.
    b. Next, the population of the MV type in the backward MV buffer shall be set to be the same as the forward MV type, i.e., if the forward MV is 1-MV, the type of MV in the backward MV buffer shall also be set to be 1-MV, and only the top field MV (computed in the first step) shall be stored. If the forward MV is of the 2 Field MV, the type of MV in the backward MV buffer shall also be set to 2 Field MV, and both the top and bottom components (computed in the first step) shall be stored.
5. If the current MB is coded using the Backward mode, the corresponding backward MV buffer entries shall be set to the backward motion vector(s). For this MB, the population of the forward MV buffer can be viewed as a two step process:
    a. First, 2 Field MV prediction shall be used to select the candidate predictors, and to compute the prediction.
    b. Next, the population of the MV type in the forward MV buffer shall be set to be the same as the backward MV type, i.e., if the backward MV is 1-MV, the MV type in the forward MV buffer shall also be set to be 1-MV, and only the top field MV (computed in the first step) shall be stored. If the backward MV is of the 2 Field MV, the MV type in the forward MV buffer shall also be set to 2 Field MV, and both the top and bottom components (computed in the first step) shall be stored.
6. If the current MB codes the top and bottom fields with opposite direction MVs (i.e. by setting the MB to be coded with 2MVs, and setting syntax element MVSW = 1), the forward MV buffer entries for both top and bottom fields shall be set to the forward MV and the backward MV buffer entries for both top and bottom fields shall be set to the backward MV. In other words, if the top field is encoded using the forward mode, both forward MV buffer entries shall be set to this value (likewise for the bottom field / backward MV).

When the current MB is coded using 1-MV, the top and bottom field MVs shall be equal to the same motion vector.

10.8.6.7.2 Prediction scheme

The scheme for interlace frame B MV prediction shall be as follows:

1. If the MB is Forward (respectively Backward) predicted, the logic of section 10.7.3.5 shall be used to predict the forward MV (respectively backward MV) from the causal neighborhood of the forward (respectively backward) MV buffer.

2. If the MB is coded as Interpolated, the logic of section 10.7.3.5 shall be used to predict the forward MV from the causal neighborhood of the forward MV buffer, and to predict the backward MV from the causal neighborhood of the backward MV buffer.

3. If the MB is coded using the Direct mode, Direct mode motion vectors shall be derived using the procedure described in section 10.8.6.6.

10.8.6.8 Decoding Motion Vector Differential

This shall be identical to P pictures as defined in 10.7.3.6.

10.8.6.9 Reconstructing Motion Vectors

This shall be identical to P pictures as defined in 10.7.3.7.

10.8.7 B Block Layer Decode

Block decoding syntax and operations shall be the same as for P pictures as defined in 10.7.4.

10.9 Overlapped Transform

If the syntax element OVERLAP (6.2.10) is set to 1, then a filtering operation shall be conditionally performed (conditions specified in 10.9.1 and 10.9.2) across edges of two neighboring Intra blocks, for both the luma and color-difference channels.

10.9.1 Overlap Smoothing for Interlace Field Pictures

The overlap smoothing in I, BI, and P interlace field pictures shall be identical to the overlap smoothing for the corresponding I, BI and P progressive frames in advanced profile as described in Section 8.5.2. No overlap smoothing shall be performed for B interlace field pictures.

10.9.2 Overlap Smoothing for Interlace Frame Pictures

The overlap smoothing process in I, BI and P interlace frame pictures shall be the identical to the overlap smoothing for the corresponding I, BI and P progressive frames in advanced profile as described in Section 8.5.2 with the exception that only the vertical edges between I blocks shall be filtered and horizontal block boundaries shall not be filtered. No overlap smoothing shall be performed for B interlace frame pictures.

10.10 In-loop Deblock Filtering

If the syntax element LOOPFILTER == 1, then a filtering operation shall be performed on each reconstructed frame in the case of interlace frame pictures, and on each reconstructed field in the case of interlace field pictures. This filtering operation shall be performed prior to using the reconstructed frame or the reconstructed field as a reference for motion predictive coding. Therefore, it is necessary that the decoder perform the filtering operation strictly as defined.

Since the intent of loop filtering is to smooth out the discontinuities at block boundaries, the filtering process shall operate on the pixels that border neighboring blocks. For P pictures, the block boundaries may occur at every 4^{th}, 8^{th}, 12^{th}, etc pixel row or column, depending on whether an 8x8, 8x4, 4x8 or 4x4 Inverse Transform is used. For I pictures filtering occurs at every 8^{th}, 16^{th}, 24^{th}, etc pixel row and column.

10.10.1 I Interlace Field Picture In-loop Deblocking

For I pictures, deblock filtering shall be performed at all 8x8 block boundaries. Figure 75 and Figure 76 show the pixels that are filtered along the horizontal and vertical border regions.

The lines that shall be filtered are defined in formal terms in section 8.6.1.As the figures show, the top horizontal line and first vertical line shall not be filtered. Although not depicted, the bottom horizontal line and last vertical line are also not filtered. In more formal terms, the following lines shall be filtered:

N = the number of horizontal 8x8 blocks in the plane (N*8 = horizontal frame size)

M = the number of vertical 8x8 blocks in the frame (M*8 = vertical frame size)

Horizontal lines (7,8), (15,16) … ((N – 1)*8 – 1, (N –1)*8) are filtered

Vertical lines (7, 8), (15, 16) … ((M-1)*8 - 1, (M – 1)*8) are filtered.

The order in which the pixels are filtered is important. All the horizontal boundary lines in the frame shall be filtered first followed by the vertical boundary lines.

10.10.2 P Interlace Field Picture In-loop Deblocking

In-loop deblocking of P interlace field pictures shall be as defined in section 8.6.2, and the filtering process shall be as defined in section 8.6.4.

Note: The boundary between a block or subblock and a neighboring block or subblock is not filtered if both have the same motion vector (same X and Y component as well as the same reference field), and both have no residual error. Otherwise, the boundary is filtered.

10.10.3 B Interlace Field Picture In-loop Deblocking

For B interlace field pictures, all block boundaries shall be filtered, and the filtering process shall be as defined in section 8.6.4. The internal boundary between adjacent sub-blocks shall be filtered only if either one of the sub-blocks has at least one non-zero coefficient (i.e. non-zero residual error).

10.10.4 Interlace Frame Pictures In-loop Deblocking

In interlace frame pictures, each macroblock shall be either frame transform coded or field transform coded according to its FIELDTX flag (see 9.1.1.18, 9.1.3.1, and 10.7.3.4). The state of the FIELDTX flag along with the size of the transform (4x4, 4x8, 8x4, 8x8) used shall affect where the in-loop deblocking takes place in the macroblock.

10.10.4.1 Field-based Filtering

The filtering process shall be the same as described in section 8.6.4 with one important difference: the filtering shall always be done using the same field lines, never mixing different field lines. Figure 151 illustrates the field-based filtering for horizontal and vertical block boundaries.

For a horizontal block boundary, the two top field lines shall be filtered across the block boundary using top field lines only and the two bottom field lines across the block boundary shall be filtered using bottom field lines only. For a vertical block boundary, the top field block boundary and the bottom field block boundary shall be filtered separately.

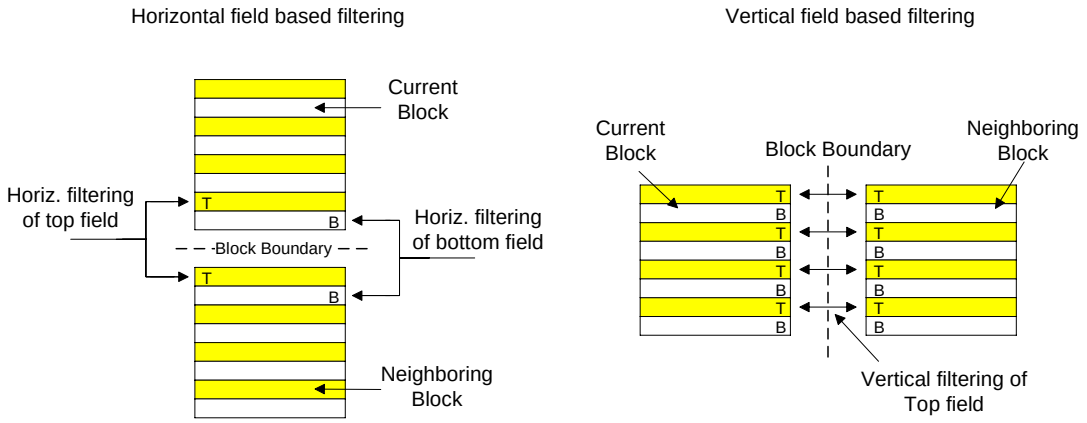


Figure 151: Field based horizontal / vertical block boundaries filtering

10.10.4.2 Filtering order

For both inter (P, B) and intra (I) interlace frame pictures, the in-loop deblocking process starts by processing all the horizontals edges first followed by all the vertical edges. The pseudo-code of Figure 152 describes this filtering process one macroblock at a time for the sake of simplicity, but the filtering process need not follow this macroblock processing order. Multiple filtering operations on the same pixels shall follow the same filtering order as that given in the pseudo-code.

```
// Processing horizontal edges
int X, Y
for (Y = 0; Y < number of MBs in a column; Y++) {
      for (X = 0; X < number of MBs in a row; X++) {
            Filter horizontal edges of MB located at Yth row, Xth col
      }
}

// Processing vertical edges
```

```
for (Y = 0; Y < number of MBs in a column; Y++) {
      for (X = 0; X < number of MBs in a row; X++) {
            Filter vertical edges of MB located at Yth row, Xth col
      }
}
```

Figure 152: Edge Ordering for In-loop Deblocking in Interlace Frame

10.10.4.3 Interlace Frame I Picture

In interlace frame I pictures, each macroblock shall be 8x8 transform coded.

For each macroblock, the horizontal block boundary filtering starts by filtering the intra-macroblock horizontal boundary only if the current macroblock is frame transform coded. Next, the horizontal block boundary between the current macroblock and the macroblock directly below it (if available) is filtered. The pseudo-code of Figure 153 shall describe the process of horizontal filtering a macroblock.

```
// Horizontal filtering of MB
// Luma
if (FIELDTX of current MB is FALSE) {
     - Filter all 16 pixels in row 6 and 8 of Y.
     - Filter all 16 pixels in row 7 and 9 of Y.
}

if (current bottom luma MB edge is not on a picture or slice boundary) {
- Filter all 16 pixels in row 14 and 16 of Y.
- Filter all 16 pixels in row 15 and 17 of Y.
}

// Color-difference
if (current bottom color-difference block edge is not on a picture or slice   boundary) {
- Filter all 8 pixels in row 6 and 8 of Cb and Cr.
- Filter all 8 pixels in row 7 and 9 of Cb and Cr.
}
```

Figure 153: Pseudo-code for Horizontal Filtering in Interlace Frame I Picture

For each macroblock, the vertical block boundary filtering starts by filtering the intra-macroblock vertical boundary and then followed by the filtering of the inter-macroblock boundary between the current macroblock and the macroblock to its immediate right (if available). The pseudo-code of Figure 154 shall describe the process of the vertical filtering a macroblock.

```
// Vertical filtering of MB
// Luma
- Filter the 8 even numbered pixels in column 7 and 8 of Y.
- Filter the 8 odd numbered pixels in column 7 and 8 of Y.

if (current right luma MB edge in not on a picture boundary) {
- Filter the 8 even numbered pixels in column 15 and 16 of Y.
- Filter the 8 odd numbered pixels in column 15 and 16 of Y.
```

```
}

// Color-difference
if (current right color-difference block edge is not on a picture boundary) {
- Filter the 4 even numbered pixels in column 7 and 8 of Cb.
- Filter the 4 odd numbered pixels in column 7 and 8 of Cb.
- Filter the 4 even numbered pixels in column 7 and 8 of Cr.
- Filter the 4 odd numbered pixels in column 7 and 8 of Cr.
}
```

Figure 154: Pseudo-code for Vertical Filtering in Interlace Frame I Picture

10.10.4.4 Interlace Frame P, B Picture

In interlace frame P, B pictures, each block shall be either 4x4, 4x8, 8x4, or 8x8 transform coded. When a block does not contain any coded coefficients, it shall be considered as 8x8 transform coded in the loop filtering process. For each macroblock, the horizontal block boundary filtering shall occur in the order of block Y_0, Y_1, Y_2, Y_3, C_b, and then C_r. The luma blocks are processed differently according to the value of the FIELDTX element.

Note: The value of FIELDTX is explicitly signaled in intra-coded macroblocks (9.1.3.1), and it is inferred from MBMODE in inter-coded macroblocks (10.7.3.4).

The pseudo-code of Figure 155 shall describe the process of horizontal filtering a macroblock. As shown in the pseudo-code, the horizontal boundaries of subblocks located in the top blocks of the first MB row, and the horizontal boundaries of bottom subblocks located in the last MB row shall not be filtered.

```
// Horizontal filtering of MB
// Luma
if (FIELDTX of current MB is FALSE) {
    // Block Y0
    if (current MB is not in the first MB row and the transform of Block Y0 is 8x4 or 4x4) {
        - Filter first 8 pixels in row 2 and 4 of Y.
        - Filter first 8 pixels in row 3 and 5 of Y.
    }
    - Filter first 8 pixels in row 6 and 8 of Y.
    - Filter first 8 pixels in row 7 and 9 of Y.
    // Block Y1
    if (current MB is not in the first MB row and the transform of Block Y1 is 8x4 or 4x4) {
        - Filter last 8 pixels in row 2 and 4 of Y.
        - Filter last 8 pixels in row 3 and 5 of Y.
    }
    - Filter last 8 pixels in row 6 and 8 of Y.
    - Filter last 8 pixels in row 7 and 9 of Y.
    // Block Y2
    if (current MB is not in the last MB row and the transform of Block Y2 is 8x4 or 4x4) {
        - Filter first 8 pixels in row 10 and 12 of Y.
        - Filter first 8 pixels in row 11 and 13 of Y.
    }
    if (current MB is not in the last MB row) {
        - Filter first 8 pixels in row 14 and 16 of Y.
```

```
            - Filter first 8 pixels in row 15 and 17 of Y.
      }
      // Block Y3
      if (current MB is not in the last MB row and the transform of Block Y3 is 8x4 or 4x4) {
            - Filter last 8 pixels in row 10 and 12 of Y.
            - Filter last 8 pixels in row 11 and 13 of Y.
      }
      if (current MB is not in the last MB row) {
            - Filter last 8 pixels in row 14 and 16 of Y.
            - Filter last 8 pixels in row 15 and 17 of Y.
      }
} else {
      // Block Y0
      if (the transform of Block Y0 is 8x4 or 4x4) {
            - Filter first 8 pixels in row 6 and 8 of Y.
      }
      if (current MB is not in the last MB row) {
            - Filter first 8 pixels in row 14 and 16 of Y.
      }

      // Block Y1
      if (the transform of Block Y1 is 8x4 or 4x4) {
            - Filter last 8 pixels in row 6 and 8 of Y.
      }
      if (current MB is not in the last MB row) {
            - Filter last 8 pixels in row 14 and 16 of Y.
      }
      // Block Y2
      if (the transform of Block Y2 is 8x4 or 4x4) {
            - Filter first 8 pixels in row 7 and 9 of Y.
      }
      if (current MB is not in the last MB row) {
            - Filter first 8 pixels in row 15 and 17 of Y.
      }
      // Block Y3
      if (the transform of Block Y3 is 8x4 or 4x4) {
            - Filter last 8 pixels in row 7 and 9 of Y.
      }
      if (current MB is not in the last MB row) {
            - Filter last 8 pixels in row 15 and 17 of Y.
      }
}

// Color-difference
if (current MB is not in the first or last MB row and the transform used for the Cb block is 8x4 or 4x4) {
      - Filter all 8 pixels in row 2 and 4 of Cb.
```

```
    - Filter all 8 pixels in row 3 and 5 of Cb.
}
If (current MB is not in the last MB row) {
    - Filter all 8 pixels in row 6 and 8 of Cb.
    - Filter all 8 pixels in row 7 and 9 of Cb.
}

if (current MB is not in the first or last MB row and the transform used for the Cr block is 8x4 or 4x4) {
    - Filter all 8 pixels in row 2 and 4 of Cr.
    - Filter all 8 pixels in row 3 and 5 of Cr.
}
If (current MB is not in the last MB row) {
    - Filter all 8 pixels in row 6 and 8 of Cr.
    - Filter all 8 pixels in row 7 and 9 of Cr.
}
```

Figure 155: Pseudo-code for Horizontal Filtering in Interlace Frame P/B Picture

Similarly, for each macroblock, the vertical block boundary filtering shall occur in the order of block Y_0, Y_1, Y_2, Y_3, C_b, and then C_r. The luma blocks shall be processed differently according to the value of the FIELDTX[MB] element as defined in Figure 156.

Note: The value of FIELDTX is explicitly signaled in intra-coded macroblocks (9.1.3.1), and it is inferred from MBMODE in inter-coded macroblocks (10.7.3.4).

The pseudo-code of Figure 156 shall describe the process of vertical filtering a macroblock:

```
// Vertical filtering of MB
// Luma
if (FIELDTX of current MB is FALSE) {
    // Block Y0
    if (the transform of Block Y0 is 4x8 or 4x4) {
        - Filter the 4 even numbered pixels of the first 8 pixels in column 3 and 4 of Y.
        - Filter the 4 odd numbered pixels of the first 8 pixels in column 3 and 4 of Y.
    }
    - Filter the 4 even numbered pixels of the first 8 pixels in column 7 and 8 of Y.
    - Filter the 4 odd numbered pixels of the first 8 pixels in column 7 and 8 of Y.
    // Block Y1
    if (the transform of Block Y1 is 4x8 or 4x4) {
        - Filter the 4 even numbered pixels of the first 8 pixels in column 11 and 12 of Y.
        - Filter the 4 odd numbered pixels of the first 8 pixels in column 11 and 12 of Y.
    }
    if (current MB is not in the last MB column) {
        - Filter the 4 even numbered pixels of the first 8 pixels in column 15 and 16 of Y.
        - Filter the 4 odd numbered pixels of the first 8 pixels in column 15 and 16 of Y.
    }
    // Block Y2
    if (the transform of Block Y2 is 4x8 or 4x4) {
        - Filter the 4 even numbered pixels of the last 8 pixels in column 3 and 4 of Y.
```

```
            - Filter the 4 odd numbered pixels of the last 8 pixels in column 3 and 4 of Y.
        }
        - Filter the 4 even numbered pixels of the last 8 pixels in column 7 and 8 of Y.
        - Filter the 4 odd numbered pixels of the last 8 pixels in column 7 and 8 of Y.
        // Block Y3
        if (the transform of Block Y3 is 4x8 or 4x4) {
            - Filter the 4 even numbered pixels of the last 8 pixels in column 11 and 12 of Y.
            - Filter the 4 odd numbered pixels of the last 8 pixels in column 11 and 12 of Y.
        }
        if (current MB is not in the last MB column) {
            - Filter the 4 even numbered pixels of the last 8 pixels in column 15 and 16 of Y.
            - Filter the 4 odd numbered pixels of the last 8 pixels in column 15 and 16 of Y.
        }
    } else {
        // Block Y0
        if (the transform of Block Y0 is 4x8 or 4x4) {
            - Filter the 8 even numbered pixels in column 3 and 4 of Y.
        }
        - Filter the 8 even numbered pixels in column 7 and 8 of Y.
        // Block Y1
        if (the transform of Block Y1 is 4x8 or 4x4) {
            - Filter the 8 even numbered pixels in column 11 and 12 of Y.
        }
        if (current MB is not in the last MB column) {
            - Filter the 8 even numbered pixels in column 15 and 16 of Y.
        }
        // Block Y2
        if (the transform of Block Y2 is 4x8 or 4x4) {
            - Filter the 8 odd numbered pixels in column 3 and 4 of Y.
        }
        - Filter the 8 odd numbered pixels in column 7 and 8 of Y.
        // Block Y3
        if (the transform of Block Y3 is 4x8 or 4x4) {
            - Filter the 8 odd numbered pixels in column 11 and 12 of Y.
        }
        if (current MB is not in the last MB column) {
            - Filter the 8 odd numbered pixels in column 15 and 16 of Y.
        }
    }

    // Color-difference
    if (the transform of Cb Block is 4x8 or 4x4) {
        - Filter the 4 even numbered pixels in column 3 and 4 of Cb.
        - Filter the 4 odd numbered pixels in column 3 and 4 of Cb.
    }
    if (current MB is not in the last MB column) {
```

```
    - Filter the 4 even numbered pixels in column 7 and 8 of Cb.
    - Filter the 4 odd numbered pixels in column 7 and 8 of Cb.
}

if (the transform of Cr Block is 4x8 or 4x4) {
    - Filter the 4 even numbered pixels in column 3 and 4 of Cr.
    - Filter the 4 odd numbered pixels in column 3 and 4 of Cr.
}
if (current MB is not in the last MB column) {
    - Filter the 4 even numbered pixels in column 7 and 8 of Cr.
    - Filter the 4 odd numbered pixels in column 7 and 8 of Cr.
}
```

Figure 156: Pseudo-code for Vertical Filtering in Interlace Frame P/B Picture

11 Tables

The tables of this section are referenced elsewhere in this specification and are aggregated here to improve the reading of the other sections. In most cases, the coefficients in these tables are required to be used in the decoding processes.

11.1 Interlace Pictures MV Block Pattern VLC Tables

11.1.1 4-MV Block Pattern Tables

Table 116: 4-MV Block Pattern Table 0

4-MV Coded Pattern	VLC Codeword	VLC Codeword Size
0	14	5
1	58	6
2	59	6
3	25	5
4	12	5
5	26	5
6	15	5
7	15	4
8	13	5
9	24	5
10	27	5
11	0	3
12	28	5
13	1	3
14	2	3
15	2	2

Table 117: 4-MV Block Pattern Table 1

4-MV Coded Pattern	VLC Codeword	VLC Codeword Size
0	8	4
1	18	5
2	19	5
3	4	4
4	20	5
5	5	4
6	30	5
7	11	4
8	21	5
9	31	5
10	6	4
11	12	4
12	7	4
13	13	4
14	14	4
15	0	2

Table 118: 4-MV Block Pattern Table 2

4-MV Coded Pattern	VLC Codeword	VLC Codeword Size
0	15	4
1	6	4
2	7	4
3	2	4
4	8	4
5	3	4
6	28	5
7	9	4
8	10	4
9	29	5
10	4	4
11	11	4
12	5	4
13	12	4
14	13	4
15	0	3

Table 119: 4-MV Block Pattern Table 3

4-MV Coded Pattern	VLC Codeword	VLC Codeword Size
0	0	2
1	11	4

2	12	4
3	4	4
4	13	4
5	5	4
6	30	5
7	16	5
8	14	4
9	31	5
10	6	4
11	17	5
12	7	4
13	18	5
14	19	5
15	10	4

11.1.2 2-MV Block Pattern Tables

Table 120: Interlace Frame 2 MVP Block Pattern Table 0

Top	Bottom	VLC Codeword	VLC Size
0	0	2	2
0	1	1	2
1	0	0	2
1	1	3	2

Table 121: Interlace Frame 2 MVP Block Pattern Table 1

Top	Bottom	VLC Codeword	VLC Size
0	0	1	1
0	1	0	2
1	0	2	3
1	1	3	3

Table 122: Interlace Frame 2 MVP Block Pattern Table 2

Top	Bottom	VLC Codeword	VLC Size
0	0	2	3
0	1	0	2
1	0	3	3
1	1	1	1

Table 123: Interlace Frame 2 MVP Block Pattern Table 3

Top	Bottom	VLC Codeword	VLC Size
0	0	1	1
0	1	3	3
1	0	2	3
1	1	0	2

11.2 Interlace CBPCY VLC Tables

Table 124: Interlaced CBPCY Table 0

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	12058	15	33	686	11
2	12059	15	34	687	11
3	6028	14	35	1506	12
4	144	9	36	310	10
5	680	11	37	622	11
6	681	11	38	623	11
7	3015	13	39	765	11
8	145	9	40	158	9
9	682	11	41	318	10
10	683	11	42	319	10
11	1504	12	43	383	10
12	74	8	44	80	8
13	150	9	45	66	8
14	151	9	46	67	8
15	189	9	47	44	7
16	146	9	48	81	8
17	684	11	49	164	9
18	685	11	50	165	9
19	1505	12	51	190	9
20	152	9	52	83	8
21	306	10	53	68	8
22	307	10	54	69	8
23	377	10	55	45	7
24	308	10	56	84	8
25	618	11	57	70	8
26	619	11	58	71	8
27	764	11	59	46	7
28	78	8	60	3	3
29	64	8	61	0	3
30	65	8	62	1	3
31	43	7	63	1	1
32	147	9			

Table 125: Interlaced CBPCY Table 1

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	65	7	33	20	7
2	66	7	34	21	7
3	256	9	35	44	8
4	67	7	36	92	8
5	136	8	37	93	9
6	137	8	38	94	9
7	257	9	39	95	9
8	69	7	40	38	7
9	140	8	41	93	8
10	141	8	42	94	8
11	258	9	43	95	8
12	16	6	44	13	6
13	34	7	45	52	7
14	35	7	46	53	7
15	36	7	47	27	6
16	71	7	48	20	6
17	16	7	49	39	7
18	17	7	50	42	7
19	259	9	51	43	7
20	37	7	52	14	6
21	88	8	53	56	7
22	89	8	54	57	7
23	90	8	55	29	6
24	91	8	56	15	6
25	90	9	57	60	7
26	91	9	58	61	7
27	92	9	59	31	6
28	12	6	60	5	3
29	48	7	61	9	4
30	49	7	62	0	3
31	25	6	63	3	2
32	9	6			

Table 126: Interlaced CBPCY Table 2

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	50	6	33	234	8
2	51	6	34	235	8
3	26	5	35	489	9
4	38	6	36	74	7
5	228	8	37	442	9
6	229	8	38	443	9
7	486	9	39	475	9
8	39	6	40	32	6
9	230	8	41	222	8

10	231	8	42	223	8
11	487	9	43	242	8
12	14	5	44	34	6
13	99	7	45	85	7
14	108	7	46	88	7
15	119	7	47	45	6
16	40	6	48	15	5
17	232	8	49	112	7
18	233	8	50	113	7
19	488	9	51	120	7
20	123	7	52	35	6
21	218	8	53	89	7
22	219	8	54	92	7
23	236	8	55	47	6
24	245	8	56	36	6
25	440	9	57	93	7
26	441	9	58	98	7
27	474	9	59	48	6
28	33	6	60	2	3
29	75	7	61	31	5
30	84	7	62	6	4
31	43	6	63	0	2
32	41	6			

Table 127: Interlaced CBPCY Table 3

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	40	6	33	499	9
2	41	6	34	500	9
3	157	8	35	501	9
4	0	4	36	17	6
5	490	9	37	978	10
6	491	9	38	979	10
7	492	9	39	305	9
8	1	4	40	9	5
9	493	9	41	350	9
10	494	9	42	351	9
11	495	9	43	156	8
12	5	4	44	16	5
13	240	8	45	168	8
14	241	8	46	169	8
15	59	7	47	56	7
16	2	4	48	6	4
17	496	9	49	242	8
18	497	9	50	243	8
19	498	9	51	77	7
20	63	6	52	17	5
21	348	9	53	170	8

22	349	9	54	171	8
23	153	8	55	57	7
24	16	6	56	18	5
25	976	10	57	172	8
26	977	10	58	173	8
27	304	9	59	58	7
28	15	5	60	6	3
29	158	8	61	22	5
30	159	8	62	23	5
31	251	8	63	14	4
32	3	4			

Table 128: Interlaced CBPCY Table 4

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	60	6	33	105	7
2	61	6	34	108	7
3	31	5	35	5	7
4	10	5	36	96	7
5	97	7	37	26	8
6	98	7	38	27	8
7	2	7	39	53	8
8	11	5	40	19	6
9	99	7	41	14	7
10	100	7	42	15	7
11	3	7	43	21	7
12	7	5	44	45	6
13	3	6	45	109	7
14	4	6	46	110	7
15	11	6	47	56	6
16	12	5	48	8	5
17	101	7	49	8	6
18	102	7	50	9	6
19	4	7	51	12	6
20	18	6	52	46	6
21	10	7	53	111	7
22	11	7	54	114	7
23	20	7	55	58	6
24	27	7	56	47	6
25	24	8	57	115	7
26	25	8	58	0	6
27	52	8	59	59	6
28	44	6	60	7	4
29	103	7	61	20	5
30	104	7	62	21	5
31	53	6	63	4	3
32	13	5			

Table 129: Interlaced CBPCY Table 5

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	56	6	33	154	8
2	57	6	34	155	8
3	157	8	35	156	8
4	10	4	36	25	6
5	145	8	37	974	10
6	146	8	38	975	10
7	147	8	39	215	9
8	11	4	40	9	5
9	148	8	41	488	9
10	149	8	42	489	9
11	150	8	43	144	8
12	3	4	44	15	5
13	238	8	45	232	8
14	239	8	46	233	8
15	54	7	47	246	8
16	12	4	48	5	4
17	151	8	49	240	8
18	152	8	50	241	8
19	153	8	51	55	7
20	8	5	52	16	5
21	484	9	53	234	8
22	485	9	54	235	8
23	106	8	55	247	8
24	24	6	56	17	5
25	972	10	57	236	8
26	973	10	58	237	8
27	214	9	59	52	7
28	14	5	60	0	3
29	158	8	61	62	6
30	159	8	62	63	6
31	245	8	63	2	4
32	13	4			

Table 130: Interlaced CBPCY Table 6

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	60	6	33	229	8
2	61	6	34	230	8
3	463	9	35	128	8
4	0	3	36	46	6
5	191	8	37	2021	11
6	224	8	38	2022	11
7	508	9	39	2023	11
8	1	3	40	22	5
9	225	8	41	1012	10

10	226	8	42	1013	10
11	509	9	43	1014	10
12	9	4	44	25	5
13	497	9	45	258	9
14	498	9	46	259	9
15	499	9	47	260	9
16	2	3	48	10	4
17	227	8	49	500	9
18	228	8	50	501	9
19	510	9	51	502	9
20	17	5	52	26	5
21	1006	10	53	261	9
22	1007	10	54	262	9
23	1008	10	55	263	9
24	33	6	56	27	5
25	2018	11	57	376	9
26	2019	11	58	377	9
27	2020	11	59	462	9
28	24	5	60	29	5
29	1015	10	61	189	8
30	1022	10	62	190	8
31	1023	10	63	496	9
32	3	3			

Table 131: Interlaced CBPCY Table 7

Coded Block Pattern	VLC Codeword	VLC Codeword Size	Coded Block Pattern	VLC Codeword	VLC Codeword Size
1	3	6	33	52	7
2	4	6	34	53	7
3	438	10	35	17	7
4	4	3	36	22	6
5	46	7	37	105	10
6	47	7	38	106	10
7	14	7	39	107	10
8	5	3	40	10	5
9	48	7	41	54	9
10	49	7	42	55	9
11	15	7	43	216	9
12	3	4	44	30	6
13	10	8	45	442	10
14	11	8	46	443	10
15	20	8	47	444	10
16	6	3	48	4	4
17	50	7	49	21	8
18	51	7	50	22	8
19	16	7	51	23	8
20	5	5	52	31	6
21	48	9	53	445	10
22	49	9	54	446	10

23	50	9	55	447	10
24	9	6	56	0	5
25	102	10	57	16	9
26	103	10	58	17	9
27	104	10	59	18	9
28	29	6	60	28	6
29	439	10	61	217	9
30	440	10	62	218	9
31	441	10	63	19	9
32	7	3			

11.3 Interlace MV Tables

Table 132: 2-Field Reference Interlace MV Table 0

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	12	4	42	207	10	84	977	11
1	28	5	43	1395	12	85	408	11
2	11	5	44	9	5	86	489	11
3	0	5	45	35	7	87	1309	12
4	14	6	46	237	8	88	180	12
5	42	7	47	24	7	89	63	8
6	80	8	48	6	7	90	1109	12
7	872	10	49	68	8	91	555	11
8	2	2	50	245	9	92	553	11
9	26	5	51	121	9	93	1105	12
10	4	5	52	1746	11	94	1400	12
11	58	6	53	110	7	95	1970	12
12	29	6	54	43	9	96	1392	12
13	108	7	55	349	10	97	341	13
14	239	8	56	23	9	98	50	8
15	444	9	57	895	10	99	976	12
16	351	10	58	324	10	100	84	11
17	15	4	59	206	10	101	1747	11
18	3	5	60	40	10	102	1393	12
19	28	6	61	171	12	103	1108	12
20	13	6	62	16	6	104	820	12
21	11	7	63	437	9	105	7153	13
22	62	8	64	247	9	106	183	12

23	167	9	65	166	9	107	41	9
24	326	10	66	123	9	108	7812	14
25	409	11	67	40	9	109	364	13
26	6	4	68	493	10	110	411	11
27	31	6	69	489	10	111	7152	13
28	4	6	70	1789	11	112	1401	12
29	60	7	71	4	7	113	3907	13
30	7	7	72	245	10	114	181	12
31	446	9	73	41	10	115	2209	13
32	139	9	74	650	11	116	42	9
33	44	10	75	651	11	117	365	13
34	1971	12	76	655	11	118	2208	13
35	5	5	77	3577	12	119	1952	12
36	219	8	78	821	12	120	977	12
37	86	8	79	7813	14	121	2789	13
38	236	8	80	238	8	122	340	13
39	82	8	81	701	11	123	2788	13
40	445	9	82	43	10	124	2617	13
41	120	9	83	984	11	125	2616	13

Table 133: 2-Field Reference Interlace MV Table 1

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	3	3	42	7408	13	84	827	10
1	9	4	43	2881	13	85	697	10
2	22	5	44	50	6	86	1771	11
3	16	6	45	230	8	87	1392	11
4	215	8	46	224	8	88	3620	12
5	821	10	47	207	8	89	925	10
6	1396	11	48	171	8	90	1442	12
7	1365	11	49	412	9	91	1443	12
8	0	2	50	683	10	92	3709	12
9	29	5	51	3627	12	93	1518	11
10	9	5	52	5593	13	94	1849	11
11	23	6	53	111	7	95	1364	11

12	44	7	54	451	9	96	2725	12
13	173	8	55	175	8	97	2724	12
14	884	10	56	191	8	98	887	10
15	1715	11	57	172	8	99	7413	13
16	1399	11	58	381	9	100	3022	12
17	15	4	59	1763	11	101	3705	12
18	24	5	60	3625	12	102	1632	11
19	10	5	61	6532	13	103	1652	11
20	46	6	62	84	7	104	1770	11
21	34	7	63	181	9	105	3708	12
22	380	9	64	378	9	106	3429	12
23	3707	12	65	429	9	107	758	10
24	7049	13	66	409	9	108	5594	13
25	5592	13	67	376	9	109	7048	13
26	8	4	68	856	10	110	1441	12
27	52	6	69	722	11	111	7412	13
28	109	7	70	7243	13	112	1510	11
29	35	7	71	91	8	113	3624	12
30	450	9	72	680	10	114	1397	11
31	886	10	73	817	10	115	3428	12
32	723	11	74	904	10	116	820	10
33	7242	13	75	907	10	117	13067	14
34	13066	14	76	880	10	118	5595	13
35	20	5	77	1811	11	119	2880	13
36	106	7	78	3267	12	120	3023	12
37	114	7	79	7409	13	121	3525	12
38	108	7	80	441	9	122	3626	12
39	227	8	81	1519	11	123	1653	11
40	411	9	82	1848	11	124	1393	11
41	1855	11	83	754	10	125	1363	11

Table 134: 2-Field Reference Interlace MV Table 2

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	4	4	42	244	10	84	4	10

1	2	4	43	1764	12	85	440	10
2	16	5	44	1	5	86	192	9
3	3	5	45	60	8	87	634	10
4	23	6	46	125	8	88	785	11
5	69	7	47	141	8	89	156	8
6	62	8	48	157	8	90	1569	12
7	126	9	49	49	8	91	409	11
8	3	2	50	110	9	92	796	11
9	2	5	51	662	10	93	247	10
10	40	6	52	205	10	94	995	11
11	30	6	53	37	6	95	854	11
12	21	6	54	329	9	96	393	10
13	71	7	55	50	8	97	5	10
14	2	7	56	137	8	98	107	8
15	333	9	57	54	8	99	2242	12
16	96	9	58	136	8	100	816	12
17	11	4	59	111	9	101	1279	11
18	38	6	60	3	9	102	1264	11
19	36	6	61	797	11	103	849	11
20	20	6	62	14	6	104	1266	11
21	50	7	63	426	10	105	498	10
22	111	8	64	638	10	106	883	11
23	195	9	65	97	9	107	0	8
24	1329	11	66	334	9	108	3137	13
25	1765	12	67	335	9	109	2243	12
26	21	5	68	103	9	110	2540	12
27	63	7	69	255	10	111	994	11
28	45	7	70	387	10	112	772	11
29	1	7	71	54	7	113	1271	11
30	318	9	72	855	11	114	1265	11
31	221	9	73	245	10	115	496	10
32	246	10	74	198	9	116	328	9
33	773	11	75	194	9	117	3136	13
34	817	12	76	665	10	118	2541	12
35	14	5	77	281	9	119	2240	12
36	3	7	78	561	10	120	2241	12

37	52	7	79	848	11	121	1267	11
38	51	7	80	44	7	122	1278	11
39	26	7	81	399	10	123	254	10
40	330	9	82	1328	11	124	499	10
41	197	9	83	663	10	125	425	10

Table 135: 2-Field Reference Interlace MV Table 3

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	3	42	16462	15	84	2580	12
1	4	4	43	5175	13	85	699	11
2	47	6	44	43	6	86	401	11
3	82	7	45	133	8	87	2127	12
4	16	7	46	167	8	88	5176	13
5	173	9	47	160	8	89	175	9
6	1291	11	48	332	9	90	2967	12
7	400	11	49	666	10	91	1155	13
8	3	2	50	812	12	92	5179	13
9	22	5	51	8499	14	93	811	12
10	7	5	52	5162	13	94	579	12
11	13	6	53	81	7	95	5163	13
12	187	8	54	644	10	96	2392	14
13	371	9	55	172	9	97	10687	14
14	201	10	56	258	9	98	73	9
15	1295	11	57	69	9	99	2668	12
16	5932	13	58	68	9	100	5339	13
17	3	3	59	2075	12	101	1197	13
18	17	5	60	1630	13	102	5342	13
19	5	5	61	3255	14	103	2126	12
20	67	7	62	24	7	104	5172	13
21	35	8	63	1292	11	105	599	12
22	75	9	64	530	10	106	11866	14
23	814	12	65	740	10	107	519	10
24	11867	14	66	515	10	108	5173	13
25	1154	13	67	148	10	109	5177	13

26	9	4	68	290	11	110	3254	14
27	42	6	69	2074	12	111	5178	13
28	20	6	70	1621	13	112	404	11
29	42	7	71	51	8	113	1620	13
30	264	9	72	698	11	114	8501	14
31	1482	11	73	582	12	115	21372	15
32	1626	13	74	578	12	116	348	10
33	8502	14	75	2670	12	117	576	12
34	8498	14	76	1036	11	118	4114	13
35	11	5	77	2056	12	119	21373	15
36	19	7	78	8500	14	120	2393	14
37	65	7	79	16463	15	121	4248	13
38	184	8	80	373	9	122	5174	13
39	372	9	81	1029	11	123	1631	13
40	256	9	82	583	12	124	8230	14
41	5338	13	83	298	11	125	8503	14

Table 136: 2-Field Reference Interlace MV Table 4

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	5	4	42	966	10	84	295	9
1	25	5	43	1935	11	85	141	9
2	22	5	44	63	6	86	539	10
3	17	5	45	166	8	87	1970	11
4	62	6	46	240	8	88	479	10
5	94	7	47	58	7	89	984	10
6	239	8	48	82	7	90	1892	12
7	226	8	49	78	7	91	3812	12
8	0	2	50	227	8	92	947	11
9	57	6	51	473	9	93	1869	11
10	43	6	52	783	10	94	472	10
11	38	6	53	16	6	95	1500	11
12	40	6	54	477	9	96	2122	12
13	18	6	55	167	8	97	1177	11
14	194	8	56	247	8	98	965	10

15	237	9	57	34	7	99	7566	13
16	285	10	58	146	8	100	1893	12
17	13	4	59	964	10	101	1077	11
18	49	6	60	751	10	102	1905	11
19	42	6	61	1890	11	103	450	10
20	37	6	62	121	7	104	280	10
21	32	6	63	143	9	105	956	11
22	92	7	64	474	9	106	897	11
23	493	9	65	135	8	107	903	11
24	589	10	66	232	8	108	31539	15
25	1904	11	67	186	8	109	4247	13
26	6	4	68	374	9	110	4246	13
27	122	7	69	238	9	111	7885	13
28	96	7	70	944	10	112	3737	12
29	79	7	71	133	8	113	3868	12
30	72	7	72	281	10	114	3869	12
31	57	7	73	782	10	115	3813	12
32	390	9	74	264	9	116	284	10
33	531	10	75	466	9	117	31538	15
34	3782	12	76	268	9	118	15768	14
35	15	5	77	1907	11	119	7567	13
36	38	7	78	1060	11	120	3736	12
37	95	7	79	1076	11	121	3943	12
38	117	7	80	113	8	122	957	11
39	112	7	81	1501	11	123	896	11
40	39	7	82	449	10	124	1176	11
41	475	9	83	935	10	125	902	11

Table 137: 2-Field Reference Interlace MV Table 5

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	13	4	42	1887	11	84	363	9
1	16	5	43	3153	12	85	957	10
2	46	6	44	21	5	86	705	10
3	57	6	45	71	7	87	1580	11

4	13	6	46	238	8	88	7678	13
5	116	7	47	226	8	89	14	7
6	237	8	48	234	8	90	1438	11
7	182	8	49	9	8	91	1471	11
8	1	2	50	362	9	92	218	11
9	2	4	51	707	10	93	1577	11
10	0	5	52	1437	11	94	1412	11
11	48	6	53	61	6	95	3767	12
12	41	6	54	8	8	96	2826	12
13	112	7	55	473	9	97	1645	13
14	243	8	56	50	8	98	12	7
15	140	8	57	14	8	99	1918	11
16	358	9	58	366	9	100	1436	11
17	9	4	59	812	10	101	1912	11
18	51	6	60	1627	11	102	1886	11
19	120	7	61	6507	13	103	1882	11
20	6	7	62	2	5	104	1581	11
21	196	8	63	15	8	105	823	12
22	11	8	64	472	9	106	820	12
23	355	9	65	141	8	107	407	9
24	204	10	66	180	8	108	7767	13
25	1470	11	67	484	9	109	7652	13
26	31	5	68	103	9	110	6506	13
27	47	6	69	791	10	111	7766	13
28	100	7	70	1940	11	112	3152	12
29	24	7	71	34	6	113	2879	12
30	198	8	72	958	10	114	7764	13
31	10	8	73	789	10	115	2827	12
32	354	9	74	52	9	116	398	9
33	704	10	75	55	9	117	438	12
34	3827	12	76	734	10	118	7765	13
35	7	5	77	108	10	119	3252	12
36	15	7	78	3838	12	120	2878	12
37	227	8	79	1644	13	121	3766	12
38	202	8	80	40	6	122	7653	13
39	178	8	81	971	10	123	7679	13

40	399	9	82	940	10	124	821	12
41	942	10	83	53	9	125	439	12

Table 138: 2-Field Reference Interlace MV Table 6

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	1	3	42	717	13	84	346	12
1	11	5	43	1037585	21	85	359	12
2	25	6	44	20	6	86	3531	13
3	111	8	45	173	9	87	1413	14
4	42	9	46	170	9	88	1037591	21
5	117	10	47	20	8	89	1015	11
6	2027	12	48	168	9	90	16213	15
7	355	12	49	339	10	91	1037592	21
8	1	1	50	232	11	92	3548	13
9	14	5	51	510	12	93	1414	14
10	26	6	52	3535	13	94	16214	15
11	62	7	53	120	8	95	1037593	21
12	28	8	54	440	10	96	16215	15
13	45	9	55	338	10	97	1037594	21
14	356	12	56	254	11	98	442	10
15	2028	12	57	689	11	99	1415	14
16	357	12	58	349	12	100	1416	14
17	4	4	59	352	12	101	3551	13
18	6	6	60	1037586	21	102	690	13
19	54	7	61	1037587	21	103	1037595	21
20	127	8	62	122	8	104	3534	13
21	174	9	63	688	11	105	1014	13
22	344	12	64	485	10	106	1037596	21
23	348	12	65	233	11	107	4052	13
24	1389	14	66	252	11	108	1037597	21
25	1037584	21	67	1766	12	109	1037598	21
26	0	4	68	3528	13	110	1037599	21
27	4	6	69	1412	14	111	518784	20
28	123	8	70	1037588	21	112	518785	20

29	243	9	71	171	9	113	1388	14
30	59	9	72	3550	13	114	518786	20
31	2029	12	73	345	10	115	518787	20
32	691	13	74	1012	11	116	886	11
33	716	13	75	3529	13	117	1417	14
34	1390	14	76	3530	13	118	1418	14
35	24	6	77	506	12	119	518788	20
36	62	9	78	1037589	21	120	518789	20
37	23	8	79	1037590	21	121	3549	13
38	30	8	80	252	9	122	518790	20
39	175	9	81	511	12	123	518791	20
40	1015	13	82	484	10	124	1419	14
41	1391	14	83	175	11	125	32425	16

Table 139: 2-Field Reference Interlace MV Table 7

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	3	2	42	25902	16	84	1608	12
1	14	5	43	214727	20	85	1602	12
2	15	6	44	62	7	86	3206	13
3	126	8	45	57	8	87	3212	13
4	98	9	46	53	8	88	214732	20
5	198	10	47	51	8	89	58	10
6	3289	13	48	415	10	90	6583	14
7	1598	13	49	448	11	91	67	11
8	2	2	50	3290	13	92	807	11
9	2	4	51	214728	20	93	140	12
10	0	5	52	214729	20	94	141	12
11	24	6	53	11	8	95	3213	13
12	12	8	54	208	10	96	214733	20
13	105	9	55	414	10	97	214734	20
14	57	10	56	34	10	98	823	11
15	1799	13	57	56	10	99	3301	13
16	3198	14	58	398	11	100	133	12
17	2	3	59	798	12	101	806	11

18	13	5	60	12948	15	102	839	12
19	27	7	61	572	14	103	3236	13
20	15	8	62	50	8	104	3199	14
21	410	10	63	18	9	105	3354	14
22	1607	12	64	19	9	106	214735	20
23	6711	15	65	113	9	107	808	11
24	214724	20	66	413	10	108	107360	19
25	13421	16	67	32	10	109	107361	19
26	1	4	68	3207	13	110	3288	13
27	30	6	69	3264	13	111	1676	13
28	127	8	70	214730	20	112	12949	15
29	10	8	71	824	11	113	12950	15
30	225	10	72	1619	12	114	25903	16
31	1633	12	73	418	11	115	26328	16
32	3300	13	74	810	11	116	817	11
33	214725	20	75	802	11	117	1798	13
34	214726	20	76	3303	13	118	573	14
35	29	7	77	132	12	119	118	11
36	48	8	78	287	13	120	3265	13
37	13	8	79	214731	20	121	898	12
38	203	9	80	805	11	122	3302	13
39	409	10	81	1609	12	123	26329	16
40	800	11	82	811	11	124	26330	16
41	142	12	83	119	11	125	26331	16

Table 140: 1-Field Reference Interlace MV Table 0

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	5	3	24	502	9	48	105	8
1	12	4	25	500	9	49	506	9
2	30	5	26	57	6	50	479	9
3	18	5	27	127	8	51	503	9
4	12	5	28	39	7	52	112	8
5	52	6	29	106	7	53	477	9
6	117	7	30	113	7	54	3661	13

7	112	7	31	53	7	55	1831	12
8	0	2	32	113	8	56	914	11
9	8	4	33	104	8	57	456	10
10	27	5	34	476	9	58	459	10
11	8	5	35	39	6	59	1016	10
12	29	6	36	115	8	60	430	9
13	124	7	37	255	8	61	504	9
14	214	8	38	232	8	62	507	9
15	478	9	39	233	8	63	58574	17
16	431	9	40	126	8	64	58575	17
17	5	4	41	505	9	65	29280	16
18	27	6	42	501	9	66	29281	16
19	38	6	43	509	9	67	29282	16
20	30	6	44	62	7	68	29283	16
21	18	6	45	458	10	69	29284	16
22	118	7	46	1017	10	70	29285	16
23	77	8	47	76	8	71	29286	16

Table 141: 1-Field Reference Interlace MV Table 1

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	7	3	24	181	9	48	361	10
1	1	3	25	206	11	49	84	10
2	7	4	26	6	4	50	1147	11
3	22	5	27	68	7	51	415	12
4	1	5	28	15	7	52	11133	14
5	69	7	29	70	7	53	142	8
6	24	8	30	14	7	54	2782	12
7	694	10	31	172	8	55	1145	11
8	6	3	32	50	9	56	1390	11
9	4	4	33	55	9	57	2292	12
10	23	5	34	4587	13	58	5567	13
11	16	5	35	10	5	59	1144	11
12	41	6	36	26	8	60	9172	14
13	44	7	37	287	9	61	44529	16

14	346	9	38	22	8	62	22265	15
15	102	10	39	20	8	63	712462	20
16	414	12	40	43	9	64	712463	20
17	9	4	41	360	10	65	356224	19
18	40	6	42	85	10	66	356225	19
19	23	6	43	9173	14	67	356226	19
20	0	5	44	87	7	68	356227	19
21	42	6	45	47	9	69	356228	19
22	4	6	46	54	9	70	356229	19
23	91	8	47	46	9	71	356230	19

Table 142: 1-Field Reference Interlace MV Table 2

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	3	24	51	8	48	1574	11
1	6	4	25	497	9	49	2037	11
2	7	4	26	2	5	50	3147	12
3	13	4	27	1019	10	51	8144	13
4	7	5	28	499	9	52	4173	15
5	48	6	29	34	8	53	101	9
6	255	8	30	508	9	54	3138	12
7	496	9	31	66	9	55	201	10
8	2	2	32	1571	11	56	1575	11
9	0	4	33	131	10	57	3139	12
10	5	5	34	1568	11	58	3146	12
11	25	5	35	125	7	59	4174	15
12	30	5	36	64	9	60	8145	13
13	7	6	37	67	9	61	4175	15
14	99	7	38	996	10	62	1042	13
15	253	8	39	997	10	63	66766	19
16	35	8	40	401	11	64	66767	19
17	14	4	41	4073	12	65	33376	18
18	27	7	42	261	11	66	33377	18
19	26	7	43	520	12	67	33378	18

20	6	6	44	252	8	68	33379	18
21	9	6	45	1572	11	69	33380	18
22	24	7	46	1570	11	70	33381	18
23	197	8	47	400	11	71	33382	18

Table 143: 1-Field Reference Interlace MV Table 3

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	13	4	24	204	8	48	240	8
1	1	4	25	150	8	49	241	8
2	4	4	26	3	4	50	205	8
3	0	4	27	117	7	51	389	9
4	23	5	28	32	6	52	357	10
5	5	5	29	45	6	53	78	7
6	127	7	30	33	6	54	145	8
7	77	7	31	41	7	55	233	8
8	3	3	32	144	8	56	388	9
9	17	5	33	464	9	57	465	9
10	62	6	34	507	9	58	486	9
11	59	6	35	28	5	59	151	8
12	23	6	36	76	7	60	487	9
13	103	7	37	96	7	61	179	9
14	74	7	38	9	6	62	316	9
15	195	8	39	8	6	63	5710	14
16	242	8	40	45	7	64	5711	14
17	10	4	41	159	8	65	2848	13
18	44	6	42	506	9	66	2849	13
19	50	6	43	317	9	67	2850	13
20	61	6	44	49	6	68	2851	13
21	21	6	45	252	8	69	2852	13
22	40	7	46	88	8	70	2853	13
23	147	8	47	146	8	71	2854	13

11.4 Interlace Pictures MB Mode Tables

11.4.1 Interlace Field P / B Pictures Mixed MV MB Mode Tables

Table 144: Mixed MV MB Mode Table 0

MB Mode	VLC Codeword	VLC Size
0	16	6
1	17	6
2	3	2
3	3	3
4	0	2
5	5	4
6	9	5
7	2	2

Table 145: Mixed MV MB Mode Table 1

MB Mode	VLC Codeword	VLC Size
0	8	5
1	9	5
2	3	3
3	6	3
4	7	3
5	0	2
6	5	4
7	2	2

Table 146: Mixed MV MB Mode Table 2

MB Mode	VLC Codeword	VLC Size
0	16	6
1	17	6
2	5	4
3	3	3
4	0	2
5	3	2
6	9	5
7	2	2

Table 147: Mixed MV MB Mode Table 3

MB Mode	VLC Codeword	VLC Size
0	56	6
1	57	6
2	15	4
3	4	3
4	5	3
5	6	3

6	29	5
7	0	1

Table 148: Mixed MV MB Mode Table 4

MB Mode	VLC Codeword	VLC Size
0	52	6
1	53	6
2	27	5
3	14	4
4	15	4
5	2	2
6	12	4
7	0	1

Table 149: Mixed MV MB Mode Table 5

MB Mode	VLC Codeword	VLC Size
0	56	6
1	57	6
2	29	5
3	5	3
4	6	3
5	0	1
6	15	4
7	4	3

Table 150: Mixed MV MB Mode Table 6

MB Mode	VLC Codeword	VLC Size
0	16	5
1	17	5
2	6	3
3	7	3
4	0	2
5	1	2
6	9	4
7	5	3

Table 151: Mixed MV MB Mode Table 7

MB Mode	VLC Codeword	VLC Size
0	56	6

1	57	6
2	0	1
3	5	3
4	6	3
5	29	5
6	4	3
7	15	4

11.4.2 Interlace Field P / B Pictures 1-MV MB Mode Tables

Table 152: 1-MV MB Mode Table 0

MB Mode	VLC Codeword	VLC Size
0	0	5
1	1	5
2	1	1
3	1	3
4	1	2
5	1	4

Table 153: 1-MV MB Mode Table 1

MB Mode	VLC Codeword	VLC Size
0	0	5
1	1	5
2	1	1
3	1	2
4	1	3
5	1	4

Table 154: 1-MV MB Mode Table 2

MB Mode	VLC Codeword	VLC Size
0	16	5
1	17	5
2	3	2
3	0	1
4	9	4
5	5	3

Table 155: 1-MV MB Mode Table 3

MB Mode	VLC Codeword	VLC Size

0	20	5
1	21	5
2	3	2
3	11	4
4	0	1
5	4	3

Table 156: 1-MV MB Mode Table 4

MB Mode	VLC Codeword	VLC Size
0	4	4
1	5	4
2	2	2
3	3	3
4	3	2
5	0	2

Table 157: 1-MV MB Mode Table 5

MB Mode	VLC Codeword	VLC Size
0	4	4
1	5	4
2	3	3
3	2	2
4	0	2
5	3	2

Table 158: 1-MV MB Mode Table 6

MB Mode	VLC Codeword	VLC Size
0	0	5
1	1	5
2	1	3
3	1	4
4	1	1
5	1	2

Table 159: 1-MV MB Mode Table 7

MB Mode	VLC Codeword	VLC Size
0	16	5
1	17	5
2	9	4

3	5	3
4	3	2
5	0	1

11.4.3 Interlace Frame P Picture 4-MV MBMODE Tables

Table 160: Interlace Frame 4-MV MB Mode Table 0

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	22	5
1-MV	1	Field	17	5
1-MV	1	No CBP	0	2
1-MV	0	Frame	47	6
1-MV	0	Field	32	6
2-MV (F)	N/A	Frame	10	4
2-MV (F)	N/A	Field	1	2
2-MV (F)	N/A	No CBP	3	2
4-MV	N/A	Frame	67	7
4-MV	N/A	Field	133	8
4-MV	N/A	No CBP	132	8
4-MV (F)	N/A	Frame	92	7
4-MV (F)	N/A	Field	19	5
4-MV (F)	N/A	No CBP	93	7
INTRA	N/A	N/A	18	5

Table 161: Interlace Frame 4-MV MB Mode Table 1

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	3	3
1-MV	1	Field	45	6
1-MV	1	No CBP	0	3
1-MV	0	Frame	7	3
1-MV	0	Field	23	5
2-MV (F)	N/A	Frame	6	3
2-MV (F)	N/A	Field	1	3
2-MV (F)	N/A	No CBP	2	3

4-MV	N/A	Frame	10	4
4-MV	N/A	Field	39	6
4-MV	N/A	No CBP	44	6
4-MV (F)	N/A	Frame	8	4
4-MV (F)	N/A	Field	18	5
4-MV (F)	N/A	No CBP	77	7
INTRA	N/A	N/A	76	7

Table 162: Interlace Frame 4-MV MB Mode Table 2

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	15	4
1-MV	1	Field	6	3
1-MV	1	No CBP	28	5
1-MV	0	Frame	9	5
1-MV	0	Field	41	7
2-MV (F)	N/A	Frame	6	4
2-MV (F)	N/A	Field	2	2
2-MV (F)	N/A	No CBP	15	5
4-MV	N/A	Frame	14	5
4-MV	N/A	Field	8	5
4-MV	N/A	No CBP	40	7
4-MV (F)	N/A	Frame	29	5
4-MV (F)	N/A	Field	0	2
4-MV (F)	N/A	No CBP	21	6
INTRA	N/A	N/A	11	5

Table 163: Interlace Frame 4-MV MB Mode Table 3

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	7	4
1-MV	1	Field	198	9
1-MV	1	No CBP	1	1
1-MV	0	Frame	2	3
1-MV	0	Field	193	9

2-MV (F)	N/A	Frame	13	5
2-MV (F)	N/A	Field	25	6
2-MV (F)	N/A	No CBP	0	2
4-MV	N/A	Frame	97	8
4-MV	N/A	Field	1599	12
4-MV	N/A	No CBP	98	8
4-MV (F)	N/A	Frame	398	10
4-MV (F)	N/A	Field	798	11
4-MV (F)	N/A	No CBP	192	9
INTRA	N/A	N/A	1598	12

11.4.4 Interlace Frame P / B Pictures Non 4-MV MBMODE Tables

Table 164: Interlace Frame Non 4-MV MB Mode Table 0

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	9	4
1-MV	1	Field	22	5
1-MV	1	No CBP	0	2
1-MV	0	Frame	17	5
1-MV	0	Field	16	5
2-MV (F)	N/A	Frame	10	4
2-MV (F)	N/A	Field	1	2
2-MV (F)	N/A	No CBP	3	2
INTRA	N/A	N/A	23	5

Table 165: Interlace Frame Non 4-MV MB Mode Table 1

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	7	3
1-MV	1	Field	0	4
1-MV	1	No CBP	5	6
1-MV	0	Frame	2	2
1-MV	0	Field	1	3
2-MV (F)	N/A	Frame	1	2

2-MV (F)	N/A	Field	6	3
2-MV (F)	N/A	No CBP	3	5
INTRA	N/A	N/A	4	6

Table 166: Interlace Frame Non 4-MV MB Mode Table 2

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	1	2
1-MV	1	Field	0	2
1-MV	1	No CBP	10	4
1-MV	0	Frame	23	5
1-MV	0	Field	44	6
2-MV (F)	N/A	Frame	8	4
2-MV (F)	N/A	Field	3	2
2-MV (F)	N/A	No CBP	9	4
INTRA	N/A	N/A	45	6

Table 167: Interlace Frame Non 4-MV MB Mode Table 3

MB Type	MV Present	Field/Frame Transform	VLC Codeword	VLC Size
1-MV	1	Frame	7	4
1-MV	1	Field	97	8
1-MV	1	No CBP	1	1
1-MV	0	Frame	2	3
1-MV	0	Field	49	7
2-MV (F)	N/A	Frame	13	5
2-MV (F)	N/A	Field	25	6
2-MV (F)	N/A	No CBP	0	2
INTRA	N/A	N/A	96	8

11.5 I-Picture CBPCY Tables

Table 168: **I-Picture CBPCY VLC Table**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	1	1	32	6	4
1	23	6	33	3	9
2	9	5	34	30	7
3	5	5	35	28	6
4	6	5	36	18	7
5	71	9	37	904	12
6	32	7	38	68	9
7	16	7	39	112	9
8	2	5	40	31	6
9	124	9	41	574	11
10	58	7	42	57	8
11	29	7	43	142	9
12	2	6	44	1	7
13	236	9	45	454	11
14	119	8	46	182	9
15	0	8	47	69	9
16	3	5	48	20	6
17	183	9	49	575	11
18	44	7	50	125	9
19	19	7	51	24	9
20	1	6	52	7	7
21	360	10	53	455	11
22	70	8	54	134	9
23	63	8	55	25	9
24	30	6	56	21	6
25	1810	13	57	475	10
26	181	9	58	2	9
27	66	8	59	70	9
28	34	7	60	13	8
29	453	11	61	1811	13
30	286	10	62	474	10
31	135	9	63	361	10

11.6 P and B-Picture CBPCY Tables

Table 169: **P and B-Picture CBPCY VLC Table 0**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	1	6
1	6	13	33	7	13
2	15	7	34	1	8
3	13	13	35	7	12
4	13	7	36	14	7
5	11	13	37	12	13
6	3	13	38	4	13
7	13	12	39	14	12
8	5	6	40	1	7
9	8	13	41	9	13
10	49	7	42	97	8
11	10	12	43	11	12
12	12	6	44	7	5
13	114	8	45	58	7
14	102	8	46	52	7
15	119	8	47	62	7
16	1	5	48	4	6
17	54	7	49	103	8
18	96	8	50	1	13
19	8	12	51	9	12
20	10	6	52	11	6
21	111	8	53	56	7
22	5	13	54	101	8
23	15	12	55	118	8
24	12	7	56	4	5
25	10	13	57	110	8
26	2	13	58	100	8
27	12	12	59	30	6
28	13	6	60	2	3

29	115	8	61	5	3
30	53	7	62	4	3
31	63	7	63	3	2

Table 170: **P and B-Picture CBPCY VLC Table 1**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	14	32	1	3
1	9	13	33	240	8
2	1	8	34	1	14
3	18	13	35	19	13
4	5	3	36	18	5
5	14	13	37	15	13
6	237	8	38	4	13
7	26	13	39	27	13
8	3	3	40	1	4
9	121	7	41	122	7
10	3	8	42	2	13
11	22	13	43	23	13
12	13	4	44	1	6
13	16	13	45	17	13
14	6	13	46	7	13
15	30	13	47	31	13
16	2	3	48	1	5
17	10	13	49	11	13
18	1	13	50	2	8
19	20	13	51	21	13
20	12	4	52	19	5
21	241	8	53	246	8
22	5	13	54	238	8
23	28	13	55	29	13
24	16	5	56	17	5
25	12	13	57	13	13
26	3	13	58	236	8
27	24	13	59	25	13
28	28	5	60	58	6

29	124	7	61	63	6
30	239	8	62	8	13
31	247	8	63	125	7

Table 171: **P and B-Picture CBPCY VLC Table 2**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	1	5
1	201	8	33	102	7
2	25	6	34	1	8
3	231	8	35	415	9
4	5	4	36	24	6
5	221	8	37	3	13
6	1	13	38	2	13
7	3	12	39	244	8
8	2	4	40	3	4
9	414	9	41	54	6
10	2	8	42	3	8
11	241	8	43	484	9
12	16	5	44	17	5
13	225	8	45	114	7
14	195	8	46	200	8
15	492	9	47	493	9
16	2	5	48	3	5
17	412	9	49	413	9
18	1	10	50	1	9
19	240	8	51	4	13
20	7	4	52	13	5
21	224	8	53	113	7
22	98	7	54	99	7
23	245	8	55	485	9
24	1	6	56	4	4
25	220	8	57	111	7
26	96	7	58	194	8
27	5	13	59	243	8
28	9	4	60	5	3

29	230	8	61	29	5
30	101	7	62	26	5
31	247	8	63	31	5

Table 172: **P and B-Picture CBPCY VLC Table 3**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	9	32	1	2
1	28	9	33	29	9
2	12	9	34	13	9
3	44	9	35	45	9
4	3	2	36	5	9
5	36	9	37	37	9
6	20	9	38	21	9
7	52	9	39	53	9
8	2	2	40	2	9
9	32	9	41	33	9
10	16	9	42	17	9
11	48	9	43	49	9
12	8	9	44	9	9
13	40	9	45	41	9
14	24	9	46	25	9
15	28	8	47	29	8
16	1	3	48	1	9
17	30	9	49	31	9
18	14	9	50	15	9
19	46	9	51	47	9
20	6	9	52	7	9
21	38	9	53	39	9
22	22	9	54	23	9
23	54	9	55	55	9
24	3	9	56	4	9
25	34	9	57	35	9
26	18	9	58	19	9
27	50	9	59	51	9
28	10	9	60	11	9

29	42	9	61	43	9
30	26	9	62	27	9
31	30	8	63	31	8

11.7 DC Differential Tables

11.7.1 Low-motion Tables

Table 173: **Low-motion Luma DC Differential VLC Table**

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	1	1	40	151	14	80	197608	23
1	1	2	41	384	14	81	197609	23
2	1	4	42	788	15	82	197610	23
3	1	5	43	789	15	83	197611	23
4	5	5	44	1541	16	84	197612	23
5	7	5	45	1540	16	85	197613	23
6	8	6	46	1542	16	86	197614	23
7	12	6	47	3086	17	87	197615	23
8	0	7	48	197581	23	88	197616	23
9	2	7	49	197577	23	89	197617	23
10	18	7	50	197576	23	90	197618	23
11	26	7	51	197578	23	91	197619	23
12	3	8	52	197579	23	92	197620	23
13	7	8	53	197580	23	93	197621	23
14	39	8	54	197582	23	94	197622	23
15	55	8	55	197583	23	95	197623	23
16	5	9	56	197584	23	96	197624	23
17	76	9	57	197585	23	97	197625	23
18	108	9	58	197586	23	98	197626	23
19	109	9	59	197587	23	99	197627	23
20	8	10	60	197588	23	100	197628	23
21	25	10	61	197589	23	101	197629	23
22	155	10	62	197590	23	102	197630	23
23	27	10	63	197591	23	103	197631	23
24	154	10	64	197592	23	104	395136	24

25	19	11	65	197593	23	105	395137	24
26	52	11	66	197594	23	106	395138	24
27	53	11	67	197595	23	107	395139	24
28	97	12	68	197596	23	108	395140	24
29	72	13	69	197597	23	109	395141	24
30	196	13	70	197598	23	110	395142	24
31	74	13	71	197599	23	111	395143	24
32	198	13	72	197600	23	112	395144	24
33	199	13	73	197601	23	113	395145	24
34	146	14	74	197602	23	114	395146	24
35	395	14	75	197603	23	115	395147	24
36	147	14	76	197604	23	116	395148	24
37	387	14	77	197605	23	117	395149	24
38	386	14	78	197606	23	118	395150	24
39	150	14	79	197607	23	ESCAPE	395151	24

Table 174: **Low-motion Color-difference DC Differential VLC Table**

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	0	2	40	1630	11	80	3163240	22
1	1	2	41	3256	12	81	3163241	22
2	5	3	42	3088	12	82	3163242	22
3	9	4	43	3257	12	83	3163243	22
4	13	4	44	6179	13	84	3163244	22
5	17	5	45	12357	14	85	3163245	22
6	29	5	46	24713	15	86	3163246	22
7	31	5	47	49424	16	87	3163247	22
8	33	6	48	3163208	22	88	3163248	22
9	49	6	49	3163209	22	89	3163249	22
10	56	6	50	3163210	22	90	3163250	22
11	51	6	51	3163211	22	91	3163251	22
12	57	6	52	3163212	22	92	3163252	22
13	61	6	53	3163213	22	93	3163253	22
14	97	7	54	3163214	22	94	3163254	22
15	121	7	55	3163215	22	95	3163255	22

16	128	8	56	3163216	22	96	3163256	22
17	200	8	57	3163217	22	97	3163257	22
18	202	8	58	3163218	22	98	3163258	22
19	240	8	59	3163219	22	99	3163259	22
20	129	8	60	3163220	22	100	3163260	22
21	192	8	61	3163221	22	101	3163261	22
22	201	8	62	3163222	22	102	3163262	22
23	263	9	63	3163223	22	103	3163263	22
24	262	9	64	3163224	22	104	6326400	23
25	406	9	65	3163225	22	105	6326401	23
26	387	9	66	3163226	22	106	6326402	23
27	483	9	67	3163227	22	107	6326403	23
28	482	9	68	3163228	22	108	6326404	23
29	522	10	69	3163229	22	109	6326405	23
30	523	10	70	3163230	22	110	6326406	23
31	1545	11	71	3163231	22	111	6326407	23
32	1042	11	72	3163232	22	112	6326408	23
33	1043	11	73	3163233	22	113	6326409	23
34	1547	11	74	3163234	22	114	6326410	23
35	1041	11	75	3163235	22	115	6326411	23
36	1546	11	76	3163236	22	116	6326412	23
37	1631	11	77	3163237	22	117	6326413	23
38	1040	11	78	3163238	22	118	6326414	23
39	1629	11	79	3163239	22	ESCAPE	6326415	23

11.7.2 High-motion Tables

Table 175: **High-motion Luma DC Differential VLC Table**

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	2	2	40	824	12	80	1993024	26
1	3	2	41	829	12	81	1993025	26
2	3	3	42	171	13	82	1993026	26
3	2	4	43	241	13	83	1993027	26
4	5	4	44	1656	13	84	1993028	26

5	1	5	45	242	13	85	1993029	26
6	3	5	46	480	14	86	1993030	26
7	8	5	47	481	14	87	1993031	26
8	0	6	48	340	14	88	1993032	26
9	5	6	49	3314	14	89	1993033	26
10	13	6	50	972	15	90	1993034	26
11	15	6	51	683	15	91	1993035	26
12	19	6	52	6631	15	92	1993036	26
13	8	7	53	974	15	93	1993037	26
14	24	7	54	6630	15	94	1993038	26
15	28	7	55	1364	16	95	1993039	26
16	36	7	56	1951	16	96	1993040	26
17	4	8	57	1365	16	97	1993041	26
18	6	8	58	3901	17	98	1993042	26
19	18	8	59	3895	17	99	1993043	26
20	50	8	60	3900	17	100	1993044	26
21	59	8	61	3893	17	101	1993045	26
22	74	8	62	7789	18	102	1993046	26
23	75	8	63	7784	18	103	1993047	26
24	11	9	64	15576	19	104	1993048	26
25	38	9	65	15571	19	105	1993049	26
26	39	9	66	15577	19	106	1993050	26
27	102	9	67	31140	20	107	1993051	26
28	116	9	68	996538	25	108	1993052	26
29	117	9	69	996532	25	109	1993053	26
30	20	10	70	996533	25	110	1993054	26
31	28	10	71	996534	25	111	1993055	26
32	31	10	72	996535	25	112	1993056	26
33	29	10	73	996536	25	113	1993057	26
34	43	11	74	996537	25	114	1993058	26
35	61	11	75	996539	25	115	1993059	26
36	413	11	76	996540	25	116	1993060	26
37	415	11	77	996541	25	117	1993061	26
38	84	12	78	996542	25	118	1993062	26
39	825	12	79	996543	25	ESCAPE	1993063	26

Table 176: **High-motion Color-difference DC Differential VLC Table**

DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size	DC Differential	VLC Codeword	VLC Size
0	0	2	40	51124	16	80	13087336	24
1	1	2	41	51125	16	81	13087337	24
2	4	3	42	25566	15	82	13087338	24
3	7	3	43	51127	16	83	13087339	24
4	11	4	44	51128	16	84	13087340	24
5	13	4	45	51129	16	85	13087341	24
6	21	5	46	102245	17	86	13087342	24
7	40	6	47	204488	18	87	13087343	24
8	48	6	48	13087304	24	88	13087344	24
9	50	6	49	13087305	24	89	13087345	24
10	82	7	50	13087306	24	90	13087346	24
11	98	7	51	13087307	24	91	13087347	24
12	102	7	52	13087308	24	92	13087348	24
13	166	8	53	13087309	24	93	13087349	24
14	198	8	54	13087310	24	94	13087350	24
15	207	8	55	13087311	24	95	13087351	24
16	335	9	56	13087312	24	96	13087352	24
17	398	9	57	13087313	24	97	13087353	24
18	412	9	58	13087314	24	98	13087354	24
19	669	10	59	13087315	24	99	13087355	24
20	826	10	60	13087316	24	100	13087356	24
21	1336	11	61	13087317	24	101	13087357	24
22	1596	11	62	13087318	24	102	13087358	24
23	1598	11	63	13087319	24	103	13087359	24
24	1599	11	64	13087320	24	104	26174592	25
25	1654	11	65	13087321	24	105	26174593	25
26	2675	12	66	13087322	24	106	26174594	25
27	3194	12	67	13087323	24	107	26174595	25
28	3311	12	68	13087324	24	108	26174596	25
29	5349	13	69	13087325	24	109	26174597	25
30	6621	13	70	13087326	24	110	26174598	25
31	10696	14	71	13087327	24	111	26174599	25

32	10697	14	72	13087328	24	112	26174600	25
33	25565	15	73	13087329	24	113	26174601	25
34	13240	14	74	13087330	24	114	26174602	25
35	13241	14	75	13087331	24	115	26174603	25
36	51126	16	76	13087332	24	116	26174604	25
37	25560	15	77	13087333	24	117	26174605	25
38	25567	15	78	13087334	24	118	26174606	25
39	51123	16	79	13087335	24	ESCAPE	26174607	25

11.8 Transform AC Coefficient Tables

11.8.1 High Motion Intra Tables

Table 177: **High Motion Intra VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	1	2	62	7920	15	124	9183	14
1	5	3	63	61	6	125	25	5
2	13	4	64	83	9	126	40	9
3	18	5	65	416	11	127	374	11
4	14	6	66	726	13	128	1181	13
5	21	7	67	3848	14	129	9181	14
6	19	8	68	19	7	130	48	6
7	63	8	69	124	9	131	162	10
8	75	9	70	1985	11	132	751	12
9	287	9	71	1196	14	133	1464	14
10	184	10	72	27	7	134	63	6
11	995	10	73	160	10	135	165	10
12	370	11	74	836	12	136	987	12
13	589	12	75	3961	14	137	2367	14
14	986	12	76	121	7	138	68	7
15	733	13	77	993	10	139	1995	11
16	8021	13	78	724	13	140	2399	15
17	1465	14	79	8966	14	141	99	7
18	16046	14	80	33	8	142	963	12
19	0	4	81	572	10	143	21	8
20	16	5	82	4014	12	144	2294	12

21	8	7	83	9182	14	145	23	8
22	32	8	84	53	8	146	1176	13
23	41	9	85	373	11	147	44	8
24	500	9	86	1971	13	148	1970	13
25	563	10	87	197	8	149	47	8
26	480	11	88	372	11	150	8020	13
27	298	12	89	1925	13	151	141	8
28	989	12	90	72	9	152	1981	13
29	1290	13	91	419	11	153	142	8
30	7977	13	92	1182	13	154	4482	13
31	2626	14	93	44	9	155	251	8
32	4722	15	94	250	10	156	1291	13
33	5943	15	95	2006	11	157	45	8
34	3	5	96	146	10	158	1984	11
35	17	7	97	1484	13	159	121	9
36	196	8	98	7921	15	160	8031	13
37	75	10	99	163	10	161	122	9
38	180	11	100	1005	12	162	8022	13
39	2004	11	101	2366	14	163	561	10
40	837	12	102	482	11	164	996	10
41	727	13	103	4723	15	165	417	11
42	1983	13	104	1988	11	166	323	11
43	2360	14	105	5255	15	167	503	11
44	3003	14	106	657	12	168	367	12
45	2398	15	107	659	12	169	658	12
46	19	5	108	3978	12	170	743	12
47	120	7	109	1289	13	171	364	12
48	105	9	110	1288	13	172	365	12
49	562	10	111	1933	13	173	988	12
50	1121	11	112	1982	13	174	3979	12
51	1004	12	113	1932	13	175	1177	13
52	1312	13	114	1198	14	176	984	12
53	7978	13	115	3002	14	177	1934	13
54	15952	14	116	8967	14	178	725	13
55	15953	14	117	2970	14	179	8030	13

56	5254	15	118	5942	15	180	7979	13
57	12	6	119	14	4	181	1935	13
58	36	9	120	69	7	182	1197	14
59	148	11	121	499	9	183	16047	14
60	2240	12	122	1146	11	184	9180	14
61	3849	14	123	1500	13	ESCAPE	74	9

Table 178: **High Motion Intra Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	40	2	7	80	9	1
1	0	2	41	2	8	81	9	2
2	0	3	42	2	9	82	9	3
3	0	4	43	2	10	83	9	4
4	0	5	44	2	11	84	10	1
5	0	6	45	2	12	85	10	2
6	0	7	46	3	1	86	10	3
7	0	8	47	3	2	87	11	1
8	0	9	48	3	3	88	11	2
9	0	10	49	3	4	89	11	3
10	0	11	50	3	5	90	12	1
11	0	12	51	3	6	91	12	2
12	0	13	52	3	7	92	12	3
13	0	14	53	3	8	93	13	1
14	0	15	54	3	9	94	13	2
15	0	16	55	3	10	95	13	3
16	0	17	56	3	11	96	14	1
17	0	18	57	4	1	97	14	2
18	0	19	58	4	2	98	14	3
19	1	1	59	4	3	99	15	1
20	1	2	60	4	4	100	15	2
21	1	3	61	4	5	101	15	3
22	1	4	62	4	6	102	16	1
23	1	5	63	5	1	103	16	2
24	1	6	64	5	2	104	17	1
25	1	7	65	5	3	105	17	2

26	1	8	66	5	4	106	18	1
27	1	9	67	5	5	107	19	1
28	1	10	68	6	1	108	20	1
29	1	11	69	6	2	109	21	1
30	1	12	70	6	3	110	22	1
31	1	13	71	6	4	111	23	1
32	1	14	72	7	1	112	24	1
33	1	15	73	7	2	113	25	1
34	2	1	74	7	3	114	26	1
35	2	2	75	7	4	115	27	1
36	2	3	76	8	1	116	28	1
37	2	4	77	8	2	117	29	1
38	2	5	78	8	3	118	30	1
39	2	6	79	8	4			

Table 179: **High Motion Intra Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
119	0	1	141	5	1	163	16	1
120	0	2	142	5	2	164	17	1
121	0	3	143	6	1	165	18	1
122	0	4	144	6	2	166	19	1
123	0	5	145	7	1	167	20	1
124	0	6	146	7	2	168	21	1
125	1	1	147	8	1	169	22	1
126	1	2	148	8	2	170	23	1
127	1	3	149	9	1	171	24	1
128	1	4	150	9	2	172	25	1
129	1	5	151	10	1	173	26	1
130	2	1	152	10	2	174	27	1
131	2	2	153	11	1	175	28	1
132	2	3	154	11	2	176	29	1
133	2	4	155	12	1	177	30	1
134	3	1	156	12	2	178	31	1
135	3	2	157	13	1	179	32	1
136	3	3	158	13	2	180	33	1

137	3	4	159	14	1	181	34	1
138	4	1	160	14	2	182	35	1
139	4	2	161	15	1	183	36	1
140	4	3	162	15	2	184	37	1

Table 180: **High Motion Intra Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	19	16	2
1	15	17	2
2	12	18	1
3	11	19	1
4	6	20	1
5	5	21	1
6	4	22	1
7	4	23	1
8	4	24	1
9	4	25	1
10	3	26	1
11	3	27	1
12	3	28	1
13	3	29	1
14	3	30	1
15	3		

Table 181: **High Motion Intra Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	6	19	1
1	5	20	1
2	4	21	1
3	4	22	1
4	3	23	1
5	2	24	1
6	2	25	1
7	2	26	1

8	2	27	1
9	2	28	1
10	2	29	1
11	2	30	1
12	2	31	1
13	2	32	1
14	2	33	1
15	2	34	1
16	1	35	1
17	1	36	1
18	1	37	1

Table 182: **High Motion Intra Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	30	11	3
2	17	12	2
3	15	13	1
4	9	14	1
5	5	15	1
6	4	16	0
7	3	17	0
8	3	18	0
9	3	19	0
10	3		

Table 183: **High Motion Intra Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	37
2	15
3	4
4	3
5	1
6	0

Table 184: **High Motion Inter VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	3	57	4188	13	113	13	4
1	3	4	58	14834	14	114	173	9
2	11	5	59	88	7	115	2086	12
3	20	6	60	543	10	116	11596	14
4	63	6	61	3710	12	117	17	5
5	93	7	62	14847	14	118	363	9
6	162	8	63	35	8	119	2943	12
7	172	9	64	739	10	120	20900	15
8	366	9	65	1253	13	121	25	5
9	522	10	66	11840	14	122	539	10
10	738	10	67	161	8	123	5885	13
11	1074	11	68	1470	11	124	29	5
12	1481	11	69	2504	14	125	916	10
13	2087	12	70	131	8	126	10451	14
14	2900	12	71	314	11	127	43	6
15	1254	13	72	5921	13	128	1468	11
16	4191	13	73	68	9	129	23194	15
17	5930	13	74	630	12	130	47	6
18	8370	14	75	14838	14	131	583	12
19	11598	14	76	139	10	132	16	7
20	14832	14	77	1263	13	133	2613	12
21	16757	15	78	23195	15	134	62	6
22	23198	15	79	520	10	135	2938	12
23	4	4	80	7422	13	136	89	7
24	30	5	81	921	10	137	4190	13
25	66	7	82	7348	13	138	38	8
26	182	8	83	926	10	139	2511	14
27	371	9	84	14835	14	140	85	8
28	917	10	85	1451	11	141	7349	13
29	1838	11	86	29667	15	142	87	8
30	2964	12	87	1847	11	143	3675	12

31	5796	13	88	23199	15	144	160	8
32	8371	14	89	2093	12	145	5224	13
33	11845	14	90	3689	12	146	368	9
34	5	5	91	3688	12	147	144	10
35	64	7	92	1075	11	148	462	9
36	73	9	93	2939	12	149	538	10
37	655	10	94	11768	14	150	536	10
38	1483	11	95	11862	14	151	360	9
39	1162	13	96	11863	14	152	542	10
40	2525	14	97	14839	14	153	580	12
41	29666	15	98	20901	15	154	1846	11
42	24	5	99	3	3	155	312	11
43	37	8	100	42	6	156	1305	11
44	138	10	101	228	8	157	3678	12
45	1307	11	102	654	10	158	1836	11
46	3679	12	103	1845	11	159	2901	12
47	2505	14	104	4184	13	160	2524	14
48	5020	15	105	7418	13	161	8379	14
49	41	6	106	11769	14	162	1164	13
50	79	9	107	16756	15	163	5923	13
51	1042	11	108	9	4	164	11844	14
52	1165	13	109	84	8	165	5797	13
53	11841	14	110	920	10	166	1304	11
54	56	6	111	1163	13	167	14846	14
55	270	9	112	5021	15	ESCAPE	361	9
56	1448	11						

Table 185: **High Motion Inter Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	33	1	11	66	7	4
1	0	2	34	2	1	67	8	1
2	0	3	35	2	2	68	8	2
3	0	4	36	2	3	69	8	3
4	0	5	37	2	4	70	9	1
5	0	6	38	2	5	71	9	2

6	0	7	39	2	6	72	9	3
7	0	8	40	2	7	73	10	1
8	0	9	41	2	8	74	10	2
9	0	10	42	3	1	75	10	3
10	0	11	43	3	2	76	11	1
11	0	12	44	3	3	77	11	2
12	0	13	45	3	4	78	11	3
13	0	14	46	3	5	79	12	1
14	0	15	47	3	6	80	12	2
15	0	16	48	3	7	81	13	1
16	0	17	49	4	1	82	13	2
17	0	18	50	4	2	83	14	1
18	0	19	51	4	3	84	14	2
19	0	20	52	4	4	85	15	1
20	0	21	53	4	5	86	15	2
21	0	22	54	5	1	87	16	1
22	0	23	55	5	2	88	16	2
23	1	1	56	5	3	89	17	1
24	1	2	57	5	4	90	18	1
25	1	3	58	5	5	91	19	1
26	1	4	59	6	1	92	20	1
27	1	5	60	6	2	93	21	1
28	1	6	61	6	3	94	22	1
29	1	7	62	6	4	95	23	1
30	1	8	63	7	1	96	24	1
31	1	9	64	7	2	97	25	1
32	1	10	65	7	3	98	26	1

Table 186: **High Motion Inter Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
99	0	1	122	4	2	145	14	2
100	0	2	123	4	3	146	15	1
101	0	3	124	5	1	147	16	1
102	0	4	125	5	2	148	17	1
103	0	5	126	5	3	149	18	1

104	0	6	127	6	1	150	19	1
105	0	7	128	6	2	151	20	1
106	0	8	129	6	3	152	21	1
107	0	9	130	7	1	153	22	1
108	1	1	131	7	2	154	23	1
109	1	2	132	8	1	155	24	1
110	1	3	133	8	2	156	25	1
111	1	4	134	9	1	157	26	1
112	1	5	135	9	2	158	27	1
113	2	1	136	10	1	159	28	1
114	2	2	137	10	2	160	29	1
115	2	3	138	11	1	161	30	1
116	2	4	139	11	2	162	31	1
117	3	1	140	12	1	163	32	1
118	3	2	141	12	2	164	33	1
119	3	3	142	13	1	165	34	1
120	3	4	143	13	2	166	35	1
121	4	1	144	14	1	167	36	1

Table 187: **High Motion Inter Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	23	14	2
1	11	15	2
2	8	16	2
3	7	17	1
4	5	18	1
5	5	19	1
6	4	20	1
7	4	21	1
8	3	22	1
9	3	23	1
10	3	24	1
11	3	25	1
12	2	26	1
13	2		

Table 188: **High Motion Inter Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	9	19	1
1	5	20	1
2	4	21	1
3	4	22	1
4	3	23	1
5	3	24	1
6	3	25	1
7	2	26	1
8	2	27	1
9	2	28	1
10	2	29	1
11	2	30	1
12	2	31	1
13	2	32	1
14	2	33	1
15	1	34	1
16	1	35	1
17	1	36	1
18	1		

Table 189: **High Motion Inter Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	26	13	0
2	16	14	0
3	11	15	0
4	7	16	0
5	5	17	0
6	3	18	0
7	3	19	0
8	2	20	0
9	1	21	0
10	1	22	0

11	1	23	0
12	0		

Table 190: **High Motion Inter Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	36
2	14
3	6
4	3
5	1
6	0
7	0
8	0
9	0

11.8.2 Low Motion Intra Tables

Table 191: **Low Motion Intra VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	1	2	45	156	12	89	18	5
1	6	3	46	317	13	90	232	8
2	15	4	47	59	6	91	76	11
3	22	5	48	28	9	92	310	13
4	32	6	49	20	11	93	57	6
5	24	7	50	2494	12	94	612	10
6	8	8	51	6	7	95	3770	12
7	154	8	52	122	9	96	0	7
8	86	9	53	400	11	97	174	10
9	318	9	54	311	13	98	2460	12
10	240	10	55	27	7	99	31	7
11	933	10	56	8	10	100	1246	11
12	119	11	57	1884	11	101	67	7
13	495	11	58	113	7	102	1244	11
14	154	12	59	215	10	103	3	8
15	93	13	60	2495	12	104	971	12

16	1	4	61	7	8	105	6	8
17	17	5	62	175	10	106	2462	12
18	2	7	63	1228	11	107	42	8
19	11	8	64	52	8	108	1521	13
20	18	9	65	613	10	109	15	8
21	470	9	66	159	12	110	2558	12
22	638	10	67	224	8	111	51	8
23	401	11	68	22	11	112	2559	12
24	234	12	69	807	12	113	152	8
25	988	12	70	21	9	114	2463	12
26	315	13	71	381	11	115	234	8
27	4	5	72	3771	12	116	316	13
28	20	7	73	20	9	117	46	8
29	158	8	74	246	10	118	402	11
30	9	10	75	484	11	119	310	9
31	428	11	76	203	10	120	106	9
32	482	11	77	2461	12	121	21	11
33	970	12	78	202	10	122	943	10
34	95	13	79	764	12	123	483	11
35	23	5	80	383	11	124	116	11
36	78	7	81	1229	11	125	235	12
37	94	9	82	765	12	126	761	12
38	243	10	83	1278	11	127	92	13
39	429	11	84	314	13	128	237	12
40	236	12	85	10	4	129	989	12
41	1520	13	86	66	7	130	806	12
42	14	6	87	467	9	131	94	13
43	225	8	88	1245	11	ESCAPE	22	7
44	932	10						

Table 192: **Low Motion Intra Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	29	2	3	57	7	3
1	0	2	30	2	4	58	8	1

2	0	3	31	2	5	59	8	2
3	0	4	32	2	6	60	8	3
4	0	5	33	2	7	61	9	1
5	0	6	34	2	8	62	9	2
6	0	7	35	3	1	63	9	3
7	0	8	36	3	2	64	10	1
8	0	9	37	3	3	65	10	2
9	0	10	38	3	4	66	10	3
10	0	11	39	3	5	67	11	1
11	0	12	40	3	6	68	11	2
12	0	13	41	3	7	69	11	3
13	0	14	42	4	1	70	12	1
14	0	15	43	4	2	71	12	2
15	0	16	44	4	3	72	12	3
16	1	1	45	4	4	73	13	1
17	1	2	46	4	5	74	13	2
18	1	3	47	5	1	75	13	3
19	1	4	48	5	2	76	14	1
20	1	5	49	5	3	77	14	2
21	1	6	50	5	4	78	15	1
22	1	7	51	6	1	79	15	2
23	1	8	52	6	2	80	16	1
24	1	9	53	6	3	81	17	1
25	1	10	54	6	4	82	18	1
26	1	11	55	7	1	83	19	1
27	2	1	56	7	2	84	20	1
28	2	2						

Table 193: **Low Motion Intra Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
85	0	1	101	5	1	117	13	1
86	0	2	102	5	2	118	13	2
87	0	3	103	6	1	119	14	1
88	0	4	104	6	2	120	15	1
89	1	1	105	7	1	121	16	1

90	1	2	106	7	2	122	17	1
91	1	3	107	8	1	123	18	1
92	1	4	108	8	2	124	19	1
93	2	1	109	9	1	125	20	1
94	2	2	110	9	2	126	21	1
95	2	3	111	10	1	127	22	1
96	3	1	112	10	2	128	23	1
97	3	2	113	11	1	129	24	1
98	3	3	114	11	2	130	25	1
99	4	1	115	12	1	131	26	1
100	4	2	116	12	2			

Table 194: **Low Motion Intra Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	16	11	3
1	11	12	3
2	8	13	3
3	7	14	2
4	5	15	2
5	4	16	1
6	4	17	1
7	3	18	1
8	3	19	1
9	3	20	1
10	3		

Table 195: **Low Motion Intra Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	4	14	1
1	4	15	1
2	3	16	1
3	3	17	1
4	2	18	1
5	2	19	1

6	2	20	1
7	2	21	1
8	2	22	1
9	2	23	1
10	2	24	1
11	2	25	1
12	2	26	1
13	2		

Table 196: **Low Motion Intra Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	20	9	1
2	15	10	1
3	13	11	1
4	6	12	0
5	4	13	0
6	3	14	0
7	3	15	0
8	2	16	0

Table 197: **Low Motion Intra Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	26
2	13
3	3
4	1

11.8.3 Low Motion Inter Tables

Table 198: **Low Motion Inter VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	4	3	50	384	11	100	4	6
1	20	5	51	1436	14	101	796	12
2	23	7	52	125	8	102	6	6

3	127	8	53	356	12	103	200	13
4	340	9	54	1901	15	104	13	6
5	498	10	55	2	9	105	474	13
6	191	11	56	397	11	106	7	6
7	101	12	57	5505	13	107	201	13
8	2730	12	58	173	8	108	1	7
9	1584	13	59	96	12	109	46	14
10	5527	13	60	3175	14	110	20	7
11	951	14	61	28	9	111	5526	13
12	11042	14	62	238	13	112	10	7
13	3046	15	63	3	9	113	2754	12
14	11	4	64	719	13	114	22	7
15	55	7	65	217	9	115	347	14
16	98	9	66	5504	13	116	21	7
17	7	11	67	2	11	117	346	14
18	358	12	68	387	11	118	15	8
19	206	13	69	87	12	119	94	15
20	5520	13	70	97	12	120	126	8
21	1526	14	71	49	11	121	171	8
22	3047	15	72	102	12	122	45	9
23	7	5	73	1585	13	123	216	9
24	109	8	74	1586	13	124	11	9
25	3	11	75	172	13	125	20	10
26	799	12	76	797	12	126	691	10
27	1522	14	77	118	12	127	499	10
28	2	6	78	58	11	128	58	10
29	97	9	79	357	12	129	0	10
30	85	12	80	3174	14	130	88	10
31	479	14	81	3	2	131	46	9
32	26	6	82	84	7	132	94	10
33	30	10	83	683	10	133	1379	11
34	2761	12	84	22	13	134	236	12
35	11043	14	85	1527	14	135	84	12
36	30	6	86	5	4	136	2753	12
37	31	10	87	248	9	137	5462	13

38	2755	12	88	2729	12	138	762	13
39	11051	14	89	95	15	139	385	11
40	6	7	90	4	4	140	5463	13
41	4	11	91	28	10	141	1437	14
42	760	13	92	5456	13	142	10915	14
43	25	7	93	4	5	143	11050	14
44	6	11	94	119	11	144	478	14
45	1597	13	95	1900	15	145	1596	13
46	87	7	96	14	5	146	207	13
47	386	11	97	10	12	147	5524	13
48	10914	14	98	12	5	ESCAPE	13	9
49	4	8	99	1378	11			

Table 199: **Low Motion Inter Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	27	2	5	54	10	3
1	0	2	28	3	1	55	11	1
2	0	3	29	3	2	56	11	2
3	0	4	30	3	3	57	11	3
4	0	5	31	3	4	58	12	1
5	0	6	32	4	1	59	12	2
6	0	7	33	4	2	60	12	3
7	0	8	34	4	3	61	13	1
8	0	9	35	4	4	62	13	2
9	0	10	36	5	1	63	14	1
10	0	11	37	5	2	64	14	2
11	0	12	38	5	3	65	15	1
12	0	13	39	5	4	66	15	2
13	0	14	40	6	1	67	16	1
14	1	1	41	6	2	68	17	1
15	1	2	42	6	3	69	18	1
16	1	3	43	7	1	70	19	1
17	1	4	44	7	2	71	20	1
18	1	5	45	7	3	72	21	1
19	1	6	46	8	1	73	22	1

20	1	7	47	8	2	74	23	1
21	1	8	48	8	3	75	24	1
22	1	9	49	9	1	76	25	1
23	2	1	50	9	2	77	26	1
24	2	2	51	9	3	78	27	1
25	2	3	52	10	1	79	28	1
26	2	4	53	10	2	80	29	1

Table 200: **Low Motion Inter Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
81	0	1	104	8	1	126	22	1
82	0	2	105	8	2	127	23	1
83	0	3	106	9	1	128	24	1
84	0	4	107	9	2	129	25	1
85	0	5	108	10	1	130	26	1
86	1	1	109	10	2	131	27	1
87	1	2	110	11	1	132	28	1
88	1	3	111	11	2	133	29	1
89	1	4	112	12	1	134	30	1
90	2	1	113	12	2	135	31	1
91	2	2	114	13	1	136	32	1
92	2	3	115	13	2	137	33	1
93	3	1	116	14	1	138	34	1
94	3	2	117	14	2	139	35	1
95	3	3	118	15	1	140	36	1
96	4	1	119	15	2	141	37	1
97	4	2	120	16	1	142	38	1
98	5	1	121	17	1	143	39	1
99	5	2	122	18	1	144	40	1
100	6	1	123	19	1	145	41	1
101	6	2	124	20	1	146	42	1
102	7	1	125	21	1	147	43	1
103	7	2						

Table 201: **Low Motion Inter Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	14	15	2
1	9	16	1
2	5	17	1
3	4	18	1
4	4	19	1
5	4	20	1
6	3	21	1
7	3	22	1
8	3	23	1
9	3	24	1
10	3	25	1
11	3	26	1
12	3	27	1
13	2	28	1
14	2	29	1

Table 202: **Low Motion Inter Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	5	22	1
1	4	23	1
2	3	24	1
3	3	25	1
4	2	26	1
5	2	27	1
6	2	28	1
7	2	29	1
8	2	30	1
9	2	31	1
10	2	32	1
11	2	33	1
12	2	34	1
13	2	35	1

14	2	36	1
15	2	37	1
16	1	38	1
17	1	39	1
18	1	40	1
19	1	41	1
20	1	42	1
21	1	43	1

Table 203: **Low Motion Inter Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	29	8	1
2	15	9	1
3	12	10	0
4	5	11	0
5	2	12	0
6	1	13	0
7	1	14	0

Table 204: **Low Motion Inter Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	43
2	15
3	3
4	1
5	0

11.8.4 Mid Rate Intra Tables

Table 205: **Mid Rate Intra VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	35	83	12	69	22	8
1	6	3	36	85	12	70	23	9

2	15	4	37	11	5	71	6	10
3	13	5	38	21	7	72	5	11
4	12	5	39	30	9	73	4	11
5	21	6	40	12	10	74	89	12
6	19	6	41	86	12	75	15	6
7	18	6	42	17	6	76	22	9
8	23	7	43	27	8	77	5	10
9	31	8	44	29	9	78	14	6
10	30	8	45	11	10	79	4	10
11	29	8	46	16	6	80	17	7
12	37	9	47	34	9	81	36	11
13	36	9	48	10	10	82	16	7
14	35	9	49	13	6	83	37	11
15	33	9	50	28	9	84	19	7
16	33	10	51	8	10	85	90	12
17	32	10	52	18	7	86	21	8
18	15	10	53	27	9	87	91	12
19	14	10	54	84	12	88	20	8
20	7	11	55	20	7	89	19	8
21	6	11	56	26	9	90	26	8
22	32	11	57	87	12	91	21	9
23	33	11	58	25	8	92	20	9
24	80	12	59	9	10	93	19	9
25	81	12	60	24	8	94	18	9
26	82	12	61	35	11	95	17	9
27	14	4	62	23	8	96	38	11
28	20	6	63	25	9	97	39	11
29	22	7	64	24	9	98	92	12
30	28	8	65	7	10	99	93	12
31	32	9	66	88	12	100	94	12
32	31	9	67	7	4	101	95	12
33	13	10	68	12	6	ESCAPE	3	7
34	34	11						

Table 206: **Mid Rate Intra Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	23	0	24	45	3	4
1	0	2	24	0	25	46	4	1
2	0	3	25	0	26	47	4	2
3	0	4	26	0	27	48	4	3
4	0	5	27	1	1	49	5	1
5	0	6	28	1	2	50	5	2
6	0	7	29	1	3	51	5	3
7	0	8	30	1	4	52	6	1
8	0	9	31	1	5	53	6	2
9	0	10	32	1	6	54	6	3
10	0	11	33	1	7	55	7	1
11	0	12	34	1	8	56	7	2
12	0	13	35	1	9	57	7	3
13	0	14	36	1	10	58	8	1
14	0	15	37	2	1	59	8	2
15	0	16	38	2	2	60	9	1
16	0	17	39	2	3	61	9	2
17	0	18	40	2	4	62	10	1
18	0	19	41	2	5	63	11	1
19	0	20	42	3	1	64	12	1
20	0	21	43	3	2	65	13	1
21	0	22	44	3	3	66	14	1
22	0	23						

Table 207: **Mid Rate Intra Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
67	0	1	79	2	2	91	10	1
68	0	2	80	3	1	92	11	1
69	0	3	81	3	2	93	12	1
70	0	4	82	4	1	94	13	1
71	0	5	83	4	2	95	14	1

72	0	6	84	5	1	96	15	1
73	0	7	85	5	2	97	16	1
74	0	8	86	6	1	98	17	1
75	1	1	87	6	2	99	18	1
76	1	2	88	7	1	100	19	1
77	1	3	89	8	1	101	20	1
78	2	1	90	9	1			

Table 208: **Mid Rate Intra Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	27	8	2
1	10	9	2
2	5	10	1
3	4	11	1
4	3	12	1
5	3	13	1
6	3	14	1
7	3		

Table 209: **Mid Rate Intra Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	8	11	1
1	3	12	1
2	2	13	1
3	2	14	1
4	2	15	1
5	2	16	1
6	2	17	1
7	1	18	1
8	1	19	1
9	1	20	1
10	1		

Table 210: **Mid Rate Intra Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	14	15	0
2	9	16	0
3	7	17	0
4	3	18	0
5	2	19	0
6	1	20	0
7	1	21	0
8	1	22	0
9	1	23	0
10	1	24	0
11	0	25	0
12	0	26	0
13	0	27	0
14	0		

Table 211: **Mid Rate Intra Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	20
2	6
3	1
4	0
5	0
6	0
7	0
8	0

11.8.5 Mid Rate Inter Tables

Table 212: **Mid Rate Inter VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	35	10	10	69	16	7

1	15	4	36	17	6	70	26	8
2	21	6	37	9	10	71	25	8
3	23	7	38	16	6	72	24	8
4	31	8	39	8	10	73	23	8
5	37	9	40	22	7	74	22	8
6	36	9	41	85	12	75	21	8
7	33	10	42	21	7	76	20	8
8	32	10	43	20	7	77	19	8
9	7	11	44	28	8	78	24	9
10	6	11	45	27	8	79	23	9
11	32	11	46	33	9	80	22	9
12	6	3	47	32	9	81	21	9
13	20	6	48	31	9	82	20	9
14	30	8	49	30	9	83	19	9
15	15	10	50	29	9	84	18	9
16	33	11	51	28	9	85	17	9
17	80	12	52	27	9	86	7	10
18	14	4	53	26	9	87	6	10
19	29	8	54	34	11	88	5	10
20	14	10	55	35	11	89	4	10
21	81	12	56	86	12	90	36	11
22	13	5	57	87	12	91	37	11
23	35	9	58	7	4	92	38	11
24	13	10	59	25	9	93	39	11
25	12	5	60	5	11	94	88	12
26	34	9	61	15	6	95	89	12
27	82	12	62	4	11	96	90	12
28	11	5	63	14	6	97	91	12
29	12	10	64	13	6	98	92	12
30	83	12	65	12	6	99	93	12
31	19	6	66	19	7	100	94	12
32	11	10	67	18	7	101	95	12
33	84	12	68	17	7	ESCAPE	3	7
34	18	6						

Table 213: **Mid Rate Inter Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	20	2	3	39	9	2
1	0	2	21	2	4	40	10	1
2	0	3	22	3	1	41	10	2
3	0	4	23	3	2	42	11	1
4	0	5	24	3	3	43	12	1
5	0	6	25	4	1	44	13	1
6	0	7	26	4	2	45	14	1
7	0	8	27	4	3	46	15	1
8	0	9	28	5	1	47	16	1
9	0	10	29	5	2	48	17	1
10	0	11	30	5	3	49	18	1
11	0	12	31	6	1	50	19	1
12	1	1	32	6	2	51	20	1
13	1	2	33	6	3	52	21	1
14	1	3	34	7	1	53	22	1
15	1	4	35	7	2	54	23	1
16	1	5	36	8	1	55	24	1
17	1	6	37	8	2	56	25	1
18	2	1	38	9	1	57	26	1
19	2	2						

Table 214: **Mid Rate Inter Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
58	0	1	73	12	1	88	27	1
59	0	2	74	13	1	89	28	1
60	0	3	75	14	1	90	29	1
61	1	1	76	15	1	91	30	1
62	1	2	77	16	1	92	31	1
63	2	1	78	17	1	93	32	1
64	3	1	79	18	1	94	33	1
65	4	1	80	19	1	95	34	1

66	5	1	81	20	1	96	35	1
67	6	1	82	21	1	97	36	1
68	7	1	83	22	1	98	37	1
69	8	1	84	23	1	99	38	1
70	9	1	85	24	1	100	39	1
71	10	1	86	25	1	101	40	1
72	11	1	87	26	1			

Table 215: **Mid Rate Inter Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	12	14	1
1	6	15	1
2	4	16	1
3	3	17	1
4	3	18	1
5	3	19	1
6	3	20	1
7	2	21	1
8	2	22	1
9	2	23	1
10	2	24	1
11	1	25	1
12	1	26	1
13	1		

Table 216: **Mid Rate Inter Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	3	21	1
1	2	22	1
2	1	23	1
3	1	24	1
4	1	25	1
5	1	26	1
6	1	27	1

7	1	28	1
8	1	29	1
9	1	30	1
10	1	31	1
11	1	32	1
12	1	33	1
13	1	34	1
14	1	35	1
15	1	36	1
16	1	37	1
17	1	38	1
18	1	39	1
19	1	40	1
20	1		

Table 217: **Mid Rate Inter Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	26	7	0
2	10	8	0
3	6	9	0
4	2	10	0
5	1	11	0
6	1	12	0

Table 218: **Mid Rate Inter Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	40
2	1
3	0

11.8.6 High Rate Intra Tables

Table 219: **High Rate Intra VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	2	54	7961	13	108	72	8

1	3	3	55	7605	13	109	1996	11
2	13	4	56	9	4	110	2721	12
3	5	4	57	16	5	111	384	9
4	28	5	58	41	6	112	1125	11
5	22	5	59	98	7	113	6405	13
6	63	6	60	243	8	114	994	10
7	58	6	61	173	8	115	3777	12
8	46	6	62	485	9	116	15515	14
9	34	6	63	377	9	117	756	10
10	123	7	64	156	9	118	2248	12
11	103	7	65	945	10	119	1985	11
12	95	7	66	686	10	120	2344	13
13	71	7	67	295	10	121	1505	11
14	38	7	68	1902	11	122	12813	14
15	239	8	69	1392	11	123	3778	12
16	205	8	70	629	11	124	25624	15
17	193	8	71	3877	12	125	7988	13
18	169	8	72	3776	12	126	120	7
19	79	8	73	2720	12	127	341	9
20	498	9	74	2263	12	128	1362	11
21	477	9	75	7756	13	129	6431	13
22	409	9	76	8	5	130	250	8
23	389	9	77	99	7	131	2012	11
24	349	9	78	175	8	132	6407	13
25	283	9	79	379	9	133	172	8
26	1007	10	80	947	10	134	585	11
27	993	10	81	2013	11	135	5041	14
28	968	10	82	1600	11	136	502	9
29	817	10	83	3981	12	137	2786	12
30	771	10	84	3009	12	138	476	9
31	753	10	85	1169	12	139	1261	12
32	672	10	86	40	6	140	388	9
33	563	10	87	195	8	141	6404	13
34	294	10	88	337	9	142	342	9
35	1984	11	89	673	10	143	2521	13

36	1903	11	90	1395	11	144	999	10
37	1900	11	91	3779	12	145	2345	13
38	1633	11	92	7989	13	146	946	10
39	1540	11	93	101	7	147	15208	14
40	1394	11	94	474	9	148	757	10
41	1361	11	95	687	10	149	5040	14
42	1130	11	96	631	11	150	802	10
43	628	11	97	2249	12	151	15209	14
44	3879	12	98	6017	13	152	564	10
45	3876	12	99	37	7	153	31029	15
46	3803	12	100	280	9	154	1991	11
47	3214	12	101	1606	11	155	51251	16
48	3083	12	102	2726	12	156	1632	11
49	3082	12	103	6016	13	157	31028	15
50	2787	12	104	201	8	158	587	11
51	2262	12	105	801	10	159	51250	16
52	1168	12	106	3995	12	160	2727	12
53	1173	12	107	6430	13	161	7960	13
						ESCAPE	122	7

Table 220: **High Rate Intra Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	42	0	43	84	2	9
1	0	2	43	0	44	85	2	10
2	0	3	44	0	45	86	3	1
3	0	4	45	0	46	87	3	2
4	0	5	46	0	47	88	3	3
5	0	6	47	0	48	89	3	4
6	0	7	48	0	49	90	3	5
7	0	8	49	0	50	91	3	6
8	0	9	50	0	51	92	3	7
9	0	10	51	0	52	93	4	1
10	0	11	52	0	53	94	4	2
11	0	12	53	0	54	95	4	3
12	0	13	54	0	55	96	4	4

13	0	14	55	0	56	97	4	5
14	0	15	56	1	1	98	4	6
15	0	16	57	1	2	99	5	1
16	0	17	58	1	3	100	5	2
17	0	18	59	1	4	101	5	3
18	0	19	60	1	5	102	5	4
19	0	20	61	1	6	103	5	5
20	0	21	62	1	7	104	6	1
21	0	22	63	1	8	105	6	2
22	0	23	64	1	9	106	6	3
23	0	24	65	1	10	107	6	4
24	0	25	66	1	11	108	7	1
25	0	26	67	1	12	109	7	2
26	0	27	68	1	13	110	7	3
27	0	28	69	1	14	111	8	1
28	0	29	70	1	15	112	8	2
29	0	30	71	1	16	113	8	3
30	0	31	72	1	17	114	9	1
31	0	32	73	1	18	115	9	2
32	0	33	74	1	19	116	9	3
33	0	34	75	1	20	117	10	1
34	0	35	76	2	1	118	10	2
35	0	36	77	2	2	119	11	1
36	0	37	78	2	3	120	11	2
37	0	38	79	2	4	121	12	1
38	0	39	80	2	5	122	12	2
39	0	40	81	2	6	123	13	1
40	0	41	82	2	7	124	13	2
41	0	42	83	2	8	125	14	1

Table 221: **High Rate Intra Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
126	0	1	138	4	1	150	10	1
127	0	2	139	4	2	151	10	2
128	0	3	140	5	1	152	11	1

129	0	4	141	5	2	153	11	2
130	1	1	142	6	1	154	12	1
131	1	2	143	6	2	155	12	2
132	1	3	144	7	1	156	13	1
133	2	1	145	7	2	157	13	2
134	2	2	146	8	1	158	14	1
135	2	3	147	8	2	159	14	2
136	3	1	148	9	1	160	15	1
137	3	2	149	9	2	161	16	1

Table 222: **High Rate Intra Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	56	8	3
1	20	9	3
2	10	10	2
3	7	11	2
4	6	12	2
5	5	13	2
6	4	14	1
7	3		

Table 223: **High Rate Intra Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	4	9	2
1	3	10	2
2	3	11	2
3	2	12	2
4	2	13	2
5	2	14	2
6	2	15	1
7	2	16	1
8	2		

Table 224: **High Rate Intra Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	14	29	0
2	13	30	0
3	9	31	0
4	6	32	0
5	5	33	0
6	4	34	0
7	3	35	0
8	2	36	0
9	2	37	0
10	2	38	0
11	1	39	0
12	1	40	0
13	1	41	0
14	1	42	0
15	1	43	0
16	1	44	0
17	1	45	0
18	1	46	0
19	1	47	0
20	1	48	0
21	0	49	0
22	0	50	0
23	0	51	0
24	0	52	0
25	0	53	0
26	0	54	0
27	0	55	0
28	0	56	0

Table 225: **High Rate Intra Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	16
2	14
3	2

11.8.7 High Rate Inter Tables

Table 226: **High Rate Inter VLC Table**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	2	2	59	7	5	118	31989	15
1	0	3	60	472	9	119	117	7
2	30	5	61	728	11	120	3364	12
3	4	5	62	7975	13	121	63977	16
4	18	6	63	13460	14	122	46	7
5	112	7	64	53	6	123	7970	13
6	26	7	65	993	10	124	33	7
7	95	8	66	1436	12	125	1359	13
8	71	8	67	14531	14	126	20	7
9	467	9	68	12	6	127	14916	14
10	181	9	69	357	10	128	228	8
11	87	9	70	7459	13	129	31991	15
12	949	10	71	5688	14	130	94	8
13	365	10	72	104	7	131	29061	15
14	354	10	73	1683	11	132	55	8
15	1998	11	74	14917	14	133	11379	15
16	1817	11	75	32	7	134	475	9
17	1681	11	76	3984	12	135	23005	16
18	710	11	77	31990	15	136	455	9
19	342	11	78	232	8	137	26923	15
20	3986	12	79	1423	12	138	422	9
21	3374	12	80	11503	15	139	22757	16
22	3360	12	81	69	8	140	180	9
23	1438	12	82	2874	13	141	127952	17
24	1128	12	83	497	9	142	176	9
25	678	12	84	15174	14	143	45513	17
26	7586	13	85	423	9	144	998	10
27	7264	13	86	5750	14	145	92016	18
28	6723	13	87	86	9	146	366	10

29	2845	13	88	26922	15	147	255906	18
30	2240	13	89	909	10	148	283	10
31	1373	13	90	58121	16	149	1023629	20
32	3	3	91	170	10	150	217	10
33	10	5	92	116241	17	151	1023631	20
34	119	7	93	735	11	152	168	10
35	229	8	94	46009	17	153	182051	19
36	473	9	95	712	11	154	1865	11
37	997	10	96	232480	18	155	929924	20
38	358	10	97	432	11	156	1686	11
39	1684	11	98	91024	18	157	364101	20
40	338	11	99	3999	12	158	734	11
41	1439	12	100	92017	18	159	728200	21
42	7996	13	101	3792	12	160	561	11
43	6731	13	102	464963	19	161	1859850	21
44	1374	13	103	3370	12	162	433	11
45	12	4	104	1023628	20	163	7439405	23
46	125	7	105	1121	12	164	3371	12
47	68	8	106	1023630	20	165	3719703	22
48	992	10	107	2919	13	166	3375	12
49	1897	11	108	1375	13	167	1456403	22
50	3633	12	109	63	6	168	1458	12
51	7974	13	110	109	9	169	1456402	22
52	1372	13	111	3728	12	170	1129	12
53	27	5	112	1358	13	171	7439404	23
54	226	8	113	19	6	172	6722	13
55	933	10	114	281	10	173	2241	13
56	713	11	115	2918	13	ESCAPE	115	7
57	7971	13	116	11	6			
58	15175	14	117	565	11			

Table 227: **High Rate Inter Indexed Run and Level Table (Last == 0)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
0	0	1	37	1	6	74	7	3
1	0	2	38	1	7	75	8	1

2	0	3	39	1	8	76	8	2
3	0	4	40	1	9	77	8	3
4	0	5	41	1	10	78	9	1
5	0	6	42	1	11	79	9	2
6	0	7	43	1	12	80	9	3
7	0	8	44	1	13	81	10	1
8	0	9	45	2	1	82	10	2
9	0	10	46	2	2	83	11	1
10	0	11	47	2	3	84	11	2
11	0	12	48	2	4	85	12	1
12	0	13	49	2	5	86	12	2
13	0	14	50	2	6	87	13	1
14	0	15	51	2	7	88	13	2
15	0	16	52	2	8	89	14	1
16	0	17	53	3	1	90	14	2
17	0	18	54	3	2	91	15	1
18	0	19	55	3	3	92	15	2
19	0	20	56	3	4	93	16	1
20	0	21	57	3	5	94	16	2
21	0	22	58	3	6	95	17	1
22	0	23	59	4	1	96	17	2
23	0	24	60	4	2	97	18	1
24	0	25	61	4	3	98	18	2
25	0	26	62	4	4	99	19	1
26	0	27	63	4	5	100	19	2
27	0	28	64	5	1	101	20	1
28	0	29	65	5	2	102	20	2
29	0	30	66	5	3	103	21	1
30	0	31	67	5	4	104	21	2
31	0	32	68	6	1	105	22	1
32	1	1	69	6	2	106	22	2
33	1	2	70	6	3	107	23	1
34	1	3	71	6	4	108	24	1
35	1	4	72	7	1			
36	1	5	73	7	2			

Table 228: **High Rate Inter Indexed Run and Level Table (Last == 1)**

Index	Run	Level	Index	Run	Level	Index	Run	Level
109	0	1	131	8	2	153	19	2
110	0	2	132	9	1	154	20	1
111	0	3	133	9	2	155	20	2
112	0	4	134	10	1	156	21	1
113	1	1	135	10	2	157	21	2
114	1	2	136	11	1	158	22	1
115	1	3	137	11	2	159	22	2
116	2	1	138	12	1	160	23	1
117	2	2	139	12	2	161	23	2
118	2	3	140	13	1	162	24	1
119	3	1	141	13	2	163	24	2
120	3	2	142	14	1	164	25	1
121	3	3	143	14	2	165	25	2
122	4	1	144	15	1	166	26	1
123	4	2	145	15	2	167	26	2
124	5	1	146	16	1	168	27	1
125	5	2	147	16	2	169	27	2
126	6	1	148	17	1	170	28	1
127	6	2	149	17	2	171	28	2
128	7	1	150	18	1	172	29	1
129	7	2	151	18	2	173	30	1
130	8	1	152	19	1			

Table 229: **High Rate Inter Delta Level Indexed by Run Table (Last == 0)**

Run	Delta Level	Run	Delta Level
0	32	13	2
1	13	14	2
2	8	15	2
3	6	16	2
4	5	17	2
5	4	18	2
6	4	19	2

7	3	20	2
8	3	21	2
9	3	22	2
10	2	23	1
11	2	24	1
12	2		

Table 230: **High Rate Inter Delta Level Indexed by Run Table (Last == 1)**

Run	Delta Level	Run	Delta Level
0	4	16	2
1	3	17	2
2	3	18	2
3	3	19	2
4	2	20	2
5	2	21	2
6	2	22	2
7	2	23	2
8	2	24	2
9	2	25	2
10	2	26	2
11	2	27	2
12	2	28	2
13	2	29	1
14	2	30	1
15	2		

Table 231: **High Rate Inter Delta Run Indexed by Level Table (Last == 0)**

Level	Delta Run	Level	Delta Run
1	24	17	0
2	22	18	0
3	9	19	0
4	6	20	0
5	4	21	0
6	3	22	0

7	2	23	0
8	2	24	0
9	1	25	0
10	1	26	0
11	1	27	0
12	1	28	0
13	1	29	0
14	0	30	0
15	0	31	0
16	0	32	0

Table 232: **High Rate Inter Delta Run Indexed by Level Table (Last == 1)**

Level	Delta Run
1	30
2	28
3	3
4	0

11.9 Zigzag Tables

11.9.1 Intra zigzag tables

Table 233: Intra Normal Scan

0	8	1	2	9	16	24	17	10	3	4	11	18	25	32	40	33	48	26	19	12	5	6	13
20	27	34	41	56	49	57	42	35	28	21	14	7	15	22	29	36	43	50	58	51	59	44	37
30	23	31	38	45	52	60	53	61	46	39	47	54	62	55	63								

Table 234: Intra Horizontal Scan

0	1	8	2	3	9	16	24	17	10	4	5	11	18	25	32	40	48	33	26	19	12	6	7
13	20	27	34	41	56	49	57	42	35	28	21	14	15	22	29	36	43	50	58	51	44	37	30
23	31	38	45	52	59	60	53	46	39	47	54	61	62	55	63								

Table 235: Intra Vertical Scan

0	8	16	1	24	32	40	9	2	3	10	17	25	48	56	41	33	26	18	11	4	5	12	19
27	34	49	57	50	42	35	28	20	13	6	7	14	21	29	36	43	51	58	59	52	44	37	30
22	15	23	31	38	45	60	53	46	39	47	54	61	62	55	63								

11.9.2 Inter zigzag tables

Table 236: **Inter 8x8 Scan for Simple and Main Profiles and Progressive Mode in Advanced Profile**

0	8	1	2	9	16	24	17	10	3	4	11	18	25	32	40	48	56	41	33	26	19	12	5
6	13	20	27	34	49	57	58	50	42	35	28	21	14	7	15	22	29	36	43	51	59	60	52
44	37	30	23	31	38	45	53	61	62	54	46	39	47	55	63								

Table 237: **Inter 8x4 Scan for Simple and Main Profiles**

0	1	2	8	3	9	10	16	4	11	17	24	18	12	5	19	25	13	20	26	27	6	21	28
14	22	29	7	30	15	23	31																

Table 238: Inter 4x8 Scan for Simple and Main Profiles

0	4	1	8	5	12	9	2	16	6	13	20	10	24	17	14	28	21	18	3	25	29	7	22
11	26	15	30	19	23	27	31																

Table 239: Inter 4x4 Scan for Simple and Main Profiles and Progressive Mode in Advanced Profile

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

Table 240: **Progressive Mode Inter 8x4 Scan for Advanced Profile**

0	8	1	16	2	9	10	3	24	17	4	11	18	12	5	19	25	13	20	26	27	6	21	28
14	22	29	7	30	15	23	31																

Table 241: **Progressive Mode Inter 4x8 Scan for Advanced Profile**

0	1	4	2	5	8	9	12	6	16	13	10	20	3	17	14	24	7	28	21	18	25	29	22
11	26	15	30	19	23	27	31																

Table 242: **Interlace Mode Inter 8x8 Scan for Advanced Profile (Also used for Intra Mode 8x8 scan for Interlace Frame Pictures)**

0	8	1	16	24	9	2	32	40	48	56	17	10	3	25	18	11	4	33	41	49	57	26	34
42	50	58	19	12	5	27	20	13	6	35	28	21	14	7	15	22	29	36	43	51	59	60	52
44	37	30	23	31	38	45	53	61	62	54	46	39	47	55	63								

Table 243: **Interlace Mode Inter 8x4 Scan for Advanced Profile**

0	8	16	24	1	9	2	17	25	10	3	18	26	4	11	19	12	5	13	20	27	6	21	28
14	22	29	7	30	15	23	31																

Table 244: **Interlace Mode Inter 4x8 Scan for Advanced Profile**

0	1	2	4	8	5	12	9	6	3	16	20	24	28	13	10	17	14	21	18	25	29	7	22
11	26	15	30	19	23	27	31																

Table 245: Interlace Mode Inter 4x4 Scan for Advanced Profile

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

11.10 Motion Vector Differential Tables

Table 246: **Motion Vector Differential VLC Table 0**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	6	25	167	10	50	21	5
1	2	7	26	49	8	51	22	5
2	3	7	27	194	10	52	39	6
3	8	8	28	195	10	53	204	9
4	576	14	29	581	14	54	103	8
5	3	6	30	582	14	55	23	5

6	2	5	31	583	14	56	24	5
7	6	6	32	292	13	57	25	5
8	5	7	33	293	13	58	104	7
9	577	14	34	294	13	59	410	10
10	578	14	35	13	6	60	105	7
11	7	6	36	2	3	61	106	7
12	8	6	37	7	5	62	107	7
13	9	6	38	24	6	63	108	7
14	40	8	39	50	8	64	109	7
15	19	9	40	102	9	65	220	8
16	37	10	41	295	13	66	411	10
17	82	9	42	13	5	67	442	9
18	21	7	43	7	4	68	222	8
19	22	7	44	8	4	69	443	9
20	23	7	45	18	5	70	446	9
21	579	14	46	50	7	71	447	9
22	580	14	47	103	9	72	7	3
23	166	10	48	38	6			
24	96	9	49	20	5			

Table 247: Motion Vector Differential VLC Table 1

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	5	25	3012	14	50	58	6
1	4	7	26	3013	14	51	163	8
2	5	7	27	3014	14	52	236	8
3	3	6	28	3015	14	53	237	8
4	4	6	29	3016	14	54	3023	14
5	3	5	30	3017	14	55	119	7
6	4	5	31	3018	14	56	120	7
7	5	6	32	3019	14	57	242	8
8	20	7	33	3020	14	58	122	7
9	6	5	34	3021	14	59	486	9
10	21	7	35	3022	14	60	1512	13
11	44	8	36	1	2	61	487	9

12	45	8	37	4	3	62	246	8
13	46	8	38	15	6	63	494	9
14	3008	14	39	160	8	64	1513	13
15	95	9	40	161	8	65	495	9
16	112	9	41	41	6	66	1514	13
17	113	9	42	6	3	67	1515	13
18	57	8	43	11	4	68	1516	13
19	3009	14	44	42	6	69	1517	13
20	3010	14	45	162	8	70	1518	13
21	116	9	46	43	6	71	1519	13
22	117	9	47	119	9	72	31	5
23	3011	14	48	56	6			
24	118	9	49	57	6			

Table 248: **Motion Vector Differential VLC Table 2**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	3	25	276	11	50	297	11
1	512	12	26	277	11	51	298	11
2	513	12	27	278	11	52	299	11
3	514	12	28	279	11	53	300	11
4	515	12	29	280	11	54	301	11
5	2	3	30	281	11	55	302	11
6	3	4	31	282	11	56	303	11
7	258	11	32	283	11	57	304	11
8	259	11	33	284	11	58	305	11
9	260	11	34	285	11	59	306	11
10	261	11	35	286	11	60	307	11
11	262	11	36	1	1	61	308	11
12	263	11	37	5	5	62	309	11
13	264	11	38	287	11	63	310	11
14	265	11	39	288	11	64	311	11
15	266	11	40	289	11	65	312	11
16	267	11	41	290	11	66	313	11
17	268	11	42	6	4	67	314	11

18	269	11	43	7	4	68	315	11
19	270	11	44	291	11	69	316	11
20	271	11	45	292	11	70	317	11
21	272	11	46	293	11	71	318	11
22	273	11	47	294	11	72	319	11
23	274	11	48	295	11			
24	275	11	49	296	11			

Table 249: **Motion Vector Differential VLC Table 3**

Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size	Index	VLC Codeword	VLC Size
0	0	15	25	6	10	50	5	3
1	1	11	26	14	11	51	18	5
2	1	15	27	8	10	52	29	6
3	2	15	28	106	15	53	152	8
4	3	15	29	107	15	54	77	7
5	4	15	30	108	15	55	24	5
6	1	12	31	15	11	56	25	5
7	5	15	32	109	15	57	26	5
8	4	12	33	9	10	58	39	6
9	3	11	34	55	14	59	108	7
10	5	12	35	10	10	60	13	9
11	8	12	36	1	4	61	109	7
12	6	15	37	2	4	62	55	6
13	9	12	38	1	5	63	56	6
14	10	12	39	2	7	64	57	6
15	11	12	40	3	8	65	116	7
16	12	12	41	12	9	66	11	10
17	7	15	42	6	5	67	153	8
18	104	15	43	2	3	68	234	8
19	14	12	44	6	4	69	235	8
20	105	15	45	7	5	70	118	7
21	4	10	46	28	6	71	119	7
22	10	11	47	7	8	72	15	4
23	15	12	48	15	5			

24	11	11	49	8	4			

12 Bibliography

J. Ribas-Corbera, P.A. Chou, and S.L. Regunathan, "A generalized hypothetical reference decoder for H.264/AVC," IEEE Transactions on Circuits and Systems for Video Technology, Aug. 2003.

ISO/IEC 13818-2:2000, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video (MPEG-2/H.262), Annex C "Video Buffering Verifier," 2nd Edition, 2000.

J. Taylor, *DVD De-Mystified*, 2nd Ed McGraw-Hill, 2001.

Recommendation ITU-T H.263, "Video Coding for Low Bit Rate Communication", Annex B "Hypothetical Reference Decoder," Jan 1998.

SMPTE RP 227, "SMPTE Recommended Practice: VC-1 Bitstream Transport Encodings".

SMPTE RP 186 – 1995, "Video Index Information Coding for 525- and 625- Line Television Systems."

SMPTE 328M – 2000, "Television - MPEG-2 Video Elementary Stream Editing Information."

SMPTE Registration Authority "http://www.smpte-ra.org/ "

Tien-Ying Kuo and C.-C. Jay Kuo, "Motion-compensated interpolation for low-bit-rate video quality enhancement," Conference on Applications of Digital Image Processing XXI, SPIE's Annual Meeting, San Diego, CA, July 19-24, 1998.

Recommendation ITU-R BT.1358, "Studio parameters of 625 and 525 line progressive scan television systems", 1998.

Recommendation ITU-T H.320, "Narrow-band visual telephone systems and terminal equipment", March 2004.

Recommendation ITU-T H.264 and ISO/IEC 14496-10, "Advanced video coding for generic audiovisual services", May 2003.

Annex A
Transform Specification

A.1 Inverse Transform

The inverse transform is similar, but not identical, to an IDCT. The transform matrices defined in Figure 157 and Figure 158 shall be used for an 8 point inverse transformation and a 4 point inverse transformation respectively. The formula in Figure 159 shall be used to compute the inverse transform of 8x8, 8x4, 4x8, and 4x4 blocks.

$$T_8 = \begin{bmatrix} 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 16 & 15 & 9 & 4 & -4 & -9 & -15 & -16 \\ 16 & 6 & -6 & -16 & -16 & -6 & 6 & 16 \\ 15 & -4 & -16 & -9 & 9 & 16 & 4 & -15 \\ 12 & -12 & -12 & 12 & 12 & -12 & -12 & 12 \\ 9 & -16 & 4 & 15 & -15 & -4 & 16 & -9 \\ 6 & -16 & 16 & -6 & -6 & 16 & -16 & 6 \\ 4 & -9 & 15 & -16 & 16 & -15 & 9 & -4 \end{bmatrix}$$

Figure 157: Matrix for 1-D 8-point Inverse Transform

$$T_4 = \begin{bmatrix} 17 & 17 & 17 & 17 \\ 22 & 10 & -10 & -22 \\ 17 & -17 & -17 & 17 \\ 10 & -22 & 22 & -10 \end{bmatrix}$$

Figure 158: Matrix for 1-D 4-point Inverse Transform

$$E_{M\times N} = (D_{M\times N} \cdot T_M + 4) >> 3$$

$$\underline{R}_{M\times N} = (T'_N \cdot E_{M\times N} + C_N \cdot 1_M + 64) >> 7$$

where

$M, N \in \{4, 8\}$

operator '·' represents matrix multiplication

operator ' >> ' represents arithmetic right shift, performed entry-wise on a matrix

operator " ' " represents matrix transpose

T_8 is a constant matrix as in Figure 157

T_4 is a constant matrix as in Figure 158

$C_8 = (0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1)'$
C_4 is a zero column vector of length 4
1_M is an M length row vector of ones
$D_{8\times8}, D_{8\times4}, D_{4\times8}, D_{4\times4}$ are inverse quantized transform coefficient blocks (input). The inverse quantized transform coefficient values shall not exceed the signed 12 bit range, i.e. entries of these matrices shall be limited to the range ≥ -2048 and < 2047.
$E_{8\times8}, E_{8\times4}, E_{4\times8}, E_{4\times4}$ are intermediate matrices. The values in the intermediate matrices shall be limited to the range ≥-4096 and <4095.
$R_{8\times8}, R_{8\times4}, R_{4\times8}, R_{4\times4}$ are inverse transformed blocks (output). The inverse transformed values shall be limited to the range ≥-512 and <511.

Figure 159: Definition of Inverse Transform

A.2 Forward Transform (Informative)

The forward transform can be implemented in scaled integer arithmetic or using floating point or other representations. The matrix-multiplication representation of the forward transform shown below is purely an analytical representation unlike for the inverse transform where the matrix multiplies specifically referred to integer multiplications with 16 bit registers. Rounding between stages can be done as necessary and this choice is left to the encoder.

The 4x4, 4x8, 8x4 and 8x8 transforms of the data matrix D can be calculated using the following set of equations for these four cases:

$$\hat{D} = (T_4 \, D \, T_4') \circ N_4$$
$$\hat{D} = (T_8 \, D \, T_4') \circ N_4$$
$$\hat{D} = (T_4 \, D \, T_8') \circ N_8$$
$$\hat{D} = (T_8 \, D \, T_8') \circ N_8$$

where the operator ∘ is a component-wise multiplication, and $\hat{D}$ represents the transform coefficient matrix. The normalization matrices N_{ij} are given by:

$$N_{ij} = c_j \, c_i'$$

where i and j represent rows and columns of the normalization matrix, and the column vectors c are

$$c_4 = \left(\frac{8}{289} \quad \frac{8}{292} \quad \frac{8}{289} \quad \frac{8}{292} \right)'$$

$$c_8 = \left(\frac{8}{288} \quad \frac{8}{289} \quad \frac{8}{292} \quad \frac{8}{289} \quad \frac{8}{288} \quad \frac{8}{289} \quad \frac{8}{292} \quad \frac{8}{289} \right)'$$

Annex B
Spatial Alignment of Video Samples in Variable Resolution Coding

B.1 Spatial alignment of samples in down-sampled frame

The samples of the down-sampled frame have the following spatial alignment with respect to the samples of the frame at the original resolution.

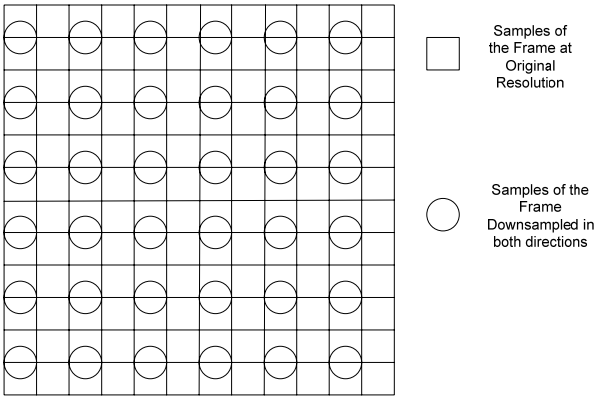


Figure 160: Relative Spatial Alignment of the video samples of the Down-sampled Frame, and video samples of the Original Frame.

The following definitions are used for the down-sampling pseudo-code examples:

N_u = number of samples in up-sampled (full resolution) line
N_d = number of samples in a down-sampled (half resolution) line

Note: The term 'line' refers to all the samples in a horizontal row or vertical column in a Y, C_b or C_r component plane. Down-sampling operations are identical for both rows and columns, so the following examples are illustrated using one dimensional line of samples. In cases where both vertical and horizontal down-sampling is performed, the horizontal lines are down-sampled first, followed by the vertical lines.

For luma lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution luma line)
if ((N_d & 15) != 0)
    $N_d = N_d + 16 – (N_d \& 15)$

For color-difference lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution color-difference line)
if ((N_d & 7) != 0)
    $N_d = N_d + 8 – (N_d \& 7)$

Figure 161: Example of down-sampling one dimensional line for luma and color-difference.

B.2 Decoder up-sampling

At the decoder, up-sampling may be applied to the decoded frame if the value of the RESPIC syntax element as defined in Table 38 indicates that the display resolution is greater than the coded resolution. In this case, attention should be paid to the relative spatial positioning of the samples produced from the encoder down-sampling when implementing the up-sampling process. In particular, decoders may implement up-sampling according to the reverse of the algorithm described in Annex B.1.

B.3 Encoder down-sampling (informative)

On the encoder side, down-sampling is applied to the input frame if the current coding resolution is smaller than the original resolution. In the main profile, the RESPIC syntax element specifies the scaling of the current relative to the full resolution frame as defined in section 8.1.1.3 and Table 38.

The encoder can implement down-sampling according to the algorithm described in Annex B.1.

B.4 Anti-alias filtering (informative)

Filters can be added to reduce alias artifacts. These filter coefficients are implementation dependent and optional.

Annex C
Hypothetical reference decoder

Coded video bitstreams shall meet the constraints imposed by a hypothetical reference decoder (HRD) defined in this annex. The HRD is conceptually connected to the output of an encoder, and consists of a buffer, a decoder and a display unit, as illustrated in Figure 162.

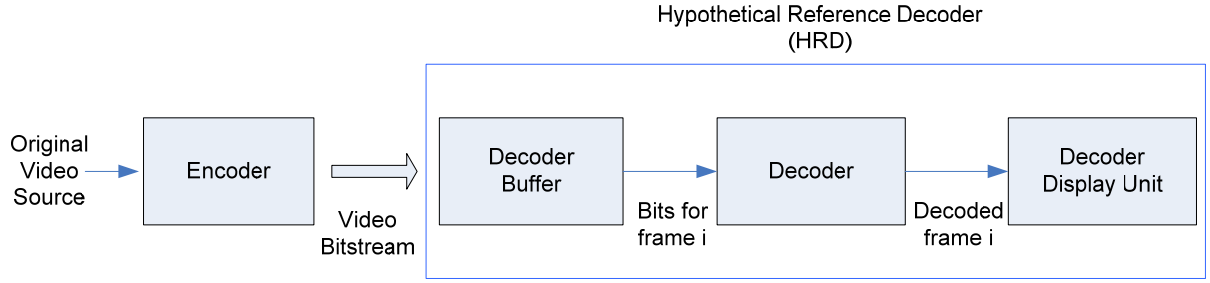


Figure 162: Components of an HRD: decoder buffer, decoder and display unit

The HRD does not mandate buffering, decoding, or display mechanisms for decoder implementations. Its purpose is to limit the encoder's bit rate fluctuations according to a basic buffering model, so that the resources necessary to decode the bitstream are predictable.

The HRD can operate in constant-delay mode or variable-delay mode. Constant-delay is appropriate for most applications, including broadcast, streaming, packaged media (e.g., DVD), etc. Variable-delay is appropriate for video conferencing.

All computations in this Annex are done with infinite precision real-values, so that no rounding errors propagate.

C.1 Leaky bucket model

C.1.1 Leaky bucket algorithm

The buffering model that governs the operation of the HRD is known as a leaky bucket. The algorithm is defined in this section. A leaky bucket shall be characterized by three parameters (R, B, F) where:

- R is the peak transmission bit rate (in bits per second) at which bits enter the decoder buffer,
- B is the capacity (in bits) of the decoder buffer and
- F is the initial decoder buffer fullness (in bits)[2], which shall be smaller than or equal to B.

In the HRD, the video bitstream shall be received at bit rate smaller than or equal to the peak transmission rate R, and it shall be stored in a decoder buffer of size B until the buffer fullness reaches F bits. This time is referred to as the initial delay. The decoder then removes the bits for the first video frame of the sequence from the buffer, and decodes that frame. The bits for the following frames are also removed and decoded at subsequent time intervals. If a frame is coded as two interlaced fields, the bits for both fields shall be removed together as a pair and decoded. From the

[2] A leaky bucket may also be specified by parameters (R, B, F^e), where F^e is the initial encoder buffer fullness. Here, the initial decoder buffer fullness F has been chosen.

perspective of the leaky buffer model, the removal of bits from the buffer and the decoding operation are treated as instantaneous.

Figure 163 illustrates the decoder buffer fullness as a function of time for a bitstream that is contained in a leaky bucket of parameters (R, B, F). The decoder buffer fullness β_i after removing frame i, with $i > 1$, shall be as follows:

$$\beta_1 = F - b_1$$

$$\beta_i = min(B, \beta_{i-1} + R_i (t_i - t_{i-1})) - b_i \quad , \qquad \text{(C.1)}$$

where t_i is the decoding time for frame i, and b_i is the number of bits for frame i, and frame 1 is the first frame. The parameter R_i is the average bit rate (in bits per second) that enters the buffer during the time interval (t_i, t_{i-1}) and shall be such that $R_i <= R$ for all i. In Figure 163, the transmission rate happens to be constant and equal to the peak R, and hence $R_i = R$ for all i.

In the leaky bucket model defined for this HRD, the decoder buffer may fill up, but shall not overflow. The buffer fullness at any time instant shall be less than or equal to B. As a result, in equation (C.1), the *min(B, x)* operator implies that $\beta_i <= B$, for all i. An example of a decoder buffer that fills up in several periods of time is shown in Figure 164.

When the decoder buffer is full, the encoder shall not send any more bits until there is room in the buffer.

Note: This phenomenon occurs frequently in practice. For example, a Digital Video Disc includes a video coded bitstream of average rate 4-6 Mbps, while the disk drive speed or peak rate R is about 10 Mbits/sec. Since the bit rate used in most time intervals is less than 10 Mbits/sec, the decoder buffer is often full. More generally, if an encoder is producing fewer bits than those available in the channel, the decoder buffer will stop filling up.

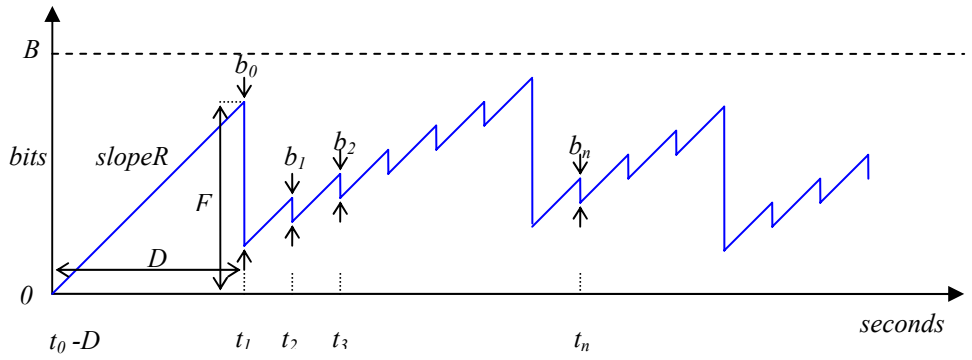


Figure 163: Decoder buffer fullness - contained

The plot illustrates an example of decoder buffer fullness when decoding a generic video bitstream that is contained in a leaky bucket of parameters (R, B, F). R is the peak incoming (or channel) bit rate in bits/sec, and in this case the transmission rate is constant and equal to the peak R throughout the video sequence. B is the buffer size in bits and F is the initial decoder buffer fullness in bits. $D = F/R$ is the initial or start-up (buffer) delay in seconds. The number of bits for the ith frame is b_i. The coded video frames are removed from the buffer (typically according to the video frame rate), as shown by the drops in buffer fullness, and are decoded instantaneously.

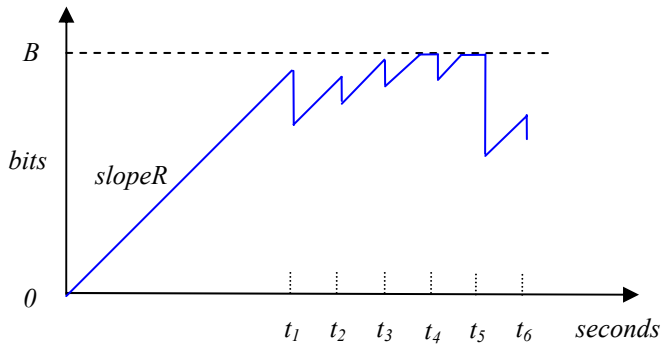


Figure 164: Decoder buffer fullness – maximum

Plot of decoder buffer fullness, where the fullness reaches the maximum buffer size *B* during some time segments. In this example, such segments are a subset of the intervals (t_2, t_3) and (t_3, t_4). When the decoder buffer is full, the encoder does not send any bits.

Decoder buffer underflow occurs usually if an encoder produces relatively large frames. The decoder buffer fullness can then be reduced to the point that the bits for the next frame are not available at the nominal decoding time.

A leaky bucket with parameters (*R*, *B*, *F*) is said to contain a coded video bitstream if there is no underflow of the decoder buffer (i.e., $\beta_i >= 0$, for all *i*). A leaky bucket with parameters (*R, B, F*) shall contain a coded video bitstream if the following constraints hold:

$$\beta_1 = F - b_1$$

$$\beta_i = min(B, \beta_{i-1} + R_i (t_i - t_{i-1})) - b_i, \quad i > 1$$

$$R_i <= R \qquad \text{all } i$$

$$\beta_i >= 0 \qquad \text{all } i$$

(C.2)

C.1.2 Constant delay mode constraints

The bitstream shall meet the restrictions imposed by equation (C.2), so that at least one leaky bucket (R, B, F) contains the bitstream. The leaky bucket values (R, B, F) shall be signaled to the decoder so that the rate and buffer size resources necessary to decode this bitstream are predictable.

C.1.3 CBR and VBR bitstreams

A bitstream that meets the constraints of the equations in (C.2) is denoted a variable bit rate or VBR bitstream, e.g., see ISO/IEC 13818-2.

If the constraints in equation (C.2) apply to a bitstream without the *min(B, x)* operator (i.e., $\beta_i = \beta_{i-1} + R_i (t_i - t_{i-1}) - b_i$, for all *i*), and if $R_i = R$ for all *i*, and if there is no buffer overflow (i.e., $\beta_i + b_i <= B$, for all *i*), the bitstream is denoted a constant bit rate or CBR bitstream.

Since CBR bitstreams are a special case of VBR bitstreams, they are subject to the same constraints.

C.2 Multiple leaky buckets

A bitstream may be contained in multiple leaky buckets.

For example, if a video stream is contained in a leaky bucket with parameters (*R*, *B*, *F*), it can also be contained in a leaky bucket with a larger buffer size (*R*, *B'*, *F*), $B' > B$, or in a leaky bucket with a higher peak transmission bit rate (*R'*, *B*, *F*), $R' > R$, or in a leaky bucket with larger initial buffer fullness (*R*, *B*, *F'*), $F' > F$, $F \leq B$. Moreover, it can also be contained in a leaky bucket with a lower peak transmission bit rate (*R'*, *B*, *F*), $R' < R$, if the video is time-limited. In the worst case, as *R'* approaches 0, the buffer size and initial buffer fullness can be as large as the bitstream itself. In short, a video bitstream can be transmitted at any peak transmission bit rate (regardless of the

average bit rate of the sequence) without suffering decoder buffer underflow, as long as the buffer size and initial delay are large enough.

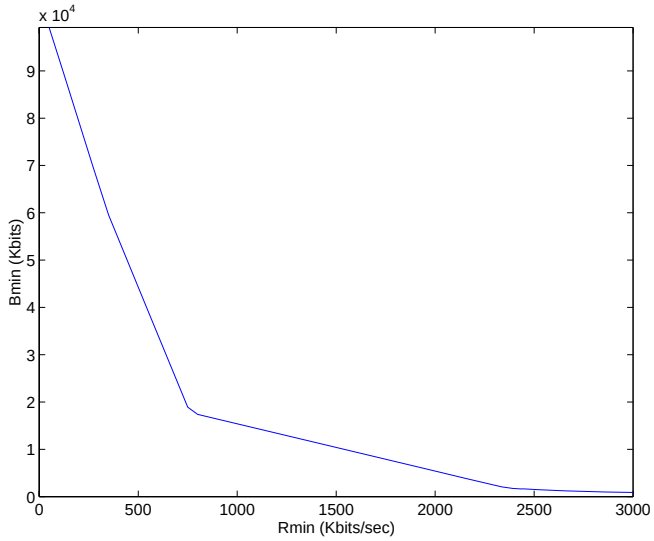


Figure 165: Illustration of peak bit rate R_{min} and buffer size B_{min} values for a given video bitstream. This curve indicates that in order to transmit the stream at a peak bit rate R, the decoder needs to buffer at least B_{min}(R) bits. Observe that higher peak rates require smaller buffer sizes. Alternatively, if the size of the decoder buffer is B, the minimum peak rate required for transmitting the bitstream is the associated $R_{min}(B)$.

Further, for any value of the peak transmission bit rate R, and assuming $R_i = R$ for all i in equation (C.2), there exists the minimum buffer size B_{min} and the minimum initial buffer fullness F_{min} that shall contain the video bitstream. These minimum values may be computed using a simple search using the constraints in (C.2), as demonstrated in Ribas-Corbera et al. 2003. By computing B_{min} for each R, one can plot a curve of optimum R-B values such as the one in Figure 165.

C.3 Bitstream syntax for the hypothetical reference decoder

C.3.1 Constant-delay mode, Advanced profile constraints.

The bitstream shall signal N leaky bucket models, each of which shall contain the video bitstream, as defined in (C.2). The desired value of N shall be selected by the encoder (where N > 0).

The parameter values of these leaky buckets shall be expressed as follows:

$$(R_1, B_1, F_1), (R_2, B_2, F_2), \ldots, (R_N, B_N, F_N), \tag{C.3}$$

The HRD syntax element values shall be communicated to the decoder by the Transport Layer for video bitstreams conformant to the Simple and Main profiles.

The HRD syntax element values shall be in the sequence header for Advanced profile bitstreams, except for variable-delay mode. The sequence header for the Advanced profile is defined in Section 6.1, and the HRD syntax element is defined in section 6.1.15.1.

Note: The number of bits used in prior frames does not affect the equations in (C.2) to determine the leaky bucket constraints for the remaining frames of the video bitstream, and hence the leaky bucket values may be modified throughout the video bitstream. Also, an encoder may want to use fewer leaky buckets later in the bitstream to avoid syntax overhead.

The following HRD syntax elements that need to be signaled are defined in Table 13 and Table 15.

Note: the syntax is an example encoding and not a syntax element.

hrd_num_leaky_buckets – A number between 0 and 31 that shall specify the number of leaky buckets *N*. See 6.1.15.1 for definition.

hrd_rate[n**]** and **bit_rate_exponent** – These syntax elements shall define the peak transmission rate R_n in bits per second for the *n*th leaky bucket. See 6.1.15.1 for definition.

hrd_buffer[n**]** and **buffer_size_exponent** – These syntax elements define the buffer size B_n in bits for the nth leaky bucket. See 6.1.15.1 for definition.

hrd_full[n**]** – This syntax element shall define the decoder buffer fullness as an upwards rounded fraction of the buffer size B_n, in units of $B_n/256$. See 6.2.12 for definition. Its value shall be computed as follows:

$$\textbf{hrd\_full[n]} = \left\lceil 256 \times \frac{\min(B_n, \beta_{i,n} + b_i)}{B_n} \right\rceil - 1 \qquad \text{(C.4)}$$

where $\min(B_n, \beta_{i,n} + b_i)$ is the decoder buffer fullness in bits *before* removing the current *i*th frame.

In equation (C.2), the decoder buffer fullness *after* removing the *i*th frame equals β_i. (C.4) uses a similar notation for the equivalent value $\beta_{i,n}$, but the subscript *n* denotes the *n*th leaky bucket.

The $\lceil x \rceil$ operator rounds up the value of *x* to the nearest higher integer. For example, $\lceil 14.3 \rceil = 15$.

Observe that in the first frame of the video stream (i.e., *i*=1), the initial buffer fullness $F_n = (\beta_{1,n} + b_1)$.

If the number of leaky buckets N =0, the leaky bucket used shall be (R_1, B_1, F_1), where R_1 and B_1 correspond to the values Rmax and Bmax for the given profile and level of the bitstream, as defined in Annex D and the initial buffer fullness shall equal the buffer size, i.e., $F_1=B_1$

C.3.2 Encoder considerations (informative)

In practice, an encoder can do the following:

(a) Pre-select the leaky bucket values in (C.3) and encode the bitstream with a rate control that makes sure that all of the leaky bucket constraints are met.
(b) Encode the bitstream and then use the equations in (C.2) to compute a set of leaky buckets containing the bitstream at *N* different values of *R*.
(c) Do both (a) and (b), i.e., pre-select the leaky buckets and later compute more after the bitstream is encoded.

Approach (a) could be applied to live or on-demand transmission applications, while (b) and (c) only apply to on-demand.. If *N*=1, the hypothetical reference decoder is a subset of the Video Buffering Verifier of ISO/IEC 13818-2.

C.4 Interpolating leaky buckets (informative)

The curve of (R_{min}, B_{min}) pairs, or that of (R_{min}, F_{min}) for any bitstream (such as the one in Figure 165) is piecewise linear and convex. This is discussed in Ribas-Corbera et al. 2003. Because of the convexity, if *N* points of the curve are provided, the decoder can linearly interpolate the values to arrive at some points (R_{interp}, $B_{interp,}$ F_{interp}) that are slightly but safely larger than (R_{min}, B_{min}, F_{min}).

As mentioned earlier, the leaky buckets in (C.3) are ordered from smallest to largest bit rate, i.e., $R_n < R_{n+1}$. Let us assume that the encoder computes these leaky bucket models correctly and hence $B_n > B_{n+1}$. Figure 166 illustrates a set of *N* leaky bucket models and their interpolated or extrapolated (*R*, *B*) values.

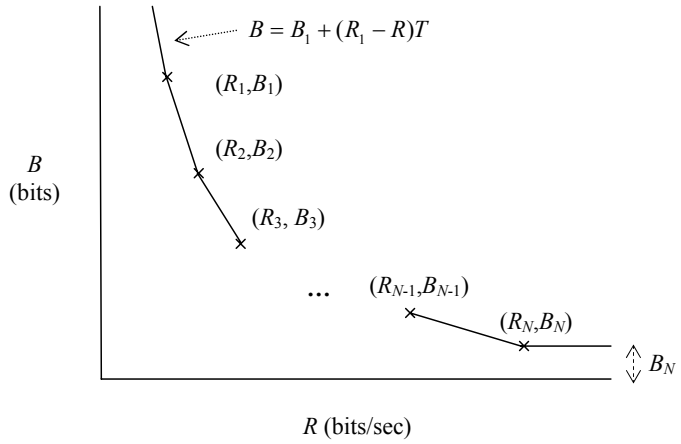


Figure 166: Example of (*R, B*) values available for the generalized hypothetical reference decoder (GHRD), all of which are guaranteed to contain the bitstream. *T* is the time length or duration of the encoded video sequence.

The interpolated buffer size B between points n and n+1 follow the straight line:

$$B = \frac{R_{n+1} - R}{R_{n+1} - R_n} B_n + \frac{R - R_n}{R_{n+1} - R_n} B_{n+1}, \qquad R_n < R < R_{n+1}. \tag{C.5}$$

Likewise, the initial decoder buffer fullness F can be linearly interpolated:

$$F = \frac{R_{n+1} - R}{R_{n+1} - R_n} F_n + \frac{R - R_n}{R_{n+1} - R_n} F_{n+1}, \qquad R_n < R < R_{n+1}. \tag{C.6}$$

The resulting leaky bucket with parameters (R, B, F) is guaranteed to contain the bitstream, because the minimum buffer size B_{min} is convex in both R and F, that is, the minimum buffer size B_{min} corresponding to any convex combination $(R,F) = a(R_k,F_k) + (1\text{-}a)(R_{k+1},F_{k+1})$, $0 < a < 1$, is less than or equal to $B = aB_k + (1\text{-}a)B_{k+1}$.

As discussed earlier, if R is larger than R_N, the leaky bucket (R, B_N, F_N) will also contain the bitstream, and hence B_N and F_N are the buffer size and initial decoder buffer fullness recommended when $R \geq R_N$. If R is smaller than R_1, then the upper bound $B = B_1 + (R_1\text{-}R)T$ may be used (and one may set $F = B$), where T is the time length of the video sequence in seconds. These (R, B) values outside the range of the N points are also shown in Figure 166.

Using equations (C.5) and (C.6), when the transmission peak rate of a given encoding/decoding system is known, the decoder can determine a nearly minimum leaky bucket buffer size and delay. Alternatively, knowing the physical buffer size, a smart decoder can ask a transmitter to use the smallest peak rate that will enable decoding in such buffer size. In short, the leaky bucket model values in equation (C.3) can be linearly interpolated or extrapolated to be able to determine nearly optimum leaky buckets.

C.5 Display issues (informative)

The leaky bucket model does not address the case when a video frame is displayed in the HRD display unit. A decoder, including this HRD, will normally display frames in the proper order. For example, if a frame is composed of two fields, it is assumed that the field that comes first in time will be displayed first. P frames and B frames could also be re-ordered properly before display. Nevertheless, constraints on display times (e.g., according to the decoding times t_1, t_2, etc.) are beyond the scope of this standard.

C.6 Time-conformant decoders

Time-conformant decoders ensure a fixed end-to-end delay, and hence they are of interest for most video coding applications (e.g., video broadcast, video on demand, DVD playback, etc.), while non time-conformant decoders are

common mainly for video conferencing. A practical time-conformant decoder decodes bitstreams without buffer underflow.

Given a fixed transmission rate and decoder buffer size, a time-conformant decoder implementation shall buffer enough data initially to prevent buffer underflow during the decoding process. Such a decoder shall therefore operate according to either one of the N leaky buckets, or one of the interpolated leaky buckets defined in equations (C.5) and (C.6).

Given a channel rate R, a time-conformant decoder implementation shall use equations (C.5) and (C.6) to find a minimum value of B and F, the physical buffer size in the decoder shall be larger than or equal to B, and shall buffer at least F bits before starting the decoding process.

Given a physical buffer size B, a time-conformant decoder implementation shall use equations (C.5) and (C.6) to find a minimum value of R and F, shall ensure that the channel rate is larger than or equal to R, and shall buffer at least F bits before starting the decoding process.

C.7 Variable-delay mode

This section refers to the variable-delay mode of operation in the HRD, which is useful for video conferencing applications.

This mode of operation in the HRD is signaled when HRD parameters are not signaled in the sequence header, i.e. when HRD_PARAM_FLAG is equal to value 0. In this mode:

- The leaky bucket shall be (R_1, B_1, F_1), where R_1 and B_1 correspond to the values Rmax and Bmax for the given profile and level of the bitstream, as defined in Annex D.
- The initial buffer fullness shall equal the buffer size, i.e., F_1=B_1. In practice, F_1 may equal the number of bits for the first frame, i.e., F_1= b_1.
- The decoder buffer in the HRD shall be examined every T seconds, where T is the inverse of the maximum frame rate in the example column of Table 253, for the respective profile and level of the given bitstream. If at least one complete coded picture is in the buffer, then all the data for the earliest picture in the bitstream order shall be instantaneously removed. Immediately before removing the picture, the buffer fullness shall be less than B_1.

In this mode, the HRD shall wait until a complete video frame has arrived at the buffer before decoding the frame. As a result, the delay is minimized for a given frame, but the end-to-end delay is not constant. This mode enables the encoder to send big pictures to the decoder while preventing buffer overflow.

The variable-delay mode of operation is similar to the low-delay mode in ISO/IEC 13818-2, or the default HRD operating mode in ITU-T H.263.

C.8 Benefits of multiple leaky buckets (informative)

In the constant-delay mode, prior hypothetical reference decoders operate with a fixed peak bit rate, buffer size, and initial delay. However, in many of today's video applications (e.g., video streaming through the Internet) the peak transmission bit rate varies according to the network path (e.g., how the user connects to the network: by dial-up modem, ISDN, xDSL, cable modem, etc.) and also fluctuates in time according to network conditions (e.g., congestion, the number of users connected, etc.) In addition, the bitstreams are delivered to a variety of devices with different buffer capabilities (e.g., hand-sets, PDAs, PCs, set-top-boxes, DVD-like players, etc.) and are created for scenarios with different delay requirements (e.g., low-delay streaming, progressive download or pseudo-streaming, etc.) The multiple leaky bucket approach used in the HRD of this standard is more flexible than prior HRDs and enables a system to decode a bitstream at different peak transmission bit rates, and with different buffer sizes and start-up delays.

To be more concrete, given a desired peak transmission bit rate, a time-conformant decoder selects the smallest buffer size and delay (according to the available leaky bucket data) such that it will be able to decode the bitstream without suffering from buffer underflow. Conversely, for a given buffer size, the hypothetical decoder selects and operates at the minimum required peak transmission bit rate.

There are multiple benefits of this generalized hypothetical reference decoder. For example, a content provider can create a bitstream once, and a server can deliver it to multiple devices of different capabilities, using a variety of channels having different peak transmission bit rates. Or a server and a terminal can negotiate the best leaky bucket for

the given networking conditions, e.g., the one that produces the lowest start-up (buffer) delay, or the one that requires the lowest peak transmission bit rate for the given buffer size of the device.

C.9 Default bit rates

The following default maximum bit rates shall be used in the HRD model in the absence of explicit signaling:

Table 250: Maximum Bit Rate as a function of Profiles and Levels

Profile	Level	Rmax[profile,level]	VBVmax[profile,level]
Simple	Low	96,000 bits/sec	327,680 bits = 40 x 1024 bytes
	Medium	384,000 bits/sec	1261568 bits = 154 x 1024 bytes
Main	Low	2,000,000 bits/sec	5013504 bits = 612 x 1024 bytes
	Medium	10,000,000 bits/sec	10010624 bits = 1222 x 1024 bytes
	High	20,000,000 bits/sec	40009728 bits = 4884 x 1024 bytes
Advanced	L0	2,000,000 bits/sec	4096000 bits = 500 x 1024 bytes
	L1	10,000,000 bits/sec	20480000 bits = 2,500 x 1024 bytes
	L2	20,000,000 bits/sec	40960000 bits = 5,000 x 1024 bytes
	L3	45,000,000 bits/sec	90112000 bits = 11,000 x 1024 bytes
	L4	135,000,000 bits/sec	270336000 bits = 33,000 x 1024 bytes

Annex D
Profile and Levels

A profile is a defined subset of the syntax of the standard, with a specific set of coding tools, algorithms, and syntax associated with it. A level is a defined set of constraints on the values which can be taken by the parameters (such as bit rate and buffer size) within a particular profile. Within the same profile, higher levels generally imply higher requirements in processing speed and memory.

Encoders produce bitstreams conformant to a given profile and level, and decoders decode bitstreams conformant to a set of given profiles and levels. Note that a bitstream conformant to a particular profile/level combination is also conformant with all higher levels at the same profile (i.e. with the exception of the highest level, each level is a sub-set of all higher levels within a given profile). Therefore, profiles and levels are critical to ensure interoperability between encoders, coded bitstreams, and decoders.

D.1 Overview (Informative)

There are three profiles in this recommendation, Simple, Main and Advanced:

- The Simple profile targets low-rate internet streaming and low-complexity applications such as mobile communications, or playback of media in personal digital assistants. There are two levels in this profile.
- The Main profile targets high-rate internet applications such as streaming, movie delivery via IP, or TV/VOD over IP. This profile contains three levels.
- The Advanced profile targets broadcast applications, such as digital TV, DVD, or HDTV. It is the only profile that supports interlaced content. In addition, this profile contains the required syntax elements to transmit video bitstreams conformant to this standard into generic systems, such as MPEG-2 Transport or Program Streams (ISO/IEC 13818-2). This profile contains five levels.

Table 251 lists all the profiles and levels, and the label associated to each of them.

Note: For definition of profile, see section 6.1.1 and Annex J.1.1. For definition of level, see section 6.1.2 and Annex J.1.2.

Profile	Level	Label
Simple	Low	SP@LL
	Medium	SP@ML
Main	Low	MP@LL
	Medium	MP@ML
	High	MP@HL
Advanced	L0	AP@L0
	L1	AP@L1
	L2	AP@L2
	L3	AP@L3
	L4	AP@L4

Table 251: List of profiles and levels defined in this standard.

D.2 Profiles (Informative)

Table 252 indicates the constraints on the algorithms or compression features for each of the profiles. If a compression feature is listed in the table, it is only supported by the profiles marked with "X". Other compression features are used in all profiles. Note that dynamic resolution change refers to scaling the coded picture size by a factor of 2 (via RESPIC syntax element) in the main profile, and to scaling the coded picture size by arbitrary scaling factors (via transmitted coded size syntax element in the entry-point header) in the advanced profile. Note that range adjustment refers to range reduction (by a factor of 2 via the RANGEREDFRM (7.1.1.3) syntax element) in the main profile, and range mapping (via RANGE_MAPY (6.2.15.1) and RANGE_MAPUV (6.2.16.1) syntax elements) in the advanced profile.

Compression Feature	Simple profile	Main Profile	Advanced Profile
Baseline intra frame compression	X	X	X
Variable-sized transform	X	X	X
Transform Specification	X	X	X
Overlapped transform	X	X	X
4 motion vectors per macroblock	X	X	X
Quarter-pixel motion compensation Y	X	X	X
Quarter-pixel motion compensation Cb, Cr		X	X
Start codes		X	X
Extended motion vectors		X	X
Loop filter		X	X
Dynamic resolution change		X	X
Adaptive macroblock quantization		X	X
Bidirectional (B) frames		X	X
Intensity compensation		X	X
Range adjustment		X	X
Interlace: Field and frame coding modes			X
Sequence level metadata			X
Entry point layer			X
Display metadata (pan/scan, colorimetry, sample aspect ratio, pull-down, top field first, repeat first field, etc.)			X

Table 252: Codec options in the Simple, Main and Advanced profile.

D.3 Levels

There are several levels for each of the profiles. Each level limits the video resolution, frame rate, HRD bit rate, HRD buffer requirements, and the motion vector range. These limitations are defined in Table 253.

As explained in Annex C, the encoder can define multiple leaky buckets that contain a given video bitstream. The HRD is able to decode a bitstream operating according to any of those leaky bucket parameters, or even according to interpolations or extrapolations of such parameters.

For a bitstream to be conformant to a given profile and level, at least one of the leaky bucket parameters needs to be within the limits defined by the profile and level. For progressive, the picture rate is described

by the number of frames per second. For interlace, the picture rate is described by the number of frames per second. i.e., 15Hz in interlace refers to 15 frames/second (which is equivalent to 30 fields/second).

Profile @Level	MB/s	MB/f	Examples	B	I	Rmax	Bmax	MV [H] x [V]
SP@LL	1,485	99	QCIF, 176x144, 15 Hz			96	20	**[-64, 63¾] x [-32, 31¾]**
SP@ML	7,200	396	CIF 352x288, 15 Hz QVGA, 320x240, 24 Hz			384	77	**[-64, 63¾] x [-32, 31¾]**
MP@LL	11,880	396	QVGA, 320x240, 24 Hz CIF 352x288, 30 Hz	x		2,000	306	**[-128, 127¾] x [-64, 63¾]**
MP@ML	40,500	1,620	480p, 720x480, 30 Hz 576p, 720x576, 25 Hz	x		10,000	611	**[-512, 511¾] x [-128, 127¾]**
MP@HL	245,760	8,192	1080p, 1920x1080, 25 Hz 1080p, 1920x1080, 30 Hz	x		20,000	2,442	**[-1024, 1023¾] x [-256, 255¾]**
AP@L0	11,880	396	CIF, 352x288, 25 Hz, CIF, 352x288, 30 Hz SIF, 352x240, 30 Hz	x		2,000	250	**[-128, 127¾] x [-64, 63¾]**
AP@L1	48,600	1,620	480i-SD, 704x480, 30 Hz 576i-SD, 720x576, 25 Hz	x	x	10,000	1,250	**[-512, 511¾] x [-128, 127¾]**
AP@L2	110,400	3,680	480p, 704x480, 60 Hz 720p, 1280x720, 25 Hz 720p, 1280x720, 30 Hz	x	x	20,000	2,500	**[-512, 511¾] x [-128, 127¾]**
AP@L3	245,760	8,192	1080i, 1920x1080, 25 Hz 1080i, 1920x1080, 30 Hz 1080p, 1920x1080, 25 Hz	x	x	45,000	5,500	**[-1024, 1023¾] x [-256, 255¾]**

			1080p, 1920x1080, 30 Hz 720p, 1280x720, 50 Hz 720p, 1280x720, 60 Hz 2048x1024, 30 Hz					
AP@L4	**491,520**	**16,384**	1080p, 1920x1080, 50 Hz 1080p, 1920x1080, **60 Hz** **2048x1536, 24 Hz** **2048x2048, 30 Hz**	**x**	**x**	**135,000**	**16,500**	**[-1024, 1023¾] x [-256, 255¾]**

Table 253: Limitations of profiles and levels. Column marked 'B' denotes B frames and loop filter support, and 'I' denotes interlace support. For interlace, picture rate is described in frames/second. (Fields/second is twice that value).

MB/s Maximum number of macroblocks per second

MB/f Maximum number of macroblocks within a frame

Example Example of maximum video resolution and frame rate. Other combinations that meet the profile and level requirements are also possible. For instance, in AP@L2, both “480p, 50Hz” and “480p, 60Hz” are supported.

Rmax HRD’s maximum peak transmission bit rate in units of 1,000 bits/sec.

Bmax HRD’s maximum buffer size in units of 16,384 bits

MV [H]x[V] Motion vector range in full pixel units. [H] = horizontal, [V] = vertical.

D.4 Syntax (Informative)

The Simple and Main profiles are communicated to the decoder by the syntax element PROFILE as part of the initialization metadata as described in Annex J.1.1. The Advanced profile is signaled to the decoder in the bitstream, by the syntax element **PROFILE**, which is included in the sequence header as described in Section 6.1.1.

The levels for Simple and Main profile are to be communicated to the decoder by the Transport Layer. The levels for the Advanced profile are defined in the syntax element **LEVEL**, which is included in the sequence header, as described in Section 6.1.2. The following codes are used to signal the levels in this profile:

000 AP@L0

001 AP@L1

010 AP@L2

011 AP@L3

100 AP@L4

101 SMPTE Reserved

110 SMPTE Reserved

111 SMPTE Reserved

Annex E
Start Codes and Emulation Prevention

The beginning of a Bitstream Data Unit (BDU) of compressed video data is signaled by an identifier called Start Code (SC). A BDU could be, for example, a sequence header, an entry-point header, or a slice (see Table 256 for a complete list).

This specification defines a sequence of four bytes as the start code which consists of a unique three-byte Start Code Prefix (SCP), and a one-byte Start Code Suffix (SCS). The SCP shall be the unique sequence of three bytes (0x000001). The SCS is used to identify the type of BDU that follows the start code. For example, the suffix of the start code before a picture is different from the suffix of the start code before a slice. Start codes shall always be byte-aligned.

An Encapsulation Mechanism (EM) is described to prevent emulation of the start code prefix in the bitstream. The compressed data before encapsulation by an encoder, or after extraction from the encapsulated form by the decoder, is called a Raw Bitstream Decodable Unit (RBDU), while Encapsulated BDU (EBDU) refers to the data when encapsulated.

Section E.1 specifies detection of start codes and EBDUs at the decoder, and is normative. Section E.2 deals with extraction of an RBDU from an EBDU, and is also normative. Section E.3 defines RBDU, EBDU and encapsulation. It also provides an encoder-side perspective on how start code and encapsulation operate. Section E.4 specifies constraints on byte stream data patterns. Section E.5 specifies start code suffixes for various BDU types, and is also normative. The constraints on bitstream construction using start codes are defined in Annex G.

E.1 Detection of Start codes and EBDU

The detection of an EBDU shall start with the search for the start code prefix.

E.1.1 Detection of Start Codes Starting from Byte-Aligned Positions

In a decoder that cannot lose byte-alignment, or once byte alignment has been established, start code detection shall be conducted by scanning the byte stream to detect the location of two or more consecutive byte-aligned bytes of value 0x00 followed by a byte of value 0x01. A start code prefix detection is declared for each such detected three-byte string.

> Note: In many application environments, the bitstream data is carried in an inherently byte aligned manner and thus byte alignment loss and recovery is not an issue. Since byte alignment is defined in relation to the location of the start of the bitstream, byte alignment is considered to always be unambiguously established for the decoding of correct and error-free conforming bitstreams. This is why the mechanism for recovery of lost byte alignment is only defined in an informative fashion (see section E.1.2).

When two successive start codes prefixes have been detected, the data bytes of the byte stream starting with the first byte of the first of the two start code prefixes and ending with the last byte prior to the first byte of the second of the two start code prefixes shall be considered to be an EBDU.

When the end of the bitstream is detected, the data bytes of the bitstream starting with the first byte of the last start code prefix in the bitstream and ending with the last byte in the bitstream shall be considered to be an EBDU.

E.1.2 Detection of Start Codes and Byte-Alignment Recovery After Loss of Byte Alignment in a Decoder (Informative)

In a decoder that has lost byte-alignment (as can happen in some transmission environments such as ITU-T H.320), start code prefix detection and byte-alignment detection are conducted as follows.

Starting at any alignment position, the decoder scans the byte stream data in a byte-wise fashion. Whenever a string of three or more consecutive bytes of value 0x00 is found, followed by any non-zero byte, a start code prefix detection is

declared and byte alignment is understood to have been recovered such that the first non-zero bit in the non-zero byte will be the last bit of a byte-aligned start code.

Note: The presence of extra zero-valued bytes prior to some start codes (or, equivalently, appended onto the end of some preceding EBDUs) as described in section E.1 is useful for ensuring that this process will result in byte alignment recovery.

E.2 Extraction of RBDU from EBDU

The decoder shall perform the extraction process of a raw BDU from an encapsulated BDU as defined below.

Step 1: The start-code suffix shall be used to identify the type of BDU. The bytes that follow the start code suffix, if any, shall then be further processed as follows.

Step 2: The decoder shall remove all zero-valued bytes at the end of EBDU. After this step, if the BDU payload after the start code suffix is not null, the last byte of the BDU will contain the '1' bit and any byte-aligning '0' bits that are present after the end of the RBDU.

Step 3: The bytes used for emulation prevention shall be detected and removed according to the following process:

Whenever a string of two consecutive bytes of value 0x00 are followed by a byte equal to 0x03, the byte equal to 0x03 is understood to be an emulation prevention byte and is discarded. This process is illustrated in Table 254.

Table 254: Decoder Removal of Emulation Prevention Data

Pattern to Replace	Replacement Pattern
0x00, 0x00, 0x03, 0x00	0x00, 0x00, 0x00
0x00, 0x00, 0x03, 0x01	0x00, 0x00, 0x01
0x00, 0x00, 0x03, 0x02	0x00, 0x00, 0x02
0x00, 0x00, 0x03, 0x03	0x00, 0x00, 0x03

Step 4: If there are bytes not removed in steps 2 or 3 that follow the start code suffix in the EBDU, in the last byte of the BDU data processed in step 3, the last non-zero bit is identified, and that non-zero bit plus all the zero bits that follow, shall be discarded. The result is the RBDU. If there are no bytes not removed in step 2 that follow the start code suffix in the EBDU, the RBDU shall be considered null.

E.3 Start Codes and Encapsulation – An encoder perspective (Informative)

The EM process for encapsulation of a RBDU to obtain an EBDU is described below.

Step 1: If the RBDU is not null, the EM appends a trailing '1' bit to the end of the RBDU and then stuffs between 0 and 7 bits onto the end of the BDU such that the BDU ends in a byte-aligned location. The value of these stuffing bits is '0'. As a result, at the end of this step, the BDU is represented in an integer number of bytes, in which the last byte of the BDU, if present, cannot be a zero-valued byte. The resulting string of bytes is called the payload bytes of the BDU.

Step 2: The encoder can begin an EBDU with any number of zero-valued bytes at the beginning of the EBDU.

Step 3: The start code is formed by starting with the three-byte start code prefix (0x000001), and appending the appropriate start code suffix that identifies the BDU type as specified in Table 256. If no additional zero-valued bytes were placed at the beginning of the EBDU, the start code is placed at the beginning of the EBDU. Otherwise, the start code is placed after the zero-valued bytes that were placed at the beginning of the EBDU.

Step 4: The remainder of the EBDU is formed by processing the payload bytes of the BDU through an emulation prevention process as follows, and appending the resulting string of bytes in the EBDU after the start code. The emulation of start code prefixes within the payload bytes of the BDU is eliminated via byte stuffing. The emulation prevention process is performed by starting at the beginning of the payload bytes of the BDU, and replacing each three-byte data string within the payload that consists of two consecutive bytes of value 0x00 followed by a byte that contains zero values in its six MSBs (regardless of the LSB values) with two bytes of value 0x00 followed by a byte equal to

0x03 followed by a byte equal to the last byte of the original three-byte data string. This process is illustrated in Table 255.

Table 255: Emulation Prevention Pattern Replacement

Pattern to Replace	Replacement Pattern
0x00, 0x00, 0x00	0x00, 0x00, 0x03, 0x00
0x00, 0x00, 0x01	0x00, 0x00, 0x03, 0x01
0x00, 0x00, 0x02	0x00, 0x00, 0x03, 0x02
0x00, 0x00, 0x03	0x00, 0x00, 0x03, 0x03

Step 5: The encoder can end an EBDU with any number of zero-valued bytes at the end of the EBDU.

Note 1: Except for the first EBDU and the last EBDU, the decoder cannot distinguish between pre-pended zero-valued bytes inserted in step 2 and appended zero-valued bytes inserted in step 5.

Note 2: Encoders that produce bitstreams used in application environments (such as ITU-T H.320) in which it is possible for byte alignment to be lost as a result of errors in bitstream transmission would add some zero-valued bytes in step 2 or step 5 at least occasionally, as these extra bytes assist in byte alignment recovery for decoders (see section E.1.2). For example, adding one extra zero-valued byte at the beginning of each sequence header, entry point header, and picture header is desirable in such application environments.

Note 3: The addition of zero-valued stuffing bytes can also be useful for splicing bitstreams, filling a constant bit-rate channel when sufficient picture quality has already been attained, etc.

Note 4: The zero-valued stuffing bytes inserted in step 2 or step 5 are not processed through the emulation prevention mechanism – only the bytes containing the RBDU or the byte-alignment stuffing bits appended to the end of the RBDU in step 1 need such processing.

E.4 Constraints on Byte Stream Data Patterns

The following byte patterns shall not be present at a byte-aligned position within the bitstream:

a) A string of two consecutive bytes of value 0x00 followed by a byte equal to 0x02.
b) A string of three or more consecutive bytes of value 0x00, if not followed by a byte of value 0x01.
c) A string of two consecutive bytes of value 0x00, followed by a byte of value 0x03, followed by a byte that has a value that is not one of 0x00, 0x01, or 0x02, or 0x03.

Note: The encapsulation process described in section E.1 can prevent these data patterns. The detection of these data patterns by a decoder should be considered an indication of an error condition. A loss of byte alignment should also be considered an error condition. For decoders operating in application environments in which decoder byte alignment can be lost, the detection of such error conditions should be used as an indication that byte alignment may have been lost.

E.5 Start Code Suffixes for BDU Types

The start code suffixes for various BDU types are presented in Table 256.

Table 256: Start Code Suffixes for Various BDU Types

Start Code Suffix	BDU Type
0x00	SMPTE Reserved
0x01-0x09	SMPTE Reserved
0x0A	End-of-Sequence

0x0B	Slice
0x0C	Field
0x0D	Frame
0x0E	Entry-point Header
0x0F	Sequence Header
0x10-0x1A	SMPTE Reserved
0x1B	Slice Level User Data
0x1C	Field Level User Data
0x1D	Frame Level User Data
0x1E	Entry-point Level User Data
0x1F	Sequence Level User Data
0x20-0x7F	SMPTE Reserved
0x80-0xFF	Forbidden

The Sequence Header BDU type is present in the bitstream to identify those BDUs which contain sequence header. See Section 6.1 for more details on sequence headers.

The Entry-point Header BDU type is present in the bitstream to identify those BDUs which contain entry-point header. See Section 6.2 for more details on entry-point header.

The Frame BDU type is present in the bitstream to identify those BDUs which contain the frame header and the frame data.

The Field BDU type is present in the bitstream to identify those BDUs which contain the second field of a picture that is coded as two separate fields.

The Slice BDU type is present in the bitstream to identify those BDUs which carry the slice data and the slice header. See Section 7.1.2 for more details on slices and slice header.

Sequence, Entry-point, Frame, Field, and Slice Level User data BDU types are used to transmit any user-defined data associated with the Sequence, Entry-point, Frame, Field, and Slice respectively. See Annex F for more details.

“End-of-sequence” is an optional BDU type which indicates that the current sequence has ended, and no further data will be transmitted for this sequence. Note that the transmission of an “end-of-sequence” may be present, but the end of a sequence shall be inferred from the header of the next sequence.

Start Code Suffix values 0x20 to 0x40 inclusive, are reserved for future use by SMPTE. Bitstreams shall not contain the start code suffix values 0x20 to 0x40, inclusive. Decoders shall ignore (remove from the bitstream and discard) these start codes and all data that follow them and precede the next start code in the bitstream, without effect on the decoding process.

NOTE: This specification for decoder treatment of the reserved start code suffix values 0x20 to 0x40 inclusive, allows the future specification of additional supplemental information which can be included within bitstreams without harm to compatibility with deployed decoders.

Annex F
User Data

User Data is conveyed as bitstream data units and may be included in the bitstream at the sequence, entry-point, frame, field, or slice layers. The User Data BDUs for each of these locations are identified by 5 unique start codes (0x1B to 0x1F inclusive) defined in Annex E.

The User Data syntax shall be defined as follows:

Table 257: User-data Syntax

User_data_parameters( )	**Number of bits**	**Descriptor**
{		
User_data_identifier	32	uimsbf
for(n=1; n <= end_of_bdu-1; n++)		
{		
User_data[n]	8	uimsbf
}		
Flushing_byte (0x80)	**8**	uimsbf

User_data_identifier is a fixed-length syntax element that identifies the type of user data. This syntax element shall be encoded using 32 bits and shall be set to a registered value of the format_identifier field as defined in ISO/IEC 13818-1 clauses 2.10, O and P. Note: contact the SMPTE Registration Authority for a list of existing values, and for registration of new values.

User_data is an array of 8-bit fixed length syntax elements that represent the user data. This data shall not emulate a start code in any 3 sequential bytes.

Flushing_byte shall be an 8-bit field set to the constant value '0x80'.

User Data shall be accounted for in the HRD buffer model.

User Data is a facility to allow for the carriage of data that can provide extra desirable user features. However, user data shall not be used for the carriage of proprietary codec performance enhancement information.

Annex G
Bitstream Construction Constraints – Advanced Profile

There are 11 distinct start code values defined in Annex E; one value for each of the following start codes: Sequence start code, entry start code, frame start code, field start code, slice start code, end-of-sequence start code and 5 values for user data start codes. Each start code is a 32-bit field. For user data, the value of the start code defines the scope of the user data.

Bitstreams shall be constructed according to the constraints below.

Conventions

The Figures below reference the bitstream constructs defined as follows:

SEQ_SC	Sequence Start Code
SEQ_HDR	Sequence Header
ENTRY_SC	Entry Point Start Code
ENTRY_HDR	Entry Point Header
FRM_SC	Frame Start Code
FRM_DAT	Frame Data (includes a Frame Header)
FLD_SC	Field Start Code
FLD1_DAT	Field 1 Data (includes a Frame Header)
FLD2_DAT	Field 2 Data (includes a Field Header)
SLC_HDR	Slide Header
SLC_DAT	Slice Data bytes (may include a Frame Header or a Field Header depending on location)
UD_SC	User Data Start Code
UD_DAT	User Data bytes

A conformant bitstream shall be one with any combination of the above start code and header pairs in any order that conforms to the syntax constraints found in this document. A conformant **picture-producing bitstream** shall be a conformant bitstream that shall be further constrained as described here.

- At least one pair of SEQ_SC and SEQ-HDR, and
- At least one pair of ENTRY_SC and ENTRY_HDR, and
- For each picture, one of:
    - A pair of FRM_SC and FRM_DAT, or
    - A 4-tuple of FRM_SC, FLD1_DAT, FLD_SC, and FLD2_DAT

The SLC_HDR/SLC_DAT, UD_SC/UD_DAT and End of Sequence Start Codes are all optional and need not be present.

A picture producing bitstream shall be further constrained by the rules defined in the remainder of this Annex.

G.1 Sequence start code

A sequence start code (value 0x0000010F) shall always be followed immediately by a sequence header. A sequence header shall always be followed by a user data start code or an entry point start code. The type of the first frame or first two fields following a sequence start code and a sequence header shall always be either I - if frame coding mode is set to Progressive or Frame Interlace - or I and P, or P and I, or I and I - if the frame coding mode is set to Field Interlace.

A sequence start code and a sequence header may be inserted at regular or irregular intervals throughout the bitstream.

Note: an encoder can adopt various policies to govern the insertion of sequence start codes and associated headers in a bitstream.

G.2 End-of-Sequence start code

An end-of-sequence start code (value 0x0000010A) may be inserted to indicate that the current sequence has ended and no further data will be transmitted for this sequence.

Note: The end of a sequence can be inferred from the sequence header of the next sequence. However, certain applications can benefit from the flexibility of explicitly indicating the end-of-sequence using this start code.

No data shall follow this start code. .

G.3 Entry point start code

An entry point start code (value 0x0000010E) shall always be followed immediately by an entry point header. In a bitstream, any entry point start code shall always be located after the last byte of a video frame and before the beginning of the next video frame. If there is a need to insert an entry point header or an entry point start code and an entry point header where there is already a sequence header between two consecutive video frames, the entry point header code or the entry point start code and the entry point header shall always follow the sequence header. An entry point header shall always be followed by a user data start code or a frame start code.

An entry point start code and an entry point header may be inserted at regular or irregular intervals in the bitstream.

Note: An encoder can adopt various policies to govern the insertion of entry point start codes and associated headers in a bitstream.

Insertion of any entry point start code and associated header shall always be made to signal a valid entry point in the bitstream, meaning that the video frames and video fields shall satisfy one of the conditions listed below (depending on the type of picture).

The purpose of the entry point start code is to signal the presence of special locations in a bitstream where there is no dependency on past decoded video fields or frames to decode the video frame following immediately the entry point start code and header. The conditions for achieving this are listed below. These conditions depend on the type of the first frames/fields past the entry point. The type of the first frame or first two fields following an entry point start code and an entry point header shall always be either I - if frame coding mode is set to Progressive or Frame Interlace - or I and P, or P and I, or I and I - if the frame coding mode is set to Field Interlace.

G.3.1 Case of I frame in Progressive mode

Figure 167 below illustrates how an entry point start code and an entry point header may be present before an I frame when the FCM syntax element (7.1.1.15) is set to '0' (Progressive mode).

Since the frame is intra-coded no additional condition is needed to make this I frame a valid entry point in a bitstream.

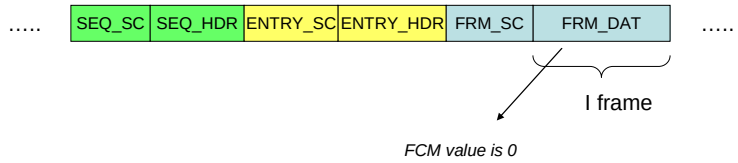


Figure 167: Entry Point Signaled before an I frame (Progressive Picture Coding)

G.3.2 Case of I/P frame in Field Interlace mode

Figure 168 below illustrates how an entry point start code and header may be present before an I/P frame when the FCM syntax element is set to 11b (Field Interlace mode).

Since the frame is composed of an I field followed by a P field, the following conditions shall be met to make this I/P frame a valid entry point in a bitstream:

- The value of the NUMREF syntax element (9.1.1.44) in the Field header of the P field of the entry I/P frame shall be '0'
- The value of the REFFIELD syntax element (9.1.1.45) in the Field header of the P field of the entry I/P frame shall be '0'

These conditions ensure that the P field is only predicted from the I field and therefore there is no dependency on frames or fields before the entry point.

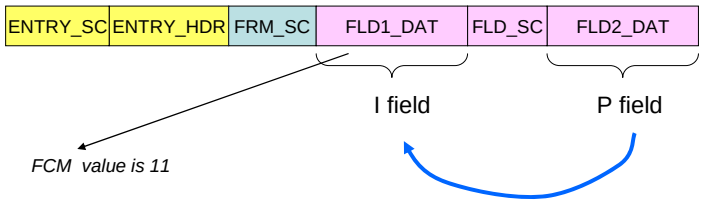


Figure 168: Entry Point Signaled before an I/P Frame (Field Interlace Picture Coding)

G.3.3 Case of P/I frame in Field Interlace mode

Figure 169 below illustrates how an entry point start code and header may be present before a P/I frame when the FCM syntax element is set to 11b (Field Interlace mode).

Since the frame is composed of a P field followed by an I field, the following conditions shall be met to make this P/I frame a valid entry point in a bitstream:

- Following the entry I field, a P/P frame (Field interlace mode) shall be present in the bitstream before any occurrence of P frames (progressive or frame interlaced modes).
- The value of the NUMREF syntax element in the Field header of the first P field following the entry P/I frame shall be '0'
- The value of the REFFIELD syntax element in the Field header of the first P field following the entry P/I frame shall be '0'
- Any B frames following the entry P/I frame in the bitstream and for which the presentation time comes later than the presentation times for that entry P/I frame shall not be encoded as depending on the P/I frame.
- The first (in temporal order) B field of any B/B frames following the entry P/I frame in the bitstream and for which the presentation time comes later than the presentation times of that P/I frame shall not be encoded as depending on the P field of the entry P/I frame.

These conditions ensure that the next P/P frame, B frame and B/B frames in the bitstream are only predicted from the entry I field and not the P field that immediately precedes it.

Note: It is impossible to have a valid entry point if there is a P frame that has been predicted from the P/I frame since this creates a dependency on the P field of the entry P/I frame.

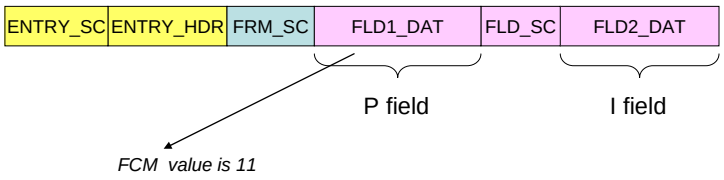


Figure 169: Entry Point Signaled before a P/I frame (Field Interlace Picture Coding)

G.3.4 Case of I/I frame in Field Interlace mode

Figure 170 below illustrates how an entry point start code and header may be present before an I/I frame when the FCM syntax element is set to 11b (Field Interlace mode). The Figure does not show a sequence start code and a sequence header before the entry point start code but such structures may precede the entry start code.

Since the frame is made of two I fields, no additional condition is needed to make this I/I frame a valid entry point in a bitstream.

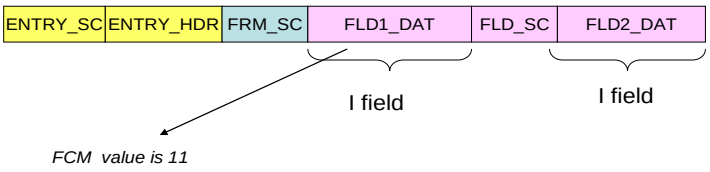


Figure 170: Entry Point Signaled before an I/I frame (Field Interlace Picture Coding)

G.3.5 Case of I frame in Frame Interlace mode

Figure 171 below illustrates how an entry point start code and header may be present before an I frame when the FCM syntax element is set to 10b (Frame Interlace mode).

Since the frame is intra-coded no additional condition is needed to make this I frame a valid entry point in a bitstream.

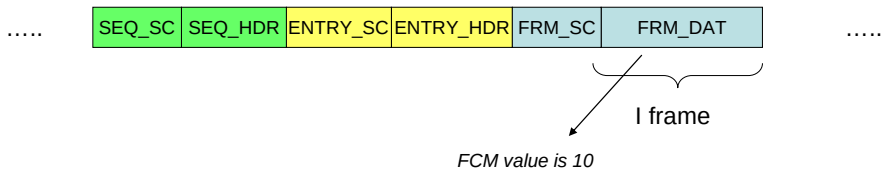


Figure 171: Entry Point Signaled before an I frame (Frame Interlace Coding)

G.4 Frame Start Code

A frame start code (value 0x0000010D) shall always be followed immediately by a frame header. In a bitstream, any frame start code shall always be located after the last byte of a video frame and before the beginning of the next frame. In the case of the Progressive or Frame Interlace mode, a frame start code shall signal the beginning of a new video frame. In the case of the Field Interlace mode, a frame start code shall signal the beginning of a sequence of two independently coded video fields.

G.5 Field Start Code

A field start code (value 0x0000010C) shall always be followed immediately by a field header. The field start code shall only be used for Field Interlaced frames and shall only be used to signal the beginning of the second field of the frame. A field start code shall always be present in Field Interlaced frames. The use of field start codes is forbidden in any frames encoded according to a Progressive or a Frame Interlace mode.

G.6 Slice Start Code

A slice start code (value 0x0000010B) shall always be followed immediately by a slice header. The slice start code shall be used to signal the beginning of a video slice.

G.7 User Data Start Codes

A user data start code (values 0x0000011B – 0x0000011F) shall always be followed by a user data header. The user data header is a 4-byte field identifying the contents of the user data that follows the header as defined in Annex E. The last user data byte of a user data structure (byte with value '0x80') shall always be followed by either a sequence start code or an entry point start code or a frame start code or a field start code or a slice start code or another user data start code (including any possible padding bytes between them), depending on the type of user data start code.

User data may be present at various locations in a bitstream. Although the value of any user data start code also specifies its scope (sequence-level, or entry point-level, or frame-level, or field-level, or slice-level user data), its location in a bitstream shall follow the rules described below. The user data structure at any level may be duplicated as many times as is needed meaning that a user start code and its user data bytes may be followed immediately by another user data start code and its user data bytes having the same scope.

G.7.1 Sequence-level user data

Figure 172 below shows sequence-level user data. When present, sequence-level user data shall be located in the bitstream after the sequence header and immediately before the start code signaling the beginning of the next bitstream data unit. Flushing bits and padding bytes may precede the first byte of the user data start code. Padding bytes may precede the first byte of the start code immediately following the last user data byte (that is, the flushing byte of value 0x80).

In Figure 172, the top bitstream illustrates the case where the next BDU is an entry point start code followed by an entry point header while the bottom bitstream illustrates the case where the next BDU is a frame start code followed by frame data (including a frame header).

Sequence-level user data shall be applicable to the entire sequence, that is until an end-of-sequence code or another sequence start code is encountered in the bitstream.

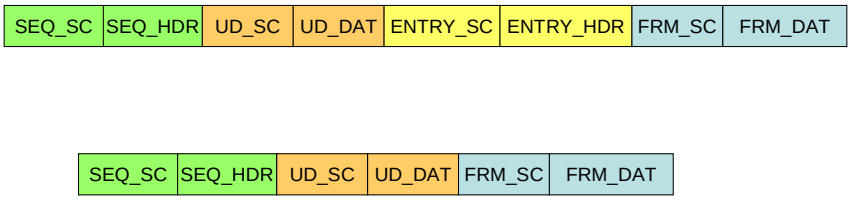


Figure 172: Sequence level User Data

G.7.2 Entry-Point level user data

Figure 173 below shows entry-point level user data. When present, entry- point level user data shall be located in the bitstream after the entry point header and immediately before the start code signaling the beginning of the start code for the next BDU – that is the start code signaling the next frame, the next entry point or the next sequence. Flushing bits and padding bytes may precede the first byte of the user data start code. Padding bytes may precede the first byte of the start code immediately following the last user data byte (that is, the flushing byte of value 0x80).

Entry-Point level user data shall be applicable to the sequence of video frames in the bitstream until another entry point start code or a sequence start code is encountered.

Figure 173: Entry-point level User Data

G.7.3 Frame-level User Data

Figure 174 below shows frame-level user data. Flushing bits and padding bytes may precede the first byte of the user data start code. Padding bytes may precede the first byte of the start code immediately following the last user data byte (that is, the flushing byte of value 0x80).

In the case of repeated fields (as the result of RFF being set to '1') or in the case of repeated frame (as the result of RPTFRM being set to a non-zero value), the number of user data BDUs shall always be a multiple of the number of displayed fields/frames. An equal number of user data BDUs shall then be assigned to each field/frame; the user data BDUs shall be placed in the bitstream in the order of the field/frame to which they are assigned. Some of the user data BDUs may be empty.

The top two bitstreams in Figure 174 consider the cases of progressive and frame interlace coded pictures. The top bitstream illustrates the case where slice start codes are not used. In this case, the frame-level user data, when used, shall appear at the end of the picture data and immediately before the start code for the next frame or the next entry point or the next sequence. The second bitstream illustrates the case where slice start codes are used. In this case, frame-level user data, when used, shall appear immediately before the start code signaling the second slice within the frame.

The bottom two bitstreams in Figure 174 consider the cases of field interlace coded pictures. The third bitstream illustrates the case where slice start codes are not used. In this case, the frame-level user data, when used, shall appear at the end of the first field data and immediately before the start code for the second field. The fourth bitstream illustrates the case where slice start codes are used. In this case, frame-level user data, when used, shall appear immediately before the start code signaling the second slice within the first field.

Frame-level user data shall be applicable to the frame until another frame start code, or an entry point start code, or a sequence start code is encountered in the bitstream.

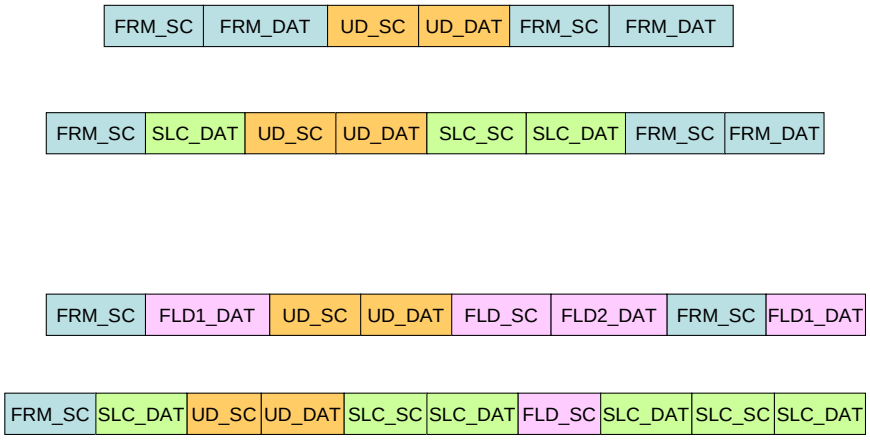


Figure 174: Frame-level User Data

G.7.4 Field-level user data

Figure 175 below shows field-level user data. Field-level user data shall only be allowed in frames that have been encoded as Field Interlace frames. Field-level user data shall not be used in frames encoded as progressive or frame interlace frames. Flushing bits and padding bytes may precede the first byte of the user data start code. Padding bytes may precede the first byte of the start code immediately following the last user data byte (that is, the flushing byte of value 0x80).

In Figure 175, the top two bitstreams show the case where slice start codes are not used. The top bitstream shows the situation where there is no frame-level user data. In this case, the field-level user data shall appear at the end of the field data and before the start code for the second field or the next frame or the next entry point or the next sequence. The second bitstream shows the situation where frame-level user data is also present. In this case, field-level user data for the first field, when present, shall appear before any frame-level user data. The values of the frame-level and field-level start codes are distinct so the scope of the user data is unambiguous.

The bitstream at the bottom of the Figure illustrates the case where slice start codes are used. The third bitstream shows the situation where there is no frame-level user data. In this case, field-level user data, when present, shall appear before the start code of the second slice within the field. The fourth bitstream shows the situation where frame-level user data is also present. In this case, field-level user data for the first field, when present, shall appear before any frame-level user data.

Field-level user data shall be applicable to the field until another field start code or a frame start code or an entry point start code or a sequence start code is encountered in the bitstream.

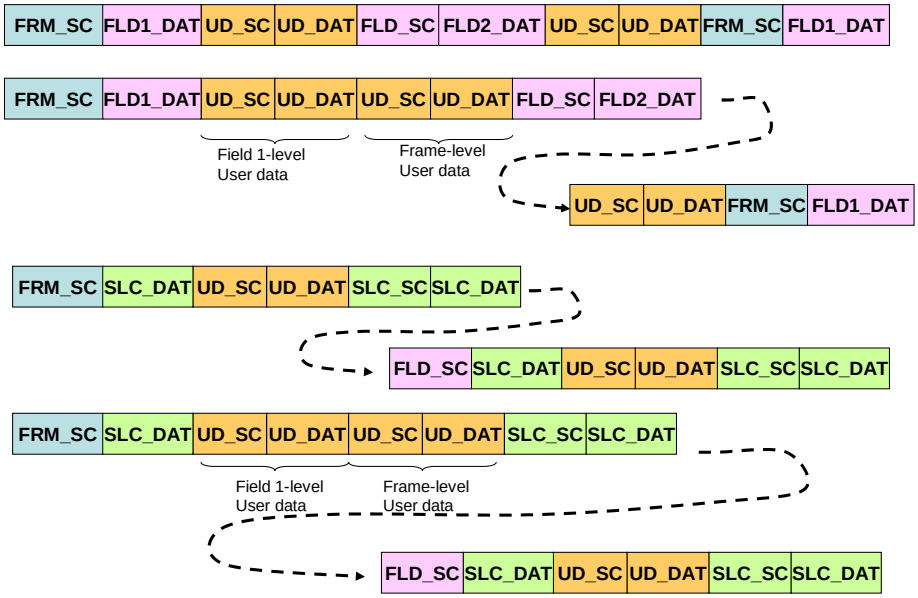


Figure 175: Field-level User Data

G.7.5 Slice-level user data

Figure 176 below shows slice-level user data. For the sake of simplicity, the figure assumes that the field is made of two distinct slices but it should not be implied that this is a bitstream constraint. Flushing bits and padding bytes may precede the first byte of the user data start code. Padding bytes may precede the first byte of the start code immediately following the last user data byte (that is, the flushing byte of value 0x80).

In Figure 176, the top two bitstreams illustrate the case of user data associated with the first slice in the picture – here a field, but the same concept shall apply for a frame. The top bitstream shows the situation where there is no field-level or frame-level user data. In this case, the slice-level user data, when present, shall appear at the end of the first slice data and before the start code for the second slice. The second bitstream shows the situation where both field-level and frame-level user data are also present. In this case, slice-level user data for the first slice, when present, shall appear before any field-level user data. The values of the frame-level, field-level and slice start codes are distinct so the scope of the user data is unambiguous.

The third bitstream at the bottom illustrates the case of slice-level user data associated with the second slice in the picture – here a field, but the same concept shall apply to a frame. In this case, the slice-level user data, when present, shall appear immediately before the start code for the next BDU – here a frame but it could be another slice-level user start code or a slice start code, a field start code or a frame start code.

Slice-level user data shall be applicable to the slice until another slice start code, a field start code, a frame start code, an entry point start code or a sequence start code is encountered in the bitstream.

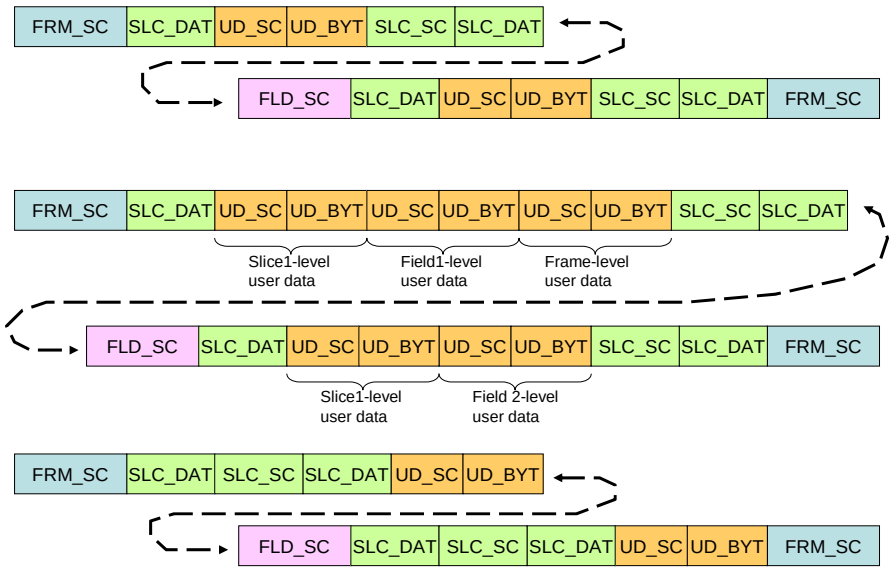


Figure 176: Slice-level User Data

G.8 Start code usage rules

Immediate (one after another of the same kind) duplication of sequence, entry point, frame, field or slice start code and header shall not be allowed. User data start codes and user bytes may be duplicated an arbitrary amount of times and at any level. Use of sequence, entry point, frame, field and slice start codes is optional.

Note: Many considerations may drive the use of start code. For example, entry start points may be used for facilitating receiver tuning or implementation of trick modes or splicing.

Note: The implementation of trick modes is facilitated by the following constraint. If a sequence start code or an entry point start code is present in the bitstream immediately before the header of a frame of type “P/I” (field interlace mode), the presence of a field start code between the last data byte of the first “P” field and the field header of the second “I” field facilitates implementation of trick modes.

Annex H
Post Processing for Coding Noise Reduction

Note: The application of post-processing (de-blocking and/or de-ringing) to images with relatively large frame sizes compressed at high bit rates can have significantly less perceptual benefit (or even cause perceptual degradation) as compared to similar post-processing applied to smaller or more highly compressed images. Similarly, the perceptual need for post-processing varies with image content (for example, artifacts may be more objectionable in a finely-detailed low-motion sequence than in an explosion). As such, it is highly beneficial to provide author-generated metadata regarding the level of post-processing to be applied to a segment.

Two post-processing filters are a deblocking filter and a de-ringing filter. The application of either one or both of them may be signaled in the bitstream with the POSTPROC field of the picture layer (see section 7.1.1.40). When so indicated, the post-processing filters described in this Annex may be applied by the decoder after decoding but before display (see Figure 3 and Figure 4).

In this Annex, the use of the mathematical symbol, ·, denotes multiplication and is interchangeable with the asterisk symbol.

H.1 Deblocking filter

The deblocking filter operations are performed across the 8x8 block edges as a post-processing operation of the decoder. Luma block edges as well as color-difference block edges are filtered. Figure 167 illustrates the block boundaries.

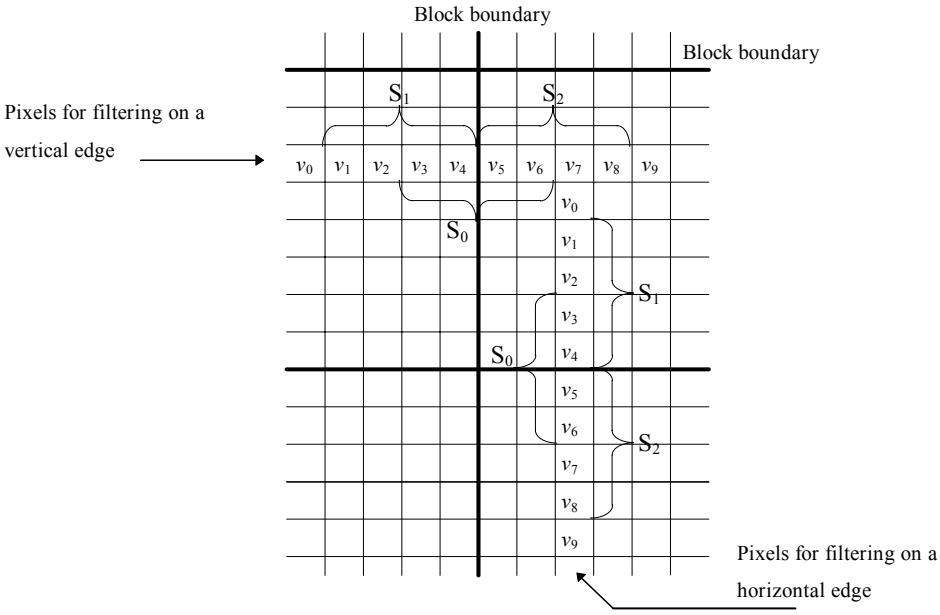


Figure 177: Boundary area around block of interest for deblocking

In the deblocking filter operations, one of two distinct modes is selected, depending on the pixel conditions across a block boundary (edge).

If the boundary is in a very smooth region with blocking artifacts due to small DC offset, the boundary is assigned to a DC offset mode.

Otherwise, the default mode operations are applied.

The procedure in Figure 178 is used to identify if the boundary is in a very smooth region.

```
// THR1 = 2
// THR2 = 6
eq_cnt = ϕ(v0–v1) + ϕ(v1–v2) + ϕ(v2–v3) + ϕ(v3–v4) + ϕ(v4–v5) + ϕ(v5–v6) + ϕ(v6–v7)
                + ϕ(v7–v8) + ϕ(v8–v9),
where          ϕ(γ ) = 1   if |γ | ≤ THR1 and 0 otherwise.
If (eq_cnt  ≥  THR2)
        DC offset mode is applied as boundary is in a very smooth region
else
        Default mode is applied.
```

Figure 178: Pseudo-code for determing Deblocking Filter mode

Default Mode:

In the default mode, a signal adaptive smoothing scheme is applied by differentiating image details at the block discontinuities using the frequency information of neighbor pixel arrays, S_0, S_1, and S_2 (see Figure 177). The filtering scheme in default mode replaces the boundary pixel values v_4 and v_5 with v_4' and v_5' as follows:

$v_4' = v_4 - d$,

$v_5' = v_5 + d$,

where $d = \text{clip}(5\cdot(a_{3,0}' - a_{3,0})//8,\ 0,\ (v_4 - v_5)/2) \cdot \delta(|a_{3,0}| < \text{QP})$

and $a_{3,0}' = \text{sign}(a_{3,0}) \cdot \text{sign}(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|)$.

and clip(x,p,q) clips x to a value between p and q,

and QP denotes the quantization parameter and is set to the value of PQUANT.

and $\delta(condition)$=1 if the "*condition*" is true and 0 otherwise.

where · (small dot) denotes multiplication.

Frequency components $a_{3,0}$, $a_{3,1}$, and $a_{3,2}$ can be evaluated from the simple inner product of the kernel [2 -5 5 -2] with the pixel vectors as follows:

```
a3,0 = ([2 -5 5 -2] • [v3 v4 v5 v6]T ) // 8,
a3,1 = ([2 -5 5 -2] • [v1 v2 v3 v4]T ) // 8,
a3,2 = ([2 -5 5 -2] • [v5 v6 v7 v8]T ) // 8

where • denotes matrix multiplication,


and T denotes matrix transpose,
```

Note: PQUANT is passed as a meta-data parameter to post-processing.

DC Offset Mode:

In the DC offset mode, a stronger smoothing filter is applied as in a very smooth region (identified by the procedure in Figure 178), the filtering in the default mode is not strong enough to reduce the blocking artifact. The filtering scheme in DC offset mode replaces the pixel values (v_1, v_2 ,v_3, v_4 v_5, v_6, v_7, v_8) with (v_1', v_2', v_3', v_4', v_5', v_6', v_7', v_8') as follows:

```
    MAX = max (v1, v2, v3, v4, v5, v6, v7, v8),
    MIN = min (v1, v2, v3, v4, v5, v6, v7, v8),
```

if (|MAX–MIN| < 2 * QP) {

$$v'_n = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8$$

$$p_m = \begin{cases} \left(\left|v_1 - v_0\right| < QP\right) ? v_0 : v_1, if \quad m < 1 \\ v_m, \qquad\qquad\qquad if\, 1 \le m \le 8 \\ \left(\left|v_8 - v_9\right| < QP\right) ? v_9 : v_8, if \quad m > 8 \end{cases}$$

$$\{b_k :- 4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\} // 16$$

}

else

No change will be done.

where x = (“condition”) ? a : b is defined as follows:

if (“condition”)

x = a

else

x = b

Figure 179: Pseudo-code for DC Offset Mode

The above filter operations are applied for all the 8x8 block boundaries by first applying the filtering process across all 8x8 horizontal edges followed by applying the filtering process across all 8x8 vertical edges. The application of the filtering process across the horizontal edges follows top-to-bottom ordering of the horizontal edges to be processed. The application of the filtering process across the vertical edges follows left-to-right ordering of the vertical edges to be processed. If a pixel value is changed by some filtering operation, the updated pixel value is used for the subsequent filtering operations.

H.2 De-ringing filter

The de-ringing filter comprises three sub-processes:

1. threshold determination,
2. index acquisition and
3. adaptive smoothing.

The filter is applied to the pixels on an 8x8 block basis. The 8x8 pixels are processed by referencing 10x10 pixels for each block by adding two overlap rows (one each from top and bottom) and two columns (one each from left and right) from adjacent blocks. The following notation is used to specify the six blocks in a macroblock. For instance, block[5] corresponds to the C_b block whereas block[*k*] is used as a general representation in the following sub-sections.

H.2.1 Threshold determination

The threshold determination process is carried out in two steps:

First, within a block in the decoded image, the maximum and minimum gray value is calculated.

Secondly, the threshold denoted by *thr[k]* and the dynamic range of gray scale denoted by *range[k]* are set as follows:

$$thr[k] = \left(\max imum[k] + \min imum[k] + 1\right) / 2$$

$$range[k] = \max imum[k] - \min imum[k]$$

An additional third process is done only for the luma blocks. Let *max_range* be the maximum value of the dynamic range among four luma blocks. Then,

$$\max\_range = range[k_{\max}]$$

Then, the threshold rearrangement is calculated as follows.

```
for( k=1 ; k<5 ; k++ ){
    if( range[k] < 32 && max_range > =64 )
        thr[k] = thr[kmax];
    if( max_range<16 )
        thr[k] = 0;
}
```

Figure 180: Pseudo-code for Threshold rearrangment in Luma Blocks for Deringing

H.2.2 Index acquisition

Once the threshold value is determined, the remaining operations are performed purely on an 8x8 block basis. Let *rec(h,v)* and *bin(h,v)* be the gray value at coordinates *(h,v)* where *h,v=0,1,2,...,7*, and the corresponding binary index, respectively. Then *bin(h,v)* can be obtained by:

$$bin(h,v) = \begin{cases} 1 & if\ rec(h,v) \geq thr \\ 0 & otherwise \end{cases}$$

Note that *(h,v)* is use to address a pixel in a block, while *(i,j)* is for accessing a pixel in a 3x3 window.

H.2.3 Adaptive smoothing

H.2.3.1 Adaptive filtering

The figure below defines the binary indices at the 8x8 block level, whereas 10x10 binary indices are calculated to process one 8x8 block.

	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	0	0	
	1	0	0	0	0	0	0	0	1	1	
	1	1	0	0	0	0	0	1	1	1	
	1	1	1	0	0	0	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	
	1	1	1	1	1	1	1	1	1	0	
	1	1	1	1	1	1	1	1	0	0	

Figure 181: Example of adaptive filtering and binary index

The filter is applied only if the binary indices in a 3x3 window are all the same, i.e., all "0" indices or all "1" indices. In the above figure, the shaded region represents the pixels to be filtered.

Note: 10x10 binary indices are obtained with a single threshold which corresponds to the 8x8 block.

The filter coefficients used for both intra and non-intra blocks denoted by *coef(i,j),* where *i,j=-1,0,1*, are:

1	2	1
2	4	2
1	2	1

Figure 182: Filter mask for adaptive smoothing

Here the coefficient at the center pixel, i.e., *coef(0,0)*, corresponds to the pixel that is to be filtered. The filter output *flt'(i,j)* is obtained by the following equation:

$$flt'(h,v) = \{8 + \sum_{i=-1}^{1}\sum_{j=-1}^{1} coef(i,j)\cdot rec(h+i,v+j)\ \}//16$$

H.2.3.2 Clipping

The maximum gray level change between the reconstructed pixel and the filtered one is limited according to the quantization parameter, i.e., QP. Let *flt(h,v)* and *flt'(h,v)* be the filtered pixel value and the pixel value before limitation, respectively. Then:

```
if( flt'(h,v) - rec(h,v) > max_diff )
    flt(h,v) = rec(h,v) + max_diff
else if( flt'(h,v) - rec(h,v) < -max_diff )
    flt(h,v) = rec(h,v) - max_diff
else
    flt(h,v) = flt'(h,v)
where max_diff=QP/2 for both intra and inter macroblocks.
```

Figure 183: Pseudo-code for Clipping in Deringing

Annex I
Display Metadata for the Advanced Profile

I.1 Overview

The Advanced Profile bitstream includes a number of fields which provide information useful to the post-decode display process. This information, collectively known as "display metadata" shall all be output by the decoding process. Its use in the display process is optional and thus may be used, for example, to adapt the 4:2:0 pictures to a format appropriate for the output of the display process. Use of the term "shall" below defines the meaning of the field in the bitstream, not the decoder behavior.

I.2 Frame Rate

The Frame Rate fields in the Advanced Profile Sequence Header (FRAMERATEIND (6.1.14.4.1), FRAMERATEEXP (6.1.14.4.4), FRAMERATENR (6.1.14.4.2), FRAMERATEDR (6.1.14.4.3) shall be the target display frame rate for the compressed stream. In the case of content with an interlaced target display (INTERLACE == 1 (6.1.9) && PSF == 0 in the Sequence Header), the frame rate is one-half the field rate of the target display.

The Frame Rate defines the output of the display process, and not necessarily the output of the decoding process (i.e. coded pictures in the compressed stream can occur less frequently than the target display frame rate).

I.2.1 Repeating Progressive Frames

For content with a progressive target display type (INTERLACE == 0 || PSF == 1) and when pull-down has been used (PULLDOWN == 1 (6.1.8)), picture headers contain the Integer field RPTFRM (7.1.1.19). RPTFRM shall represent the number of times the decoded frame may be repeated by the display process.

For example, if a compressed bitstream with 24 frames per second is targeted for a 60-frame-per-second progressive display (INTERLACE == 0 && FRAMERATEEXP == 0x0780), RPTFRM alternates between 1 and 2 in successive frames, and the display process may then display decoded frames for 2 or 3 display frame periods respectively.

I.2.2 Field Order

When PULLDOWN is signaled (PULLDOWN == 1) in the Sequence Header, the order in which to display the two fields of a frame on the target display may be determined by the TFF boolean syntax element (7.1.1.17) in the picture header. If TFF == 1, the decoded field spatially closer to the top of the display shall be signaled to be displayed for one display field period, followed by the decoded field spatially closer to the bottom of the display. If TFF == 0, the decoded field spatially closer to the bottom of the display shall be signaled to be displayed for one display field period, followed by the decoded field spatially located closer to the top of the display.

If PULLDOWN is not signaled (PULLDOWN == 0) in the Sequence Header, decoded pictures shall be signaled as if TFF == 1.

Note: A conformant bitstream does not change the value of TFF except as noted below.

I.2.3 Repeating Fields

When a sequence has an interlaced target display type (INTERLACE == 1 && PSF == 0) and pull-down has been used (PULLDOWN == 1), picture headers contain the boolean field RFF (7.1.1.18). When the RFF == 1, this shall indicate that the display process may display the first field of a field pair again after displaying the second field of the pair – thus extending the duration of the field-pair (frame) to three display field periods.

When a decoded frame is displayed for three display field periods, the subsequent decoded frame shall be flagged with the opposite value of TFF (i.e. if the first decoded frame has TFF == 1 && RFF == 1, the second decoded frame has TFF == 0).

A bitstream shall only change the value of TFF between subsequent decoded frames in this manner.

I.2.4 Frame Interpolation Flag

INTERPFRM (7.1.1.1) is a 1-bit syntax element that indicates that the current temporal region is not appropriate for inter-frame interpolation (INTERPFRM == 0).

Note: Technologies to provide motion-adaptive interpolation between two frames are becoming commercially viable, and could be used for error resiliency (e.g. replacing dropped frames) or bandwidth reduction (e.g. coding fewer frames and interpolating to a higher frame rate on display) independent of any particular compression technology. It has generally been found that such techniques – while performing quite well for small or consistent motions – can cause very significant degradations in perceptual quality when scene motion is high or non-uniform (e.g. a scene with foreground, background, and camera all moving independently). For an example on using frame interpolation to improve the quality of video at low bit rates, please see Kuo et. al 1998.

I.3 Coded Picture Size

The Advanced Profile bitstream provides the ability to change the coded size (in pixels) at Entry Points using the CODED_WIDTH (6.2.13.1) and CODED_HEIGHT (6.2.13.2) fields. This provides an encoder with the ability to reduce or increase the coded picture size and thus the bit rate, without incurring the potential disruption of a sequence change.

To properly support such changes, a decoder needs to know the largest coded picture size it will encounter within a sequence. The sequence header Integer fields MAX_CODED_WIDTH (6.1.6) and MAX_CODED_HEIGHT (6.1.7) shall signal the largest coded picture size within a sequence.

If CODED_WIDTH and CODED_HEIGHT are not present in an Entry Point Header (i.e. CODED_SIZE_FLAG == 0), the coded picture size shall be the maximum coded picture size.

A bitstream shall not have a coded picture size (signaled in the Entry Point Header) which exceeds the maximum coded picture size (as signaled in the Sequence Header).

I.4 Display Geometry Information

The Advanced Profile uses the concept of a Target Display to transport picture geometry information to the Display Process. The actual display being utilized by the display process is not necessarily representative of the Target Display, but the display process may use the display geometry information to optimally render the decoded pictures.

I.4.1 Target Display Size

The horizontal and vertical dimensions of the Target Display (in pixels) may be provided by the DISP_HORIZ_SIZE (6.1.14.1) and DISP_VERT_SIZE (6.1.14.2) fields. The display size represents the entire picture area – including over-scan regions.

When the display size dimensions are not present in the Sequence Header, the display dimensions shall be assumed to be identical to the maximum coded picture dimensions.

I.4.2 Sample Aspect Ratio

The Sample Aspect Ratio (as communicated in the Integer ASPECT_RATIO (6.1.14.3.1) field, or the ASPECT_HORIZ_SIZE (6.1.14.3.2) and ASPECT_VERT_SIZE (6.1.14.3.3) fields) is used in conjunction with the display size to determine the actual picture geometry. By defining the picture geometry using these two values, it is

possible to describe cases where the region which corresponds to a common picture geometry (e.g. 4:3 or 16:9) is a sub-region of the display picture area (i.e. the picture has over-scan regions).

Note: For example, if the display size is 720 pixels wide by 480 pixels high, and uses a sample aspect ratio of 10:11 (width:height), the actual region of the picture which represents a 4:3 geometry would be 704 pixels wide by 480 pixels high (480 x 4/3 x 11/10 = 704)

The region representing the common picture geometry is assumed to be centered within the target display area.

Note: When sample aspect ratio information is not present (i.e. ASPECT_RATIO_FLAG == 0 or for Simple and Main profile), the sample aspect ratio is undefined and application-dependent.

I.4.3 Relating Display Size to Coded Picture Size

In the case where the display size does not match the coded picture size, the display process would normally scale the decoded image to the display size. The center of the display region and the center of the coded picture are considered coincident.

I.5 Pan Scan Regions

The pan scan region is a sub-region of the display region which may be used as an alternative presentation format. The most common application is to display a 4:3 sub-region of 16:9 content.

The Boolean PANSCAN_FLAG (6.2.3) in the Entry Point Header, when set to 1, signals that pan scan windows are present for pictures within that entry point segment. Pan scan information is not necessarily available throughout an entire sequence.

Pan scan regions are described with respect to the display region, not the coded picture size. Thus the geometric relationship between a display and the pan scan information is not affected by changes in the coded picture size.

Note: For a more detailed discussion of the application of this type of Pan Scan Region signaling, consult Taylor 2001.

Note: Pan Scan Regions may also be signaled using alternative methods, such as the Video Index Information defined in SMPTE RP 186-1995. When using Video Index Information, the data can be carried in the elementary stream user data using the method described in SMPTE 328M-2000.

I.6 Post-processing Information

FRMRTQ_POSTPROC (6.1.4.1) and BITRTQ_POSTPROC (6.1.4.2) in the sequence header shall provide the decoder with information so that a decoder may estimate the computational complexity of deblocking and de-ringing operations for this sequence. This estimate in combination with knowledge of the computational capability of the post-processor may be used by the decoder to decide whether to disable or enable deblocking and de-ringing operations as indicated by the POSTPROC meta data bits (see section 7.1.1.27).

Note: The computation of this estimate, and its use in disabling post-processing, are specific to each implementation, and are outside the scope of this standard. An example implementation is provided in Figure 184.

```
ComplexityofPostProcessing = (MAX_CODED_WIDTH * MAX_CODEDHEIGHT) * (FRMRTQ_POSTPROC)
* √ (BITRTQ_POSTPROC);

if (ComplexityofPostProcessing < Threshold1) {
  Deblocking = On;
  Deringing = On;
 }
```

```
else if (ComplexityofPostProcessing < Threshold2) {
   Deblocking = On;
   Deblocking = Off;
}
else {
  Deblocking = Off;
  Deringing = Off;
}
```

Threshold1 and Threshold2 are a measure of the computation capability (higher values of threshold mean greater computational capability) of the post-processing unit, and are platform specific.

Figure 184: Example pseudo-code to show how post-processing fields can be used to control de-ringing and deblocking operations

Annex J
Decoder Initialization Metadata

Certain metadata is required to be delivered to the decoder in order for it to decode the bitstream. While the specification of a particular transport encoding, file system, user input or network protocol that conveys this initialization metadata is not a part of this document, the data format at the interface to the decoder shall be specified herein to enable decoder interoperability with any transport or delivery system.

Note: See Annex L for an encoding of this metadata that could be used as a basis for specific encodings.

The metadata items defined in this annex shall be made available to the decoder prior to commencement of the decoding process. The delivery of this metadata is specific to the simple and main profiles only. See 4.12 for definition of terms used in this annex.

J.1 Initialization Metadata Elements

The following metadata elements shall be communicated to the decoder by the transport demultiplexer, file reader, or other means:

J.1.1 Profile (PROFILE)

PROFILE specifies the encoding profile used to produce the sequence, and shall be set to 0, 4 or 12 to indicate Simple, Main or Advanced profile respectively. Other values shall be SMPTE Reserved.

Note: When encoding, this uses values 0, 4 & 12 so at least 4 bits are implied though 2 bits minimum could be used.

J.1.2 Level (LEVEL)

LEVEL shall specify the encoding level used to produce the sequence, and shall be set to 0 or 2 to indicate Low and Medium levels respectively for the simple profile. LEVEL shall be set to 0, 2 or 4 to indicate Low, Medium and High levels respectively for the main profile. For the advanced profile, LEVEL shall take a value from 0 through 4, corresponding to levels L0 through L4 in sequence. All other values shall be SMPTE Reserved.

The following elements are defined only for the Simple and Main profiles. These syntax elements shall be ignored for the Advanced profile and shall not be used by the decoder.

Note: When encoding, this uses values 0 to 4 so 3 bits minimum are needed.

J.1.3 Horizontal Size of Picture (HORIZ_SIZE)

HORIZ_SIZE shall specify the horizontal size of the coded picture in pixels ranging from 2 to 8192 in units of 2 pixels. Values greater than 8192 shall be SMPTE Reserved.

CodedWidth = HORIZ_SIZE.

Note: When encoding, this uses values from 2 to 8192 in 2 pixel steps so 12 bits minimum are needed.

J.1.4 Vertical Size of Picture (VERT_SIZE)

VERT_SIZE shall specify the vertical size of the coded picture in pixels ranging from 2 to 8192 in units of 2 pixels. Values greater than 8192 shall be SMPTE Reserved.

CodedWidth = VERT_SIZE.

Note: When encoding, this uses values from 2 to 8192 in 2 pixel steps so 12 bits minimum are needed.

J.1.5 HRD Rate (HRD_RATE)

HRD_RATE shall specify the peak transmission rate *R* in bits per second. The range is from 1 to 65536.

Note: When encoding, 16 bits minimum are needed.

J.1.6 HRD Buffer Size (HRD_BUFFER)

HRD_BUFFER shall signal the buffer size *B* in milliseconds. Thus size of the buffer *B* in bits is given by (HRD_BUFFER * HRD_RATE)/1000. The range is from 1 to 65536.

Note: This syntax element is different from the syntax element defined in section 6.1.15.1.

Note: When encoding, 16 bits minimum are needed.

J.1.7 Quantized Frame Rate for Post processing Indicator (FRMRTQ_POSTPROC)

FRMRTQ_POSTPROC may be present and shall signal the (quantized) frame rate information for controlling the strength of post-processing operation. The decoding procedure for FRMRTQ_POSTPROC may be performed as specified in Table 258. Annex I.6 describes a mechanism for controlling post-processing using the post-processing indicators.

Note: When encoding, 7 is the upper limit so 3 bits minimum are needed.

J.1.8 Quantized Bit Rate for Post processing Indicator (BITRTQ_POSTPROC)

BITRTQ_POSTPROC may be present and shall signal the (quantized) bit rate information for controlling the strength of post-processing operation. The decoding procedure for BITRTQ_POSTPROC may be performed as specified in Table 258. Annex I.6 describes a mechanism for controlling post-processing using the post-processing indicators.

Note: When encoding, the upper limit is 31, so 7 bits minimum are needed.

Table 258: Decoding Procedure for Post-processing Indicators in Simple/Main Profile

if (FRMRTQ_POSTPROC == 7) {
“frame rate is around 30 frames/second or more
}
else {
“frame rate is around “(2+FRMRTQ_POSTPROC*4)” frames/second
}
if (BITRTQ_POSTPROC == 31) {
“bit rate” is around 2016 kbps or more
}
else {
“bit rate” is around “(32 + BITRTQ_POSTPROC * 64)” kbps
}

J.1.9 Loop Filter Flag (LOOPFILTER)

LOOPFILTER is a Boolean that shall indicate whether loop filtering is enabled for the sequence. If LOOPFILTER == 0, then loop filtering shall not be enabled. If LOOPFILTER == 1, then loop filtering shall be enabled. If the PROFILE

syntax takes the value corresponding to simple profile, the LOOPFILTER syntax element shall have the value 0. See section 8.6 for a description of loop filtering.

Note: When encoding, a minimum of 1 bit is needed.

J.1.10 Multi-resolution Coding (MULTIRES)

MULTIRES is a Boolean that shall indicate whether the frames may be coded at smaller resolutions than the specified frame resolution. Resolution changes shall only be allowed on I pictures. If MULTIRES == 1, then the frame level RESPIC syntax element shall be present which indicates the resolution for that frame. If MULTIRES == 0, then RESPIC shall not be present. See section 8.1.1.3 for a description of multi-resolution decoding in I pictures.

Note: When encoding, a minimum of 1 bit is needed.

J.1.11 FAST UV Motion Compensation Flag (FASTUVMC)

FASTUVMC is Boolean that shall control the subpixel interpolation and rounding of color-difference motion vectors. If FASTUVMC == 1, then the color-difference motion vectors that are at quarter pel offsets shall be rounded to the nearest half or full pel positions If FASTUVMC == 0, then no special rounding or filtering shall be done for color-difference. See section 8.3.5.4.2 for details on how color-difference motion vector computation is performed for the two cases. (Informative – The purpose of this mode is speed optimization of the decoder). FASTUVMC shall be 1 for the Simple Profile.

Note: When encoding, a minimum of 1 bit is needed.

J.1.12 Extended Motion Vector Flag (EXTENDED_MV)

EXTENDED_MV is a Boolean that shall indicate whether extended motion vectors are enabled (value 1) or disabled (value 0). This bit shall always set to zero for the Simple Profile. For the Main Profile, the extended motion vector mode shall indicate the possibility of extended motion vectors in P and B pictures.

Note: When encoding, a minimum of 1 bit is needed.

J.1.13 Macroblock Quantization Flag (DQUANT)

DQUANT shall indicate whether or not the quantization step size may vary within a frame. If DQUANT == 0, then only one quantization step size (i.e. the frame quantization step size) shall be used per frame. If DQUANT == 1 or 2, then the quantization step size may vary within the frame. In simple profile, DQUANT shall be 0. In the main profile, if MULTIRES == 1, DQUANT shall be 0. Values greater than 2 shall be SMPTE Reserved. See section 7.1.1.31 for a description of DQUANT.

Note: When encoding, a minimum of 2 bits are needed.

J.1.14 Variable Sized Transform Flag (VSTRANSFORM)

VSTRANSFORM is a Boolean that shall indicate whether variable-sized transform coding is enabled for the sequence. If VSTRANSFORM == 0, then variable-sized transform coding shall not be enabled. If VSTRANSFORM == 1, then variable-sized transform coding shall be enabled. See section 8.3.6.2 for a description of variable-sized transform coding.

Note: When encoding, a minimum of 1 bit is needed.

J.1.15 Overlapped Transform Flag (OVERLAP)

OVERLAP is a Boolean that shall indicate whether Overlapped Transforms (Section 7.4) are used. If OVERLAP == 1, then Overlapped Transforms may be used. If OVERLAP == 0, Overlapped Transforms shall not be used.

Note: When encoding, a minimum of 1 bit is needed.

J.1.16 Sync Marker Flag (SYNCMARKER)

SYNCMARKER is a Boolean that shall indicate whether synchronization markers may be present in the bitstream. This bit shall always be set to zero in the simple profile. In the main profile, the synchronizations markers may be present if

SYNCMARKER == 1, and the markers shall not be present if SYNCMARKER == 0. See section 8.8 for description of synchronization markers.

Note: When encoding, a minimum of 1 bit is needed.

J.1.17 Range Reduction Flag (RANGERED)

RANGERED is a Boolean that shall indicate whether range reduction is used for each frame. If RANGERED == 1, then there shall be a syntax element in each frame header (RANGEREDFRM) that indicates whether range reduction is used for that frame. If RANGERED == 0, the syntax element RANGEREDFRM shall not be present, and range reduction shall not used. RANGERED shall be set to zero in simple profile.

Note: When encoding, a minimum of 1 bit is needed.

J.1.18 Maximum Number of consecutive B frames (MAXBFRAMES)

MAXBFRAMES shall indicate the presence of B frames between I or P frames. If MAXBFRAMES == 0, then there shall be no B frames in the sequence. If MAXBFRAMES is not equal to zero, B Frames may be present in the sequence. Values greater than 7 are SMPTE Reserved.

Note: When encoding, a minimum of 3 bits are needed.

J.1.19 Quantizer Specifier (QUANTIZER)

QUANTIZER shall indicate the quantizer used for the sequence. The quantizer types shall be as defined in Table 259 below.

Table 259: Quantizer specification

Value	Meaning
0	Quantizer implicitly specified at frame level
1	Quantizer explicitly specified at frame level
2	Non-uniform quantizer used for all frames
3	Uniform quantizer used for all frames
other	SMPTE Reserved

J.1.20 Frame Interpolation Flag (FINTERPFLAG)

FINTERPFLAG is a Boolean that shall indicate if the syntax element INTERPFRM is present in the picture header. If FINTERPFLAG == 0, then INTERPFRM shall not be present in picture headers. If FINTERPFLAG == 1, INTERPFRM shall be present in picture headers.

Note: When encoding, a minimum of 2 bits are needed.

J.2 Initialization Interface Data Structure

This section defines a common initialization interface data structure for all profiles.

The size elements may be encoded according to a 64 bit sequence header data structure ‘STRUCT_A’ as shown in Table 260 below for the Simple and Main profiles. For the Advanced profile, STRUCT_A may be 8 consecutive zero bytes.

See Annex L for a recommended bitstream layout.

Table 260: Sequence Header Data Structure STRUCT_A for Simple and Main Profiles

STRUCT_SEQUENCE_HEADER_A {	Number of bits	Descriptor	Reference

VERT_SIZE	32	uimsbf	J.1.4
HORIZ_SIZE	32	uimsbf	J.1.3
}			

The level and HRD parameters may be encoded according to a 96 bit sequence header data structure 'STRUCT_B' as shown below in Table 261 for the Simple and Main profiles and in Table 262 for the Advanced Profile. Additional fields are included in STRUCT_B, and are described below.

J.2.1 Constant Bitrate Sequence (CBR)

CBR shall indicate that the content was generated using a constant bitrate model (set to 1) or not (set to 0).

Note: This bit is usually not used by the decoder.

J.2.2 Reserved1 (RES1)

RES1 shall be set to zero. Other values shall be SMPTE Reserved.

J.2.3 Integer Frame Rate (FRAMERATE)

FRAMERATE is a 32-bit unsigned word in the bitstream which may signal the rounded frame rate (fps) of the encoded clip. FRAMERATE should be set to 0xffffffff if it is not known, unspecified, or non-constant.

Note: This value is usually not used by the decoder.

J.2.4 Reserved2 (RES2)

RES2 shall be set to zero. Other values shall be SMPTE Reserved.

Table 261: Sequence Header Data Structure STRUCT_B for Simple and Main Profiles

STRUCT_SEQUENCE_HEADER_B {	Number of bits	Descriptor	Reference
LEVEL	3	uimsbf	J.1.2
CBR	1	uimsbf	J.2.1
RES1	4	uimsbf	J.2.2
HRD_BUFFER	24	uimsbf	J.1.6
HRD_RATE	32	uimsbf	J.1.5
FRAMERATE	32	uimsbf	J.2.3
}			

Table 262: Sequence Header Data Structure STRUCT_B for Advanced Profile

STRUCT_SEQUENCE_HEADER_B {	Number of bits	Descriptor	
LEVEL	3	uimsbf	J.1.2
CBR	1	uimsbf	J.2.1
RES1	4	uimsbf	J.2.2
RES2	56	uimsbf	J.2.4
FRAMERATE	32	uimsbf	J.2.3
}			

The remaining metadata elements may be encoded according to a 32 bit sequence header data structure 'STRUCT_C' as shown in Table 263 below for the Simple and Main profile, and in Table 264 for the Advanced profile.

Table 263: Sequence Header Data Structure STRUCT_C for Simple and Main Profiles

STRUCT_SEQUENCE_HEADER_C {	Number of bits	Descriptor	
PROFILE	4	uimsbf	J.1.1
FRMRTQ_POSTPROC	3	uimsbf	J.1.7
BITRTQ_POSTPROC	5	uimsbf	J.1.8
LOOPFILTER	1	uimsbf	J.1.9
Reserved3	1	uimsbf	See below
MULTIRES	1	uimsbf	J.1.10
Reserved4	1	uimsbf	See below
FASTUVMC	1	uimsbf	J.1.11
EXTENDED_MV	1	uimsbf	J.1.12
DQUANT	2	uimsbf	J.1.13
VSTRANSFORM	1	uimsbf	J.1.14
Reserved5	1	uimsbf	See below
OVERLAP	1	uimsbf	J.1.15
SYNCMARKER	1	uimsbf	J.1.16
RANGERED	1	uimsbf	J.1.17
MAXBFRAMES	3	uimsbf	J.1.18
QUANTIZER	2	uimsbf	J.1.19
FINTERPFLAG	1	uimsbf	J.1.20
Reserved6	1	uimsbf	See below
}			

Table 264: Sequence Header Data Structure STRUCT_C for Advanced Profile

STRUCT_SEQUENCE_HEADER_C {	Number of bits	Descriptor	
PROFILE	4	uimsbf	J.1.1
Reserved7	28	uimsbf	See below
}			

The fields are as described above in section J.1. In addition:

Reserved3 shall be set to zero and other values shall be forbidden.

Reserved4 shall be set to one and other values shall be forbidden.

Reserved5 shall be set to zero and other values shall be forbidden.

Reserved6 shall be set to one and other values shall be forbidden.

Reserved7 shall be set to zero and other values shall be forbidden.

Annex K
Encoder Overview and Internal Representation (Informative)

K.1 Coding Description

Figure 185 and Figure 186 illustrate the basic steps used to encode a series of pictures. Note that the compression process uses block-based motion predictive coding to reduce temporal redundancy and transform coding to reduce spatial redundancy.

A highly level summary of the encoder is as follows.

- A frame is decomposed into blocks.
- For an intra-coded block:
  - The block is transformed using an 8x8 transform.
  - Before applying the transform, an overlap operation can be applied to neighboring intra-coded blocks.
  - Quantization is applied to the transform coefficients, and DC/AC prediction can be applied to the quantized coefficients to remove additional redundancy.
  - The DC coefficient is coded by a VLC. A zigzag scan is applied to the AC coefficients, and the run-levels are encoded with another VLC.
- For an inter-coded block:
  - The block is predicted from motion compensated block in the previous frame. If the frame is bidirectional predicted, the block can also be predicted from motion compensated blocks in the next frame.
  - The motion vectors themselves are predicted from the motion vectors of neighboring blocks, and the motion vector differentials are coded with a VLC.
  - An 8x8, or 8x4, or 4x8, or 4x4 transform is applied to the prediction error of the block. The transform coefficients are quantized.
  - A zigzag scan is applied to the coefficients, and the run-levels are encoded with a VLC.
- In both intra-coded pictures, and inter-coded pictures, macroblock level flags such as ACPRED flag or SKIPMB flag can be jointly encoded at the frame level using bitplane coding.

The encoding of an 8x8 intra-coded block (after overlap operation) is shown in Figure 185. The encoding of an 8x8 inter-coded block is shown in Figure 186.

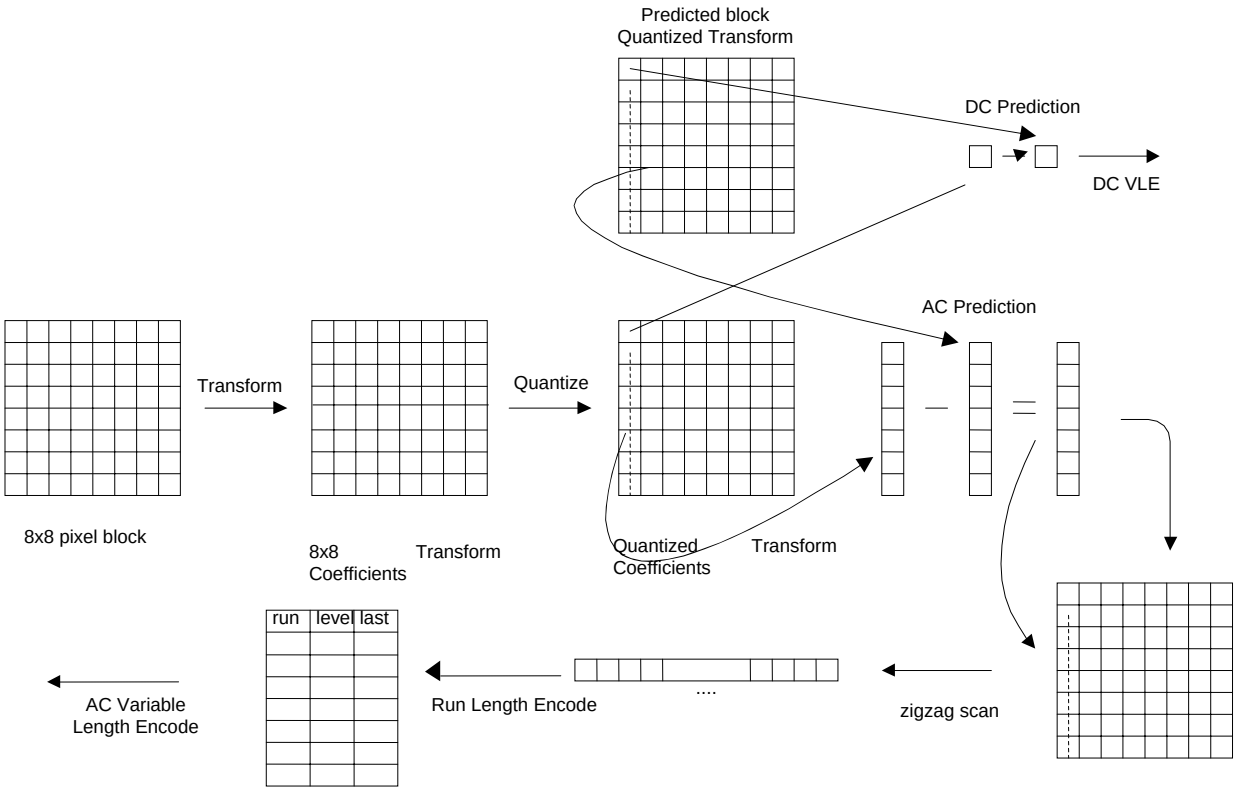


Figure 185: Coding of Intra blocks

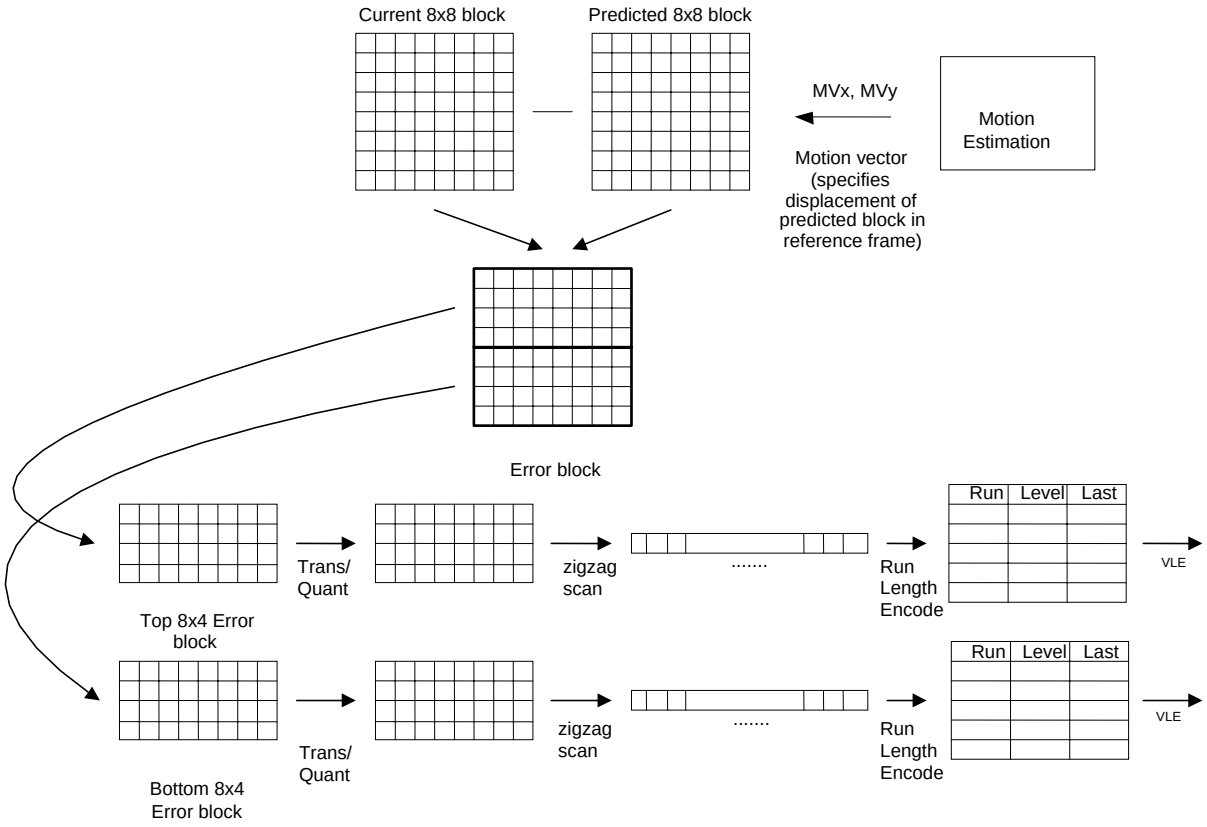


Figure 186: Coding of Inter blocks

K.2 Internal Representation of a Frame

Implementations typically work with 16x16 pixels of luma component per MB, and thus the internal frame dimensions used in the decoder are multiples of 16. The internal luma arrays used for decoding the video have dimensions of 16 * ((‘frame height’ + 15) / 16) rows by 16 * ((‘frame_width’ + 15) / 16) columns. The internal color-difference arrays used for decoding the video have dimensions half as much as the luma arrays in both the horizontal and vertical axes. Note that all decoding operations use the internal array size, but the output of the decoder is the cropped array obtained by taking the first rows 0 to ‘frame height’ - 1 and first columns 0 to ‘frame width’ - 1 out of the internal luma array, and by taking the first rows 0 to ‘frame height’/2 -1 and first columns 0 to ‘frame width’ -1 out of the internal color-difference array.

Annex L
Bitstream Metadata Serialization

The metadata elements for decoder initialization as defined in Annex J may be carried in a demarcated container such as a disk file format, or within a transport stream. The decoder may be passed these elements via auxiliary data paths, or as a single concatenated bitstream. This annex defines a concatenation of the decoder initialization metadata and compressed frame data so as to embed the information necessary for the decoding process into a common minimal serial bitstream that can be used in defining file or streaming format encodings of the elementary bitstream. This concatenation is applicable to all profiles of the standard.

L.1 General Layout

There are two layers – the sequence layer and the frame layer, which are shown in Table 265 and Table 266 respectively. There shall be only one sequence layer in a serialized bitstream. The frame layer shall immediately follow the sequence layer, or the preceding frame layer.

L.2 Sequence Layer

The sequence layer metadata may be formed as defined in Table 265.

Table 265: Sequence Layer Data Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0xC5								NUMFRAMES : 24																							
0x00000004																															
STRUCT_C (refer to Table 263 and Table 264)																															
STRUCT_A (VERT_SIZE) (refer to Table 260 and Annex J.2)																															
STRUCT_A (HORIZ_SIZE)																															
0x0000000C																															
STRUCT_B (LEVEL:3 | CBR: 1 | RES1: 4 | HRD_BUFFER:24) (refer to Table 261 and Table 262)																															
STRUCT_B (HRD_RATE)																															
STRUCT_B (FRAMERATE)																															

The fields in the sequence layer of Table 265 shall be as described in Annex J, with field NUMFRAMES described below:

Number of Compressed Frames (NUMFRAMES)

NUMFRAMES is a 24 bit unsigned integer field that shall denote the number of compressed frames in the serialization. The value of 0xffffff shall indicate an indefinitely long sequence, or a sequence longer than 16777214 frames.

L.3 Frame Layer

Table 266: Frame Layer Data Structure

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KEY	RES							FRAMESIZE : 24																							
TIMESTAMP																															
FRAMEDATA (byte aligned)																															

The fields in the frame layer are described below:

Key Frame Indicator (KEY)

KEY == 1 shall indicate an intra frame for simple and main profiles or an entry point for advanced profile, and shall be set to 0 otherwise.

Reserved (RES)

Seven bits marked RES shall be set to zero. Other values shall be SMPTE Reserved.

Compressed Frame Size (FRAMESIZE)

FRAMESIZE shall be the size of the FRAMEDATA field in bytes. The subsequent frame layer shall be located at an offset of 8 + FRAMESIZE bytes from the start of the current frame layer. FRAMESIZE set to 0 or 1 shall indicate a skipped P frame.

Time stamp in ms (TIMESTAMP)

TIMESTAMP shall be the 32-bit time stamp of the current frame in milliseconds relative to the time stamp of the first frame. TIMESTAMP can be used to signal the target display rate of the compressed steam.

Compressed Frame Data (FRAMEDATA)

For simple and main profiles, FRAMEDATA shall be the compressed frame level data, padded to a byte boundary. Padded bits are MSB and shall be set to zero. Other values shall be SMPTE Reserved.

For advanced profile bitstreams, FRAMEDATA shall minimally contain a frame start code and corresponding frame data. If the frame is an entry point, FRAMEDATA shall contain the entry point start code, entry point header, frame start code and frame data. If the frame is the first frame of a sequence, FRAMEDATA shall contain the sequence start code, sequence header, entry point start code, entry point header, frame start code and frame data. Refer to Annex G for details on start codes and bitstream construction constraints.

Note: The data elements STRUCT_A, STRUCT_B and STRUCT_C (defined in Annex J.2), their constituent fields, field sizes, and insertion of reserved bits and their values are chosen to maintain compatibility with a pre-existing bitstream serialization format and existing Simple and Main profile decoders.

The sequence layer and frame layer data structures (except FRAMEDATA) shall be represented as a sequence of 32 bit unsigned integers. In Table 265 and Table 266, 0 in the first row represents LSB. Each integer, except STRUCT_C shall be serialized in little-endian byte order. STRUCT_C and FRAMEDATA shall be serialized in big-endian byte order. Therefore, the constant byte value 0xC5 shall be present in the fourth byte of the serialization.