Proposed SMPTE Standard for Television

Date: 2004-03-31

SMPTE CD xxxM

SMPTE Technology Committee C24 on Video Compression Technology

Proposed SMPTE Standard for Television: VC-9 Compressed Video Bitstream Format and Decoding Process

Warning

This document is not a SMPTE Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as a SMPTE Standard. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation. Distribution does not constitute publication.

Document type: Standard

Document subtype:

Document stage: Committee Draft 1

Document language: English

595 W. Hartsdale Ave.
White Plains, NY 10607
+1 914 761 1100
Fax +1 914 xxx xxxx
E-mail eng@smpte.org
Web 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 document is proposed standard VC-9 submitted to SMPTE Technology Committee C24.

Introduction

This document was prepared for the primary purpose of documenting the bitstream format and decoding process used in the VC-9 video codec. It defines the bitstream syntax, semantics and constraints for compressed video bitstreams and describes the complete process required to decode them.

SMPTE Standard for Television: VC-9 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-9 format, and specifies constraints that are required for conforming 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 the VC-9 standard include progressive and interlaced video sampled in the form of Y luminance samples and U,V chrominance in 8-bit sample values resulting from a 4:2:0 sampling grid. The decoding process outputs 8-bit video samples corresponding to the original 4:2:0 sampling grid. The display rendering process by which decoded YUV 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-9. A VC-9 bitstream may convey additional metadata and user data which shall be accounted for in the buffer model. Metadata may be included in VC-9 streams that is not used by the decoding process, but it passed to the display rendering process for the identification and reconstruction of the sampled video format, aspect ratio, color space, etc.

2 References

2.1 Normative References

2.2 Informative References

[HRD] 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.

[MPEG2] *ISO/IEC 138180-2, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video (MPEG-2/H.262),* Annex C "Video Buffering Verifier," 2nd Edition, 2000.

[H263] *Video Coding for Low Bit Rate Communication, ITU-T recommendation H.263,* Annex B "Hypothetical Reference Decoder," Jan 1998.

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

[RP] Proposed SMPTE Recommended Practice : VC-9 Transport Encoding.

3 Overview

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

3.1 Syntax Overview

The syntax of this standard consists of hierarchical layers – sequence, entry-point, picture, slices, macroblocks (MB), and blocks. A picture is decomposed into macroblocks, each of which consists of four blocks. A slice is one or more contiguous rows of macroblocks. An entry-point provides random access to a particular picture. The standard specifies a syntax and decoding process both for progressive and interlace video. Interlaced pictures may be coded as a single

frame, or as two fields. Progressive picture shall be coded as a single frame. Both progressive and interlace picture may be mixed in the same sequence. Each picture may be coded as an I-picture, or as a P-picture, or as a B-picture. There are three profiles in VC-9: simple, main and advanced.

3.2 Decoding Process Overview

An overview of the decoding process, as defined in this document, is shown in Figure 1. The parts of the process, with the exception of Out-of-Loop Processing, must be performed as described in this document to provide successful decoding of the compressed bit stream. Non-conforming implementations of in-loop processes can create errors in the reconstructed pictures which will be exacerbated by the temporal prediction loop.

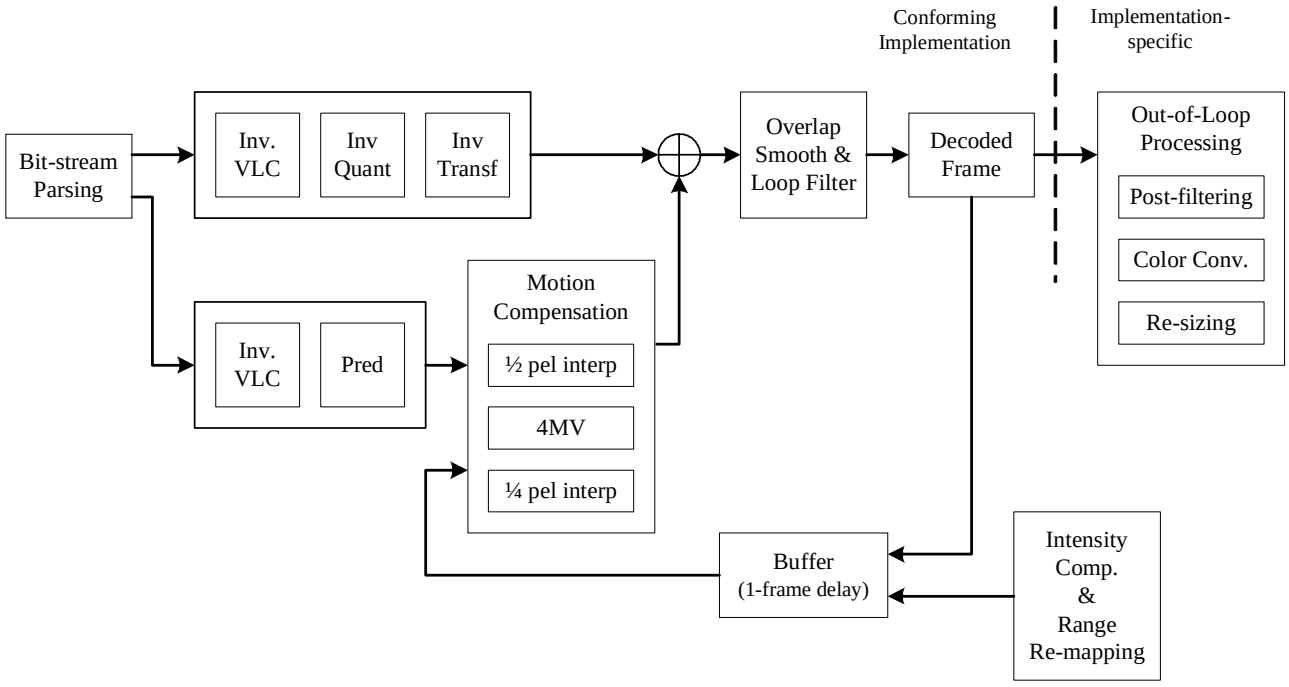


Figure 1: Decoding Process Block Diagram

Out-of-loop processing may be assisted by information carried in the compressed bit stream (e.g. display aspect ratio or post-filtering level). However, because the effect of such processing does not propagate in the prediction loop (i.e. errors do not magnify through feedback), and the implementation of such processing may vary depending on the architecture of the overall system implementation, the normative definition of these out-of-loop processes is beyond the scope of this document.

3.3 Transport Requirements (Normative)

The elementary stream of this standard shall be encoded into some transport layer, such as MPEG-2 and ASF. For simple and main profiles of this standard, certain syntax elements of the video stream shall be communicated as meta-data to the decoder by the transport layer. These meta-data elements are: a) coded width and coded height of video in simple and main profiles, b) levels corresponding to simple and main profiles, and c) pointer to the coded bitstream, and its size for coded picture in simple/main profiles. In advanced profile, the coded width and height of video are communicated to the decoder by the transport layer if the syntax element PIC_SIZE_FLAG = 0 in the sequence header. For more information on the communication of VC-9 syntax elements as meta-data via the transport layer, see [RP].

3.4 Document structure

Section **Error! Reference source not found.** presents notation and definition of terms used in this document. Section **Error! Reference source not found.** describes the input source format, and the hierarchical elements of the syntax. Section **Error! Reference source not found.** describes the syntax and semantics of the sequence and entry-point layer. Section **Error! Reference source not found.** describes the syntax and semantics of the picture, slice, macroblock, and block layers of a progressive picture. Section **Error! Reference source not found.** describes the decoding process of a progressive picture. Section **Error! Reference source not found.** describes the syntax and semantics of the picture, slice, macroblock and block layers of an interlace-coded picture. Section **Error! Reference source not found.** describes the decoding process of an interlace picture. In sections **Error! Reference source not found.** and **Error! Reference source not found.**, the interlace picture coded as two fields is described first followed by the interlace picture coded as a frame.

4 Notation

The following notation is used in this document.

4.1 Compliance Notation

As used in this document, the capitalized keywords ‘shall’ and ‘shall not’ denote mandatory provisions of the specification. The capitalized keyword ‘should’ is used to indicate a provision that is recommended but not mandatory. The capitalized keyword ‘may’ denotes a feature whose presence does not preclude compliance; that may or may not be present at the option of the implementer.

4.2 Arithmetic Operators

+	Addition.
–	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment.
– –	Decrement.
*	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.
//	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.
##	Rest of the line is a comment.
| |	Absolute value. | x | = x , when x > 0 | x | = 0, when x == 0 | x | = –x, when x < 0
%	Modulus operator. Defined only for positive numbers.
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
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 4.9 for definition)
median4 ()	Median of 4 values (see section 4.9 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.

4.6 Assignment

=	Assignment operator.

4.7 Mnemonics

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

uimsbf	Unsigned integer, most significant 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

4.8 Bitstream Parsing Operations

The pseudo-code examples use the following bitstream parsing operations

get_bits(n)	Reads n bits from the bitstream and returns the value. get_bits(0) is defined to be zero.
vlc_decode()	Decodes the next variable-length codeword in the bitstream and returns the decoded symbol

4.9 Definition of Median3 and Median4 Functions

The functions median3() and median4() are used in some of the pseudocode examples in this spec. The functions median3 and median4 are computed as illustrated in the following pseudocode examples.

```
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 (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
}
```

4.10 Definition of Terminology

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

access unit : A coded representation of a single picture in a VC-9 elementary stream.

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

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

B-frame picture: A frame structure B-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.

backward compatibility: A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.

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).

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

bitrate: The rate at which the coded bitstream is delivered from the storage medium 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.|

chrominance component: A matrix, block or single sample representing one of the two colour difference signals related to the primary colours in the manner defined in the bitstream. The symbols used for the chrominance signals are U and V.

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 pictures.

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

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

component: A matrix, block or single sample from one of the three matrices (luminance and two chrominance) 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.

decoder: An embodiment of a decoding process.

decoding process: The process defined in VC-9 whereby a serialized bitstream is converted to an array of 8-bit YUV samples with 4:2:0 color subsampling. In other words, the decoding algorithm. The VC-9 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, YUV matrix coefficients, pixel aspect ratios, etc., and may display the images with frequency and timing different from the sampled rate.

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

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, that reads a stream of input pictures and produces a valid coded bitstream as.

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

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.

forbidden: The term "forbidden" when used in the clauses defining the coded bitstream indicates that the value shall never be used.

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).

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 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.

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.

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

interlace: The property of conventional television frames where alternating lines of the frame representdifferent instances in time. In an interlaced frame, one of the field 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.

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

I-frame picture: A frame structure I-Picture.

I-picture; intra-coded picture: A picture coded using information only from itself.

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

luminance component: A matrix, block or single sample representing a monochrome representation of the signal and related to the primary colours in the manner defined in the bitstream. The symbol used for luminance is Y.

macroblock: The four 8 by 8 blocks of luminance data and the two corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance 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: 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.

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

P-field picture: A field structure P-Picture.

P-frame picture: A frame structure P-Picture.

P-picture; predictive-coded picture: A picture that is coded using motion compensated prediction from past reference fields or frame.

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.

picture: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance 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.

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.

profile: A defined subset of the syntax of the VC-9 standard, with a specific set of coding tools, algorithms, and syntax associated with it. There are three profiles in VC-9: simple, main and advanced. Note that the profiles in VC-9 are not heirarchical. Main profile is not a subset of advanced profile, nor is it a superset.

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

random access: A random access point in the bitstream is defined by the following guarantee: If decoding begins at this point, there will be no decoding dependency on any data preceding this point, and all frames needed for display after this point 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. Luminance and chrominance 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. Luminance and chrominace 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.

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.

reserved: The term "reserved" when used in the clauses defining the coded bitstream, indicates that the value may be used in the future for SMPTE defined extensions.

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".

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

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

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 (system and video): 32-bit codes embedded in that coded bitstream that are unique.

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.

top layer: The topmost layer (with the highest layer_id) of a scalable hierarchy.

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

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

Video Codec 9 (VC-9): 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.

4.11 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, it is enclosed in square brackets.

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 Recommendation. Unless specified otherwise, all unused or spare bits are set to “0”. All values of syntax not explicitly defined in this document are, by default, reserved for future use.

5 Source Coder/Decoder

5.1 Progressive Coding Mode

5.1.1 Input/output Format

Figure 2 below defines the YUV 4:2:0 sampling grid, which is the input/output format. The figure also shows the spatial relationship between the luma and chroma samples.

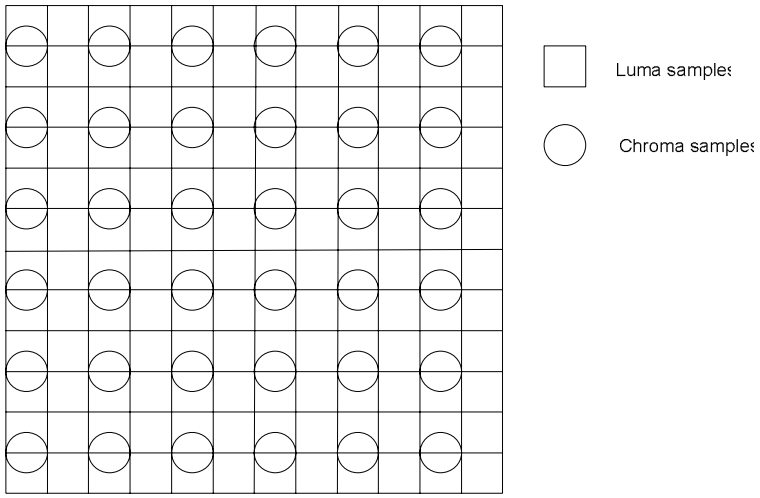


Figure 2: 4:2:0 Luma and chroma sample horizontal and vertical positions

5.1.2 Hierarchical Elements

The syntax of this standard consists of hierarchical layers – sequence, entry-point, picture, slices, macroblocks (MB), and blocks. In the advanced profile, an optional entry-point layer may be present between the sequence and picture layers to signal a random access 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 is defined to contain one or more contiguous rows of macroblocks in their original left-to-right order. Note that a slice always begins at the first macroblock of a row, and ends at the last macroblock of the same or another row. The entry-point and slice layers are present only in advanced profile.

Figure 3 illustrates the picture, macroblock, slice and block layers.

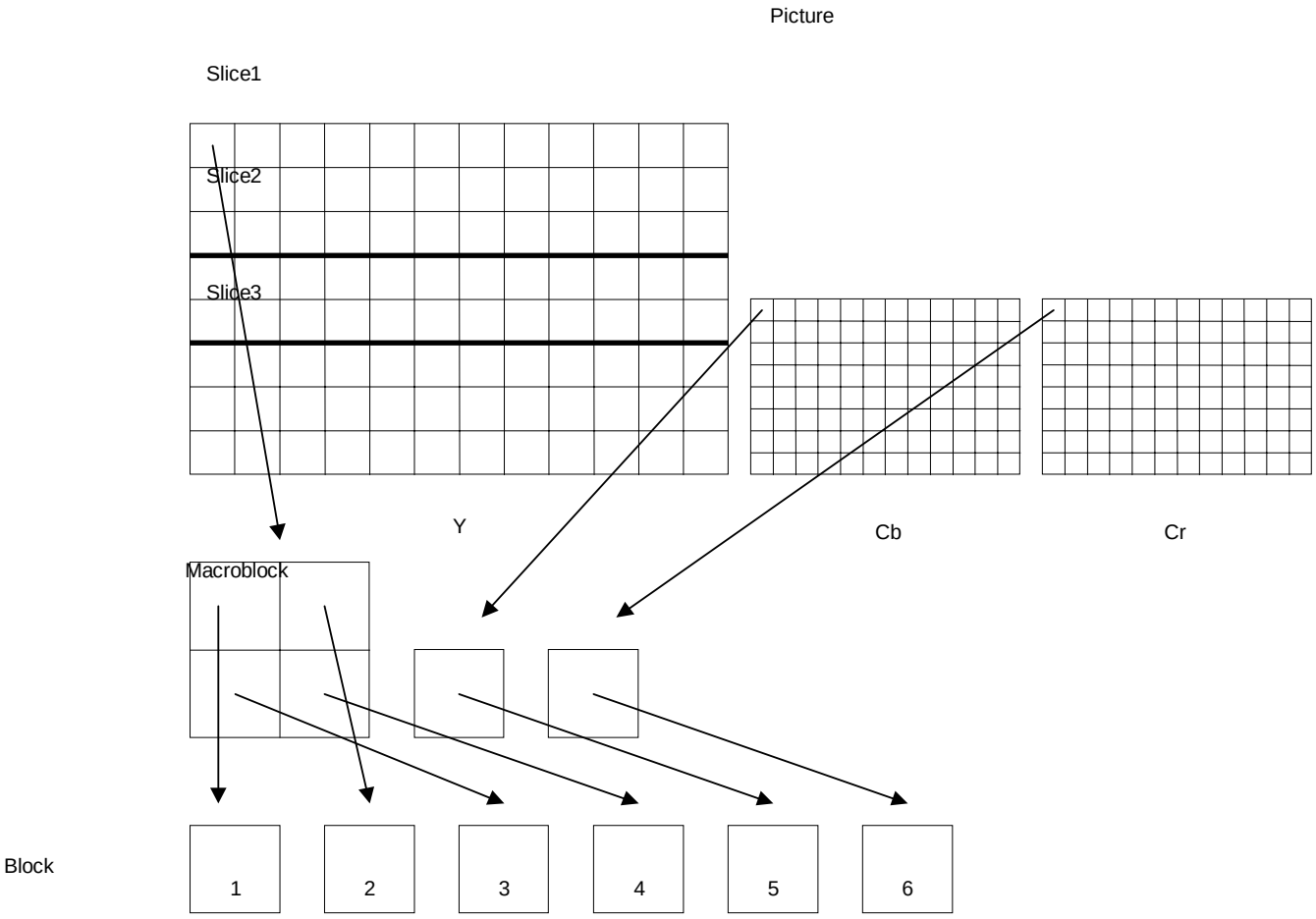


Figure 3: Coding Hierarchy showing Picture, Slice, Macroblock and Block layers

5.1.3 Coding Description (Informative)

This section is not an integral part of this standard.

The compression process uses block-based motion predictive coding to reduce temporal redundancy and transform coding to reduce spatial redundancy. Figure 4 and Figure 5 illustrate the basic steps used to compress the video data in the VC-9 compression algorithm.

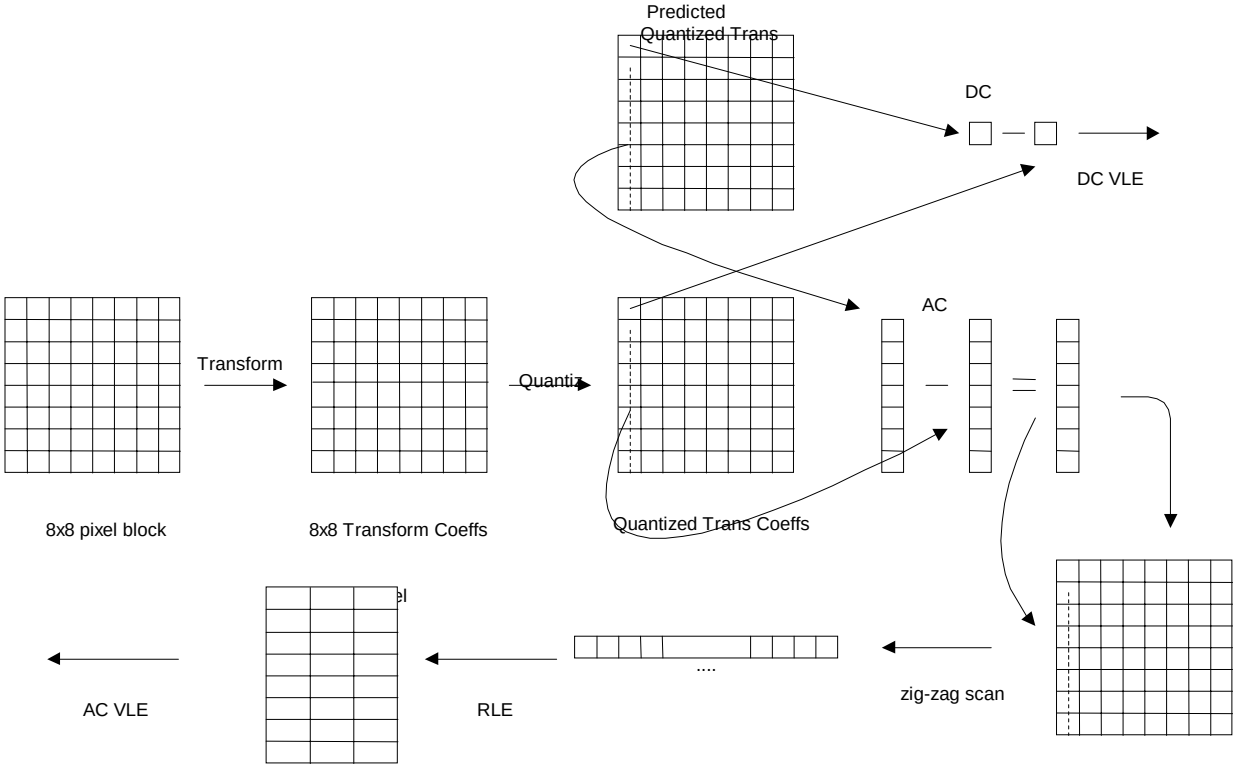


Figure 4: Coding of Intra blocks

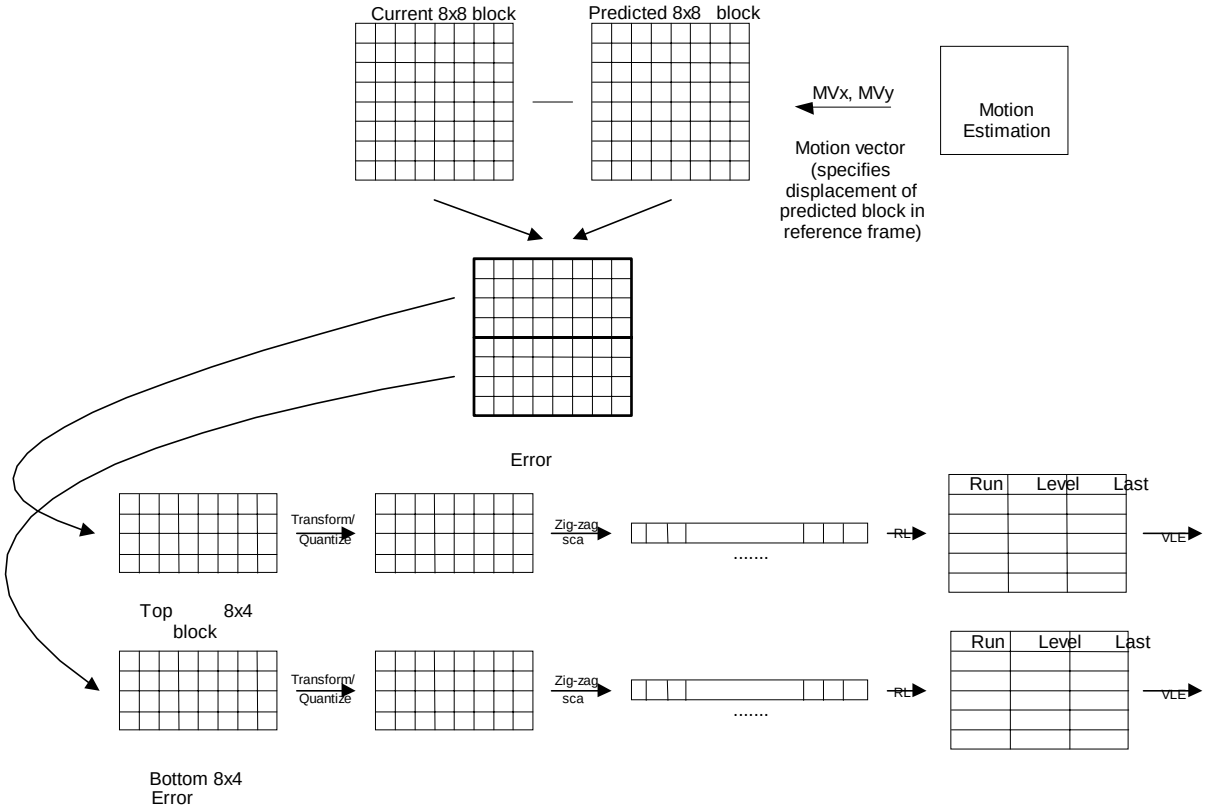


Figure 5: Coding of Inter blocks

5.2 Interlace Coding Mode

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

Figure 2 shows the spatial relationship between the luma and chroma samples in the YUV 4:2:0 format. Figure 6 shows the relationship between vertical sample position and sampling time instant. Note that Figure 6 does not show the spatial relationship between horizontal and vertical sampling positions.

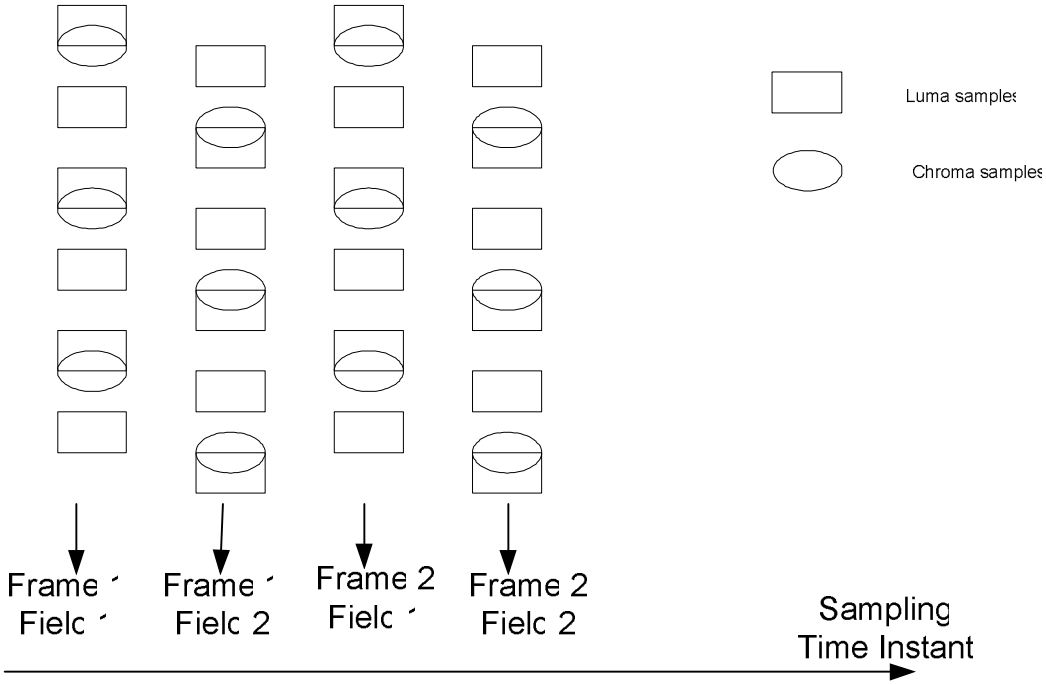


Figure 6: 4:2:0 Luma and chroma temporal and vertical sample positions shown relative to sampling time instant
(where from left to right is shown a top field, bottom field, top field, and bottom field)

5.3 Decoder Limitations

5.3.1 Minimum and maximum sizes

For progressive frames, the frame height and frame width shall be an 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. Note that the codec works with 16x16 pixels of luminance component per MB, and thus the internal frame dimensions used in the codec are multiples of 16. The internal luma arrays used for decoding the video shall have dimensions of 16 * ((‘frame height’ + 15) / 16) rows by 16 * ((‘frame_width’ + 15) / 16) columns. The internal chroma arrays used for decoding the video shall have dimensions half as much as the luma arrays. 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 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 columns 0 to

'frame width' -1 out of the internal chroma array. The maximum dimensions of the frame are limited by the target profile and level of the bitstream as listed in Annex D.

5.3.2 Maximum size constraint on compressed bits

A valid VC-9 bitstream shall satisfy the following constraint imposed on the maximum size (in bits) of compressed data corresponding to any single row of macroblocks.

The data size corresponding to any macroblock row in VC-9 shall not exceed the greater of the two limits: (i) 6144 bits, and (ii) 1536 bits times the number of macroblocks in the horizontal direction.

Compressed data corresponding to a macroblock row is 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.

Bitplane coding, if used in a mode other than the raw mode, is assumed to be part of the header and is therefore outside of the constraint. In the raw mode, bits used in coding macroblock information such as 1/4MV 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.

6 Sequence And Entry-Point Bitstream Syntax and Semantics

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

6.1 Sequence-level Syntax and Semantics

A sequence-level header contains sequence-level parameters used to decode the sequence of compressed pictures. In simple and main profiles, this header shall be communicated to the decoder by the transport layer. In the advanced profile, this header is part of the video data bitstream and its presence is subject to the rules described in Annex G. Figure 7 shows the bitstream elements that make up the sequence layer for the simple and main profiles. Figure 8 shows the bitstream elements that make up the sequence header for the advanced profile.

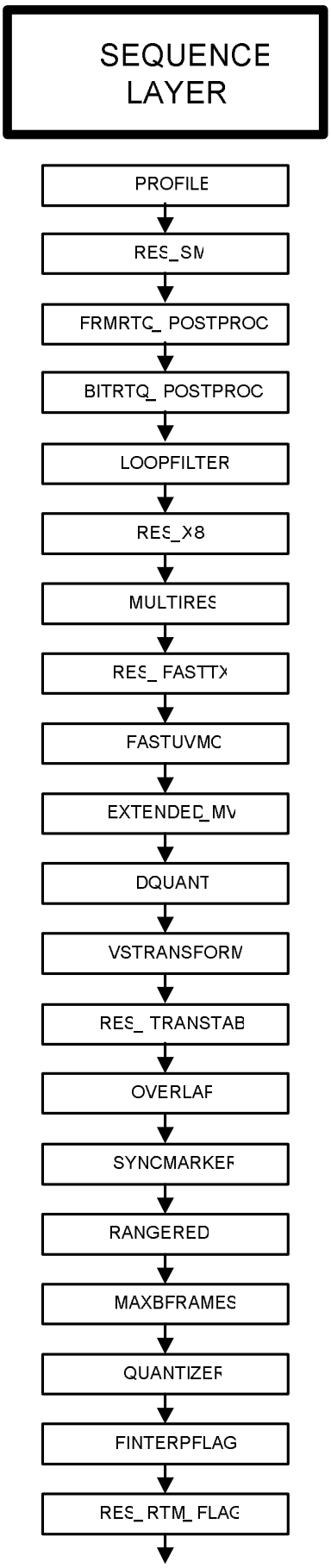

SEQUENCE LAYER
PROFILE
RES_SM
FRMRTQ_ POSTPROC
BITRTQ_ POSTPROC
LOOPFILTER
RES_X8
MULTIRES
RES_ FASTTX
FASTUVMC
EXTENDED_MV
DQUANT
VSTRANSFORM
RES_ TRANSTAB
OVERLAP
SYNCMARKER
RANGERED
MAXBFRAMES
QUANTIZER
FINTERPFLAG
RES_ RTM_ FLAG

Figure 7: Syntax diagram for the sequence layer bitstream for simple and main profiles.

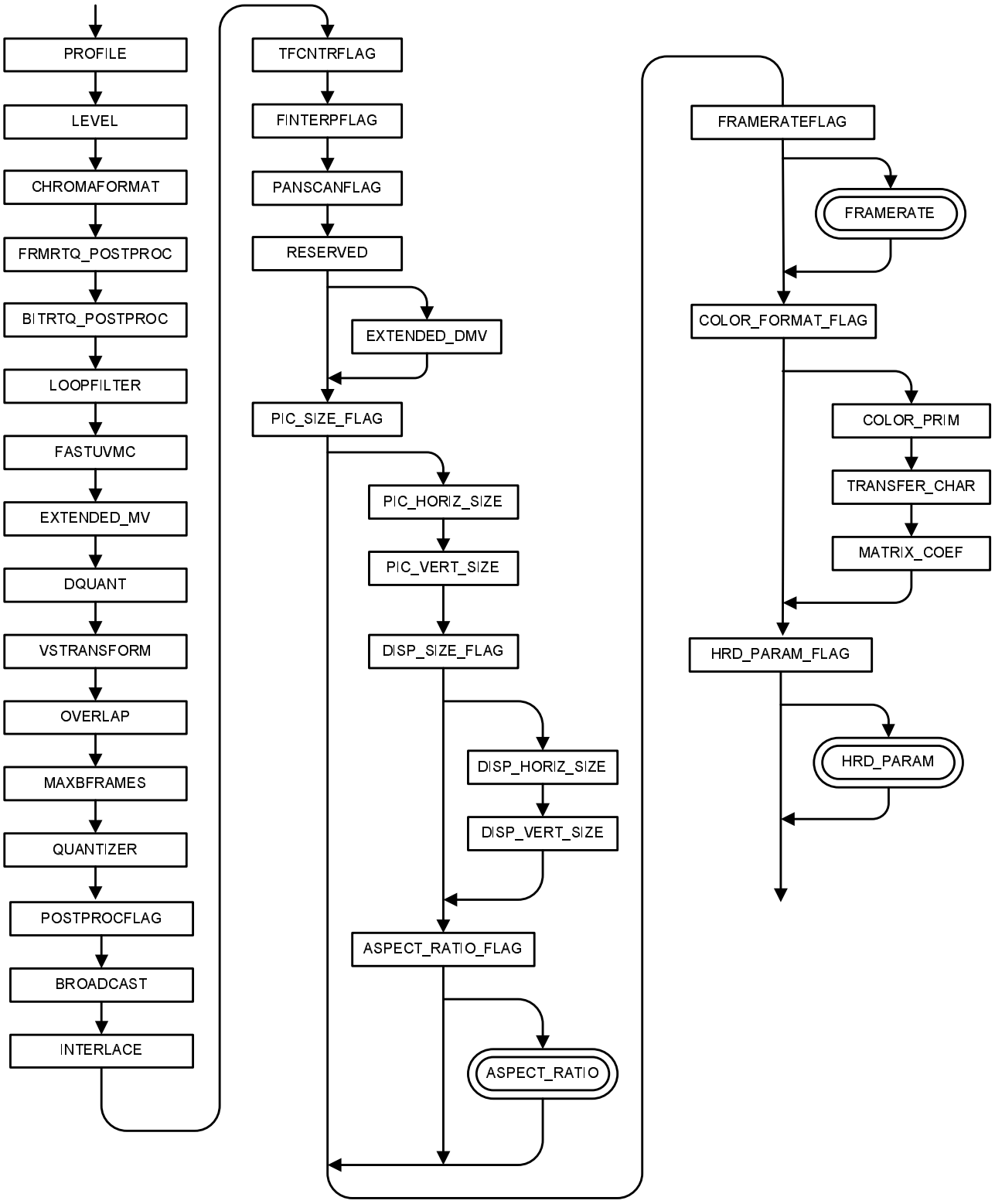


Figure 8: Syntax diagram for the sequence layer bitstream for the Advanced Profile

The following tables show the syntax elements of the sequence layer.

Table 1: Sequence layer bitstream for Simple and Main Profile

SEQUENCE LAYER( ) {	Number of bits	
PROFILE	2	
RES_SM	2	
FRMRTQ_POSTPROC	3	
BITRTQ_POSTPROC	5	
LOOPFILTER	1	
RES_X8	1	
MULTIRES	1	
RES_FASTTX	1	
FASTUVMC	1	
EXTENDED_MV	1	
DQUANT	2	
VSTRANSFORM	1	
RES_TRANSTAB	1	
OVERLAP	1	
SYNCMARKER	1	
RANGERED	1	
MAXBFRAMES	3	
QUANTIZER	2	
FINTERPFLAG	1	
RES_RTM_FLAG	1	
}		

Table 2: Sequence layer bitstream for Advanced Profile

SEQUENCE LAYER( ) {	Number of bits	
PROFILE	2	
LEVEL	3	
CHROMAFORMAT	2	
FRMRTQ_POSTPROC	3	
BITRTQ_POSTPROC	5	
LOOPFILTER	1	

FASTUVMC	1	
EXTENDED_MV	1	
DQUANT	2	
VSTRANSFORM	1	
OVERLAP	1	
MAXBFRAMES	3	
QUANTIZER	2	
POSTPROCFLAG	1	
BROADCAST	1	
INTERLACE	1	
TFCNTRFLAG	1	
FINTERPFLAG	1	
PANSCANFLAG	1	
RESERVED	1	
if (EXTENDED_MV == 1) {		
EXTENDED_DMV	1	
}		
PIC_SIZE_FLAG	1	
if (PIC_SIZE_FLAG == 1) {		
PIC_HORIZ_SIZE	12	
PIC_VERT_SIZE	12	
DISP_SIZE_FLAG	1	
if (DISP_SIZE_FLAG == 1) {		
DISP_HORIZ_SIZE	14	
DISP_VERT_SIZE	14	
}		
ASPECT_RATIO_FLAG	1	
if (ASPECT_RATIO_FLAG == 1) {		
ASPECT_RATIO	4	
if (ASPECT_RATIO == ‘15’) {		
ASPECT_HORIZ_SIZE	8	
ASPECT_VERT_SIZE	8	
}		
}		
}		

FRAMERATEFLAG	1	
if (FRAMERATEFLAG == 1) {		
FRAMERATEIND	1	
if (FRAMERATEIND == 0) {		
FRAMERATENR	8	
FRAMERATEDR	4	
} else {		
FRAMERATEEXP	16	
}		
}		
COLOR_FORMAT_FLAG	1	
if (COLOR_FORMAT_FLAG ==1) {		
COLOR_PRIM	8	
TRANSFER_CHAR	8	
MATRIX_COEF	8	
}		
HRD_PARAM_ FLAG	1	
if (HRD_PARAM_ FLAG == 1) {		
HRD_parameters( )		
}		
}		

6.1.1 Profile (PROFILE)(2 bits)

PROFILE is a 2-bit syntax element that specifies the encoding profile used to produce the sequence. The three profiles are simple, main, and advanced profile, and they correspond to PROFILE = 0, 1, and 3, respectively. The value 2 is forbidden. The simple profile is designed to ease the computation load for the codec by placing restrictions on certain compression tools. The relation of profiles to coding tools is presented in Annex D.

6.1.2 Level (LEVEL)(3 bits)

LEVEL is a 3-bit syntax element that is present only if the PROFILE takes the value corresponding to advanced profile, 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 are defined as follows:

LEVEL	Meaning
000	L0

001	L1
010	L2
011	L3
100	L4
101-111	SMPTE Reserved

The levels for Simple and Main profile shall be communicated to the decoder by the Transport Layer. See Annex D on the use of this syntax element.

6.1.3 Chroma Format (CHROMAFORMAT) (2 bits)

The CHROMAFORMAT syntax element is a 2-bit syntax element that is present only in advanced profile. It indicates the chrominance/luminance format used to represent each picture. The formats are defined as follows:

CHROMAFORMAT	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 forbidden

6.1.4 Reserved (RES_SM)(2 bits)

RES_SM is a 2-bit syntax element, that is present only in simple and main profiles, and shall be set to zero. All other values are forbidden.

6.1.5 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 for controlling the strength of post-processing operation. FRMRTQ_POSTPROC represents the value of quantized framerate from 2 frames/second to 30 frames/second in steps of 4. Note that this parameter does not affect the decoding of the bitstream in any way.

6.1.6 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 for controlling the strength of post-processing operation. BITRTQ_POSTPROC represents the value of quantized bit rate from 32 kbps to 2016 kbps in steps of 64 kbps. Note that this parameter does not affect the decoding of the bitstream in any way.

6.1.7 Picture Size Indicator Flag (PIC_SIZE_FLAG)(1 bit)

PIC_ SIZE_FLAG is a 1-bit syntax element, present only in the advanced profile, that specifies if the size of the coded picture is sent in the bitstream. If PIC_ SIZE_FLAG is 0, the size is to be communicated to the decoder by the transport layer. If PIC_SIZE_FLAG is 1, the size is indicated by the following syntax elements.

6.1.7.1 Horizontal Size of Picture (PIC_HORIZ_SIZE)(12 bits)

PIC_HORIZ_SIZE is a fixed-length syntax element that is present only in advanced profile, and only if PIC_SIZE_FLAG takes the value 1, and specifies the horizontal size of the coded picture in units of 2 pixels. This syntax element is a 12-bit binary encoding of sizes ranging from 2 to 8192 in units of 2.

6.1.7.2 Vertical Size of Picture (PIC_VERT_SIZE)(12 bits)

PIC_VERT_SIZE is a fixed-length syntax element that is present only in advanced profile, and only if PIC_SIZE_FLAG takes the value 1, and specifies the vertical size of the coded picture in units of 2 pixels. This syntax element is a 12-bit binary encoding of sizes ranging from 2 to 8192 in units of 2.

6.1.7.3 Display Size Indicator Flag (DISP_SIZE_FLAG)(1 bit)

DISP_ SIZE_FLAG is a 1-bit syntax element that is present only in advanced profile, and only if PIC_SIZE_FLAG takes the value 1, and specifies if the display size of the picture is sent in the bitstream. If DISP_ SIZE_FLAG is 0, the display size is taken to be identical to the coded picture size. If DISP_SIZE_FLAG is 1, the display size is indicated by the following syntax elements.

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

DISP_HORIZ_SIZE is a 14-bit syntax element that is present only in advanced profile, and only if DISP_SIZE_FLAG is 1 and PIC_SIZE_FLAG is 1, and specifies the horizontal display size of the picture in pixels. This syntax element is a 14-bit binary encoding of sizes ranging from 1 to 16384.

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

DISP_VERT_SIZE is a 14-bit syntax element that is present only in advanced profile, and only if DISP_SIZE_FLAG is 1 and PIC_SIZE_FLAG is 1, and specifies the vertical display size of the picture in pixels. This syntax element is a 14-bit binary encoding of sizes ranging from 1 to 16384.

6.1.7.4 Aspect Ratio Indicator Flag (ASPECT_RATIO_FLAG)(1 bit)

ASPECT_RATIO_FLAG is a 1-bit syntax element that is present only in advanced profile, and only if PIC_SIZE_FLAG takes the value 1, and specifies if the aspect ratio is sent in the bitstream. If ASPECT_RATIO_FLAG is 0, the sample aspect ratio is not transmitted. If ASPECT_RATIO_FLAG is 1, the sample aspect ratio of the sequence is transmitted in the following syntax element.

6.1.7.4.1 Aspect Ratio (ASPECT_RATIO)(4 bits)

ASPECT_RATIO is a 4-bit syntax element that is present only in advanced profile, and only if the ASPECT_RATIO_FLAG is 1 and PIC_SIZE_FLAG is 1, and it specifies the Sample Aspect Ratio (SAR) for the sequence. Note that the sample aspect ratio is often referred to as the pixel aspect ratio.

The table below specifies the value of the Sample Aspect Ratio for each value of the ASPECT_RATIO syntax element.

ASPECT_RATIO	SAR
0	Unspecified
1	1:1
2	12:11
3	10:11
4	16:11
5	40:33
6	24:11
7	20:11
8	32:11
9	80:33
10	18:11
11	15:11
12	64:33
13	160:99
14	SMPTE Reserved
15	Aspect width and height transmitted.

If ASPECT_RATIO takes the value ‘15’, the aspect width and aspect height are transmitted as the following 2 syntax elements.

6.1.7.4.2 Aspect Width (ASPECT_HORIZ_SIZE)(8 bits)

ASPECT_HORIZ_SIZE is an 8-bit syntax element that is present only in advanced profile, and only if ASPECT_RATIO_FLAG is 1 and PIC_SIZE_FLAG is 1, and specifies the horizontal aspect size of the sample. This syntax element is a binary encoding of sizes ranging from 1 to 256.

6.1.7.4.3 Aspect Height (ASPECT_VERT_SIZE)(8 bits)

ASPECT_VERT_SIZE is an 8-bit syntax element that is present only in advanced profile, and only if ASPECT_RATIO_FLAG is 1 and PIC_SIZE_FLAG is 1, and specifies the vertical aspect size of the sample. This syntax element is a binary encoding of sizes ranging from 1 to 256. The SAR is defined as the ratio of ASPECT_HORIZ_SIZE to ASPECT_VERT_SIZE.

6.1.8 Frame Rate Flag (FRAMERATEFLAG)(1 bit)

The syntax element FRAMERATEFLAG is a 1-bit syntax element that is present only in advanced profile, and indicates that frame rate information is present. If FRAMERATEFLAG = 0, no frame rate information is present. In this case, the receiver may rely on the underlying protocol (such as Program Clock References in MPEG-2 transport) to estimate the frame rate. If FRAMERATEFLAG = 1, frame rate information may be obtained from subsequent syntax elements.

If the video sequence is signaled as progressive (either implicitly as when 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.

6.1.8.1 Frame Rate Indicator (FRAMERATEIND)(1 bit)

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

6.1.8.2 Frame Rate Numerator (FRAMERATENR)(8bits)

The syntax element FRAMERATENR is an 8-bit syntax element that is present only in advanced profile, and only if FRAMERATEIND = 0 and FRAMERATEFLAG = 1, and it indicates the frame rate numerator of the encoded video sequence. The following table gives the meaning of the 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
6-255	SMPTE Reserved

6.1.8.3 Frame Rate Denominator (FRAMERATEDR)(4 bits)

The syntax element FRAMERATEDR is a 4-bit syntax element that is present only in advanced profile, and only if FRAMERATEIND = 0 and FRAMERATEFLAG = 1, and it indicates the frame rate denominator of the encoded video sequence. The following table gives the meaning of the FRAMERATEDR syntax element. The frame rate of the sequence is the ratio of the Frame rate Numerator to the Frame rate Denominator.

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

6.1.8.4 Frame Rate Explicit (FRAMERATEEXP)(16bits)

The syntax element FRAMERATEEXP is a 16-bit syntax element that is present only in advanced profile, and only if FRAMERATEIND = 1 and FRAMERATEFLAG = 1. FRAMERATEEXP explicitly indicates the frame rate of the encoded video sequence. This element is used signal frame rate ranging from 0.03125 Hz to 2048 Hz in uniform steps of 0.03125 Hz.

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

COLOR_FORMAT_FLAG is a 1-bit syntax element that is present only in advanced profile, and indicates if color format information is present. If COLOR_FORMAT_FLAG is 1, color format information, such as Color Primaries, Transfer Characteristics, and Matrix Coefficients, may be obtained from subsequent syntax elements. COLOR_FORMAT_FLAG is 0, no color format information is present in the bitstream, and these syntax elements are set to the default values specified below.

6.1.9.1 Color Primaries (COLOR_PRIM)(8 bits)

COLOR_PRIM is an 8-bit syntax element that is present only in advanced profile, and only if COLOR_FORMAT_FLAG is 1, and describes the chromaticity coordinates of the color primaries. The table below defines the syntax element values indicating the technical specifications where the chromaticity coordinates are specified. The default value is Recommendation ITU-R BT. 709.

COLOR_PRIM	Color Primaries Specification
0	Forbidden
1	Recommendation ITU-R BT.709-2, SMPTE 274M-1995, and SMPTE296M-1997 primary x y green 0.300 0.600 blue 0.150 0.060 red 0.640 0.330 white D65 0.3127 0.3290
2	Unspecified color primaries
3	SMPTE Reserved
4	Recommendation ITU-R BT.470-2 System M primary x y green 0.21 0.71 blue 0.14 0.08 red 0.67 0.33 white C 0.310 0.316
5	ITU-R Recommendation BT.470-2 System B, G; EBU Tech. 3213 (1981) primary x y green 0.29 0.60 blue 0.15 0.06 red 0.64 0.33 white D65 0.3127 0.3290
6	SMPTE C Primaries from SMPTE RP145-1993. SMPTE 293M-1996, SMPTE 240M-1995, and SMPTE 170M-1994, 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

6.1.9.2 Transfer Characteristics (TRANSFER_CHAR)(8 bits)

TRANSFER_CHAR is an 8-bit syntax element that is present only in advanced profile, and only if COLOR_FORMAT_FLAG is 1, and describes the opto-electronic transfer characteristics of the source picture. The table below defines the syntax element values indicating the technical specification where the transfer characteristics are specified. The default value is Recommendation ITU-R BT.709.

TRANSFER_CHAR	Transfer Characteristics Specification
0	Forbidden
1	Recommendation ITU-R BT.709, SMPTE 274M-1995, SMPTE 296M-1997, SMPTE 293M-1996 and SMPTE 170M-1994 (Default) $V = 1.099\ L_C^{0.45} - 0.099$ for $1 >= L_C >= 0.018$ $V = 4.500\ L_C$ for $0.018 > L_C >= 0$
2	Unspecified Transfer Characteristics
3	SMPTE Reserved
4	Recommendation ITU-R BT.470-2 System M Assumed display gamma 2.2
5	Recommendation ITU-R BT.470-2 System B,G Assumed display gamma 2.8
6	SMPTE 170M $V = 1.099\ L_C^{0.45} - 0.099$ for $1 >= L_C >= 0.018$ $V = 4.500\ L_C$ for $0.018 > L_C >= 0$
7	SMPTE 240M-1995 $V = 1.1115\ L_C^{0.45} - 0.1115$ for $L_C >= 0.0228$ $V = 4.0\ L_C$ for $0.0228 > L_C$
8	Linear Transfer Characteristrics
9-255	SMPTE Reserved

6.1.9.3 Matrix Coefficients (MATRIX_COEF)(8 bits)

MATRIX_COEF is an 8-bit syntax element that is present only in advanced profile, and only if COLOR_FORMAT_FLAG is 1, and describes the matrix coefficients used to derive Y, Cb and Cr signals from the color primaries. The table below defines the syntax element values indicating the technical specification where the matrices are specified. The default value is Recommendation ITU-R BT. 709.

MATRIX_COEF	Matrix Coefficients Specification
0	Forbidden
1	Recommendation ITU-R BT.709-2 (1125/60/2:1 only), SMPTE 274M-1995 and SMPTE 296M-1997. $K_R = 0.2126$; $K_B = 0.0722$
2	Unspecified Matrix
3-5	SMPTE Reserved
6	Recommendation ITU-R BT.601-4, Recommendation ITU-R BT.470-4 System B and G, SMPTE 170M-1994 and SMPTE 293M-1996. $K_R = 0.299$; $K_B = 0.114$
7	SMPTE 240M-1995 $K_R = 0.212$; $K_B = 0.087$
8-255	SMPTE Reserved

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

The HRD_PARAM_FLAG is a 1-bit flag that is present only in advanced profile, and indicates the presence of HRD parameters in the bitstream. If this flag is 0, HRD parameters are not present. If HRD_PARAM_FLAG is 1, syntax elements of the HRD are present as detailed next.

6.1.10.1 Hypothetical Reference Decoder (HRD)(Variable size)

The HRD syntax elements are present only in advanced profile, and only if HRD_PARAM_FLAG is 1, and are as follows. See Annex C for additional details on the semantics and use of HRD parameters.

hrd_parameters( )	**Descriptor**	**Range**
{		
HRD_NUM_LEAKY_BUCKETS	FLC-5	(1, 32)
BIT_RATE_EXPONENT	FLC-4	(6,21)
BUFFER_SIZE_EXPONENT	FLC-4	(4,19)
for(n=1; n <= HRD_NUM_LEAKY_BUCKETS; n++)		
{		
HRD_RATE[n **]**	FLC-16	$(1,2^{16})$
HRD_BUFFER[n **]**	FLC-16	$(1,2^{16})$
}		

HRD_NUM_LEAKY_BUCKETS – A number between 1 and 32 that specifies the number of leaky buckets *N*. The value of *N*-1 is encoded as a fixed length code in binary using 5 bits.

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 is 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 is encoded in the syntax element BIT_RATE_EXPONENT in fixed length using 4 bits , and takes the range from 6 to 21.

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 is 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 is encoded in the syntax element BUFFER_SIZE_EXPONENT using a fixed length of 4 bits, and takes the range from 4 to 19.

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

6.1.11 Loop Filter (LOOPFILTER)(1 bit)

LOOPFILTER is a 1-bit syntax element that indicates whether loop filtering is enabled for the sequence. If LOOPFILTER = 0, then loop filtering is not enabled. If LOOPFILTER = 1, then loop filtering is 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.

6.1.12 Reserved Coding (RES_X8)(1 bit)

RES_X8 is a 1-bit syntax element that is present only in simple and main profiles, and shall be set to zero. The value 1 is forbidden.

6.1.13 Multiresolution Coding (MULTIRES)(1 bit)

MULTIRES is a 1-bit syntax element that is present only in simple and main profiles, and indicates whether the frames may be coded at smaller resolutions than the specified frame resolution. Resolution changes are allowed only on I pictures. If MULTIRES = 1, then the frame level RESPIC syntax element is present which indicates the resolution for that frame. See sections 8.1.1.3 for a description of multiresolution decoding in I pictures.

6.1.14 Reserved (RES_FASTTX)(1 bit)

RES_FASTTX is a 1-bit syntax element that is present only in simple and main profiles, and shall be set to the value 1. The value 0 is forbidden.

6.1.15 FAST UV Motion Comp (FASTUVMC)(1 bit)

FASTUVMC is a 1-bit syntax element that controls the subpixel interpolation and rounding of chroma motion vectors. If FASTUVMC = 1, then the chroma motion vectors that are at quarter pel offsets will be rounded to the nearest half or full pel positions If FASTUVMC = 0, then no special rounding or filtering is done for chroma. See section 8.3.5.4.2 for details on how chroma motion vector computation is performed for the two cases. (Informative – The purpose of this mode is speed optimization of the decoder).

FASTUVMC is always 1 for the Simple Profile.

6.1.16 Extended Motion Vectors (EXTENDED_MV)(1 bit)

EXTENDED_MV is a 1-bit syntax element that indicates whether extended motion vectors is turned on (value 1) or off (value 0). This bit is always set to zero for the Simple Profile. For the Main and Advanced Profiles, the extended motion vector mode indicates the possibility of extended motion vectors in P and B pictures.

6.1.17 Extended Differential Motion Vector Range (EXTENDED_DMV)(1 bit)

EXTENDED_DMV is a 1-bit syntax element that is present in Advanced Profile sequence headers if EXTENDED_MV = 1. This bit indicates whether extended differential motion vector range is signaled at the picture layer for P and B pictures.

6.1.18 Macroblock Quantization (DQUANT)(2 bit)

DQUANT is a 2-bit syntax element that indicates whether or not the quantization step size may vary within a frame. There are three possible values for DQUANT. If DQUANT = 0, then only one quantization step size (i.e. the frame quantization step size) may be used per frame. If DQUANT = 1 or 2, then the quantization step size may vary within the frame. In simple profile, DQUANT = 0. See section 7.1.1.29 for a description of DQUANT.

6.1.19 Variable Sized Transform (VSTRANSFORM)(1 bit)

VSTRANSFORM is a 1-bit syntax element that indicates whether variable-sized transform coding is enabled for the sequence. If VSTRANSFORM = 0, then variable-sized transform coding is not enabled. If VSTRANSFORM = 1, then variable-sized transform coding is enabled. See section 8.3.6.2 for a description of variable-sized transform coding.

6.1.20 Reserved (RES_TRANSTAB)(1 bit)

RES_TRANSTAB is a 1-bit syntax element that is present only in simple and main profiles, and shall be set to 0. The value 1 is forbidden.

6.1.21 Overlapped Transform Flag (OVERLAP) (1 bit)

OVERLAP is a 1-bit flag that indicates whether Overlapped Transforms (Section 7.4) are used. If OVERLAP = 1, then Overlapped Transforms are used, otherwise they are not used.

6.1.22 Syncmarker Flag (SYNCMARKER) (1 bit)

SYNCMARKER is a 1-bit flag that is present only in simple and main profiles, and indicates whether synchronization markers may be present in the bitstream. If SYNCMARKER =1, then the markers may be present, otherwise they are not present. See section 8.8 for description of synchronization markers.

6.1.23 Range Reduction Flag (RANGERED) (1 bit)

RANGERED is a 1-bit syntax element that is present only in simple and main profiles. RANGERED is always set to zero in simple profile. In main profile, RANGERED indicates whether range reduction is used for each frame. If RANGERED = 1, then there is 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 is absent, and range reduction is not used.

6.1.24 Maximum Number of consecutive B frames (MAXBFRAMES) (3 bits)

MAXBFRAMES is a 3-bit syntax element that indicates the maximum number of consecutive B frames between I or P frames. If MAXBFRAMES = 0, then there are no B frames in the sequence. If MAXBFRAMES is not equal to zero, B Frames are present in the sequence. However, the actual non-zero value is only of informative value, and is not used in the decoding process.

6.1.25 Quantizer Specifier (QUANTIZER) (2 bits)

QUANTIZER is a 2-bit syntax element that indicates the quantizer used for the sequence. The quantizer types are encoded according to Table 3.

Table 3: Quantizer specification

FLC	Quantizer specification
00	Quantizer implicitly specified at frame level
01	Quantizer explicitly specified at frame level
10	Nonuniform quantizer used for all frames
11	Uniform quantizer used for all frames

6.1.26 Postprocessing Flag (POSTPROCFLAG) (1 bit)

POSTPROCFLAG is a 1-bit syntax element that is present only in advanced profile, and indicates at the sequence level whether frame based post processing is used.

6.1.27 Broadcast Flag (BROADCAST) (1 bit)

BROADCAST is a 1-bit syntax element that is present only in advanced profile, and indicates if the interlace syntax element flags TFF and RFF are present in advanced profile picture headers.

6.1.28 Interlace Content (INTERLACE) (1 bit)

INTERLACE is a 1-bit syntax element that is present only in the advanced profile. INTERLACE = 0 signals that the source content is progressive. INTERLACE = 1 signals that the source content is interlaced. The individual frames may still be coded using the progressive or interlace syntax when INTERLACE = 1. This bit controls the presence of the TFF, RFF and RPTFRM syntax elements in the picture headers.

6.1.29 Frame Counter Flag (TFCNTRFLAG) (1 bit)

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

6.1.30 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 = 0 then INTERPFRM is not present in picture headers. If FINTERPFLAG = 1 INTERPFRM is present in picture headers.

6.1.31 Pan Scan Flag (PANSCANFLAG)(1 bit)

PANSCANFLAG is a 1-bit syntax element that is only present in the advanced profile, and indicates if the syntax elements NUMPANSCANWIN, TOPLEFTX, TOPLEFTY, BOTRIGHTX and BOTRIGHTY are present in the bit stream. If PANSCANFLAG = 0 then above elements are not present.

6.1.32 Reserved RTM Flag (RES_RTM_FLAG)(1 bit)

RES_RTM_FLAG is a 1-bit syntax element that is present only in simple and main profiles, and shall be set to 1. The value 0 is forbidden.

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

RESERVED is a 1-bit syntax element that is present only in advanced profile, and shall be set to 1. The value 0 is forbidden.

6.2 Entry-Point Header Syntax and Semantics

An entry-point header is present only in advanced profile, and it signals a random access point for the bitstream. An entry-point guarantees that subsequent pictures may be decoded if decoding begins from at this point. An entry-point header contains syntax elements specifying the buffer fullness of the HRD leaky bucket. The syntax elements that make up the entry-point layer are shown in Figure 9, and Table 4. The use of entry-point header is discussed in Annex G.

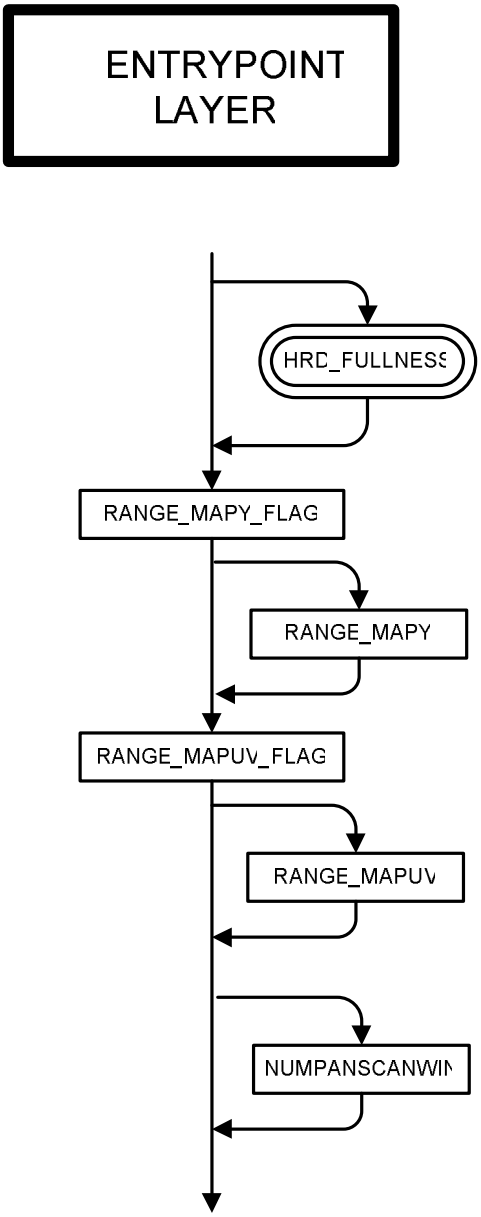


Figure 9: Syntax diagram for the entry-point layer bitstream for the Advanced Profile

Table 4: Entry-point layer bitstream for Advanced Profile

SEQUENCE LAYER( ) {	Number of bits	
if (HRD_PARAM_ FLAG == 1) {		
hrd_fullness ( )		

}		
RANGE_MAPY_FLAG	1	
if (RANGE_MAPY_FLAG == 1) {		
RANGE_MAPY	3	
}		
RANGE_MAPUV_FLAG	1	
if (RANGE_MAPUV_FLAG == 1) {		
RANGE_MAPUV	3	
}		
if (PANSCANFLAG == 1) {		
NUMPANSCANWIN	3	
}		
}		

6.2.1 HRD Buffer Fullness (HRD_FULLNESS)(Variable Size)

HRD_FULLNESS is a variable size syntax element that is present in the entry-point header only if the HRD_PARAM_FLAG in the sequence header is set to to 1. If the HRD_PARAM_FLAG in the sequence header is set to zero, HRD_FULLNESS syntax element is not present. See Annex C for additional details on the semantics and use of HRD parameters.

hrd_fullness( )	**Descriptor**	**Range**
{		
for(n=1; n <= HRD_NUM_LEAKY_BUCKETS; n++)		
{		
HRD_FULLNESS[n **]**	FLC - 8	(0, 255)
}		

HRD_FULLNESS**[**n**]** – This syntax element defines the decoder buffer fullness as an upwards rounded fraction of the buffer size B_n, in units of $B_n/256$. This element may take values in the range 1 to 256 and is encoded in binary using the 8 bit values 0 through 255 to uniformly cover the range. See section 6.1.10.1 for more information on HRD_NUM_LEAKY_BUCKETS and buffer size B_n.

6.2.2 Range Mapping Luminance Flag (RANGE_MAPY_FLAG)(1 bit)

RANGE_MAPY_FLAG is a 1-bit flag. If RANGE_MAPY_FLAG is set to one, the syntax element RANGE_MAPY is present. Otherwise, that syntax element is absent.

6.2.2.1 Range Mapping Luminance (RANGE_MAPY)(3 bits)

RANGE_MAPY is an 3-bit syntax element that is present only if RANGE_MAPY_FLAG = 1, and takes the value from 0 to 7. If this syntax element is present, the luminance component of the decoded picture is scaled according to the formula:

Y[n] = CLIP (((Y[n] – 128) * (RANGE_MAPY + 9) + 4) >> 3) + 128);

Note that this scaling is performed after all other decoding stages (including loop-filter) have been performed. Note that RANGE_MAPY_FLAG is reset to zero at the beginning of the next sequence.

6.2.3 Range Mapping Chrominance Flag (RANGE_MAPUV_FLAG)(1 bit)

RANGE_MAPUV_FLAG is a 1-bit flag. If RANGE_MAPUV_FLAG is set to one, the syntax element RANGE_MAPUV is present. Otherwise, that syntax element is absent.

6.2.3.1 Range Mapping Chrominance (RANGE_MAPUV)(3 bits)

RANGE_MAPUV is an 3-bit syntax element that is present only if RANGE_MAPUV_FLAG = 1, and takes the value from 0 to 7. If this syntax element is present, the chrominance component of the decoded picture is scaled according to the formula:

U[n] = CLIP (((U[n] – 128) * (RANGE_MAPUV + 9) + 4) >> 3) + 128);

V[n] = CLIP (((V[n] – 128) * (RANGE_MAPUV + 9) + 4) >> 3) + 128);

Note that this scaling is performed after all other decoding stages (including loop-filter) have been performed. Note that RANGE_MAPUV_FLAG is reset to zero at the beginning of the next sequence.

6.2.4 Number of pan scan windows (NUMPANSCANWIN)(3 bits)

If PANSCANFLAG = 1, then NUMPANSCANWIN gives the number of pan scan windows being used in the following pictures.

7 Progressive Bitstream Syntax and Semantics

7.1 Picture-level Syntax and Semantics

This section describes 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. Note that the slice layer is present only in the Advanced Profile bitstream. In the advanced profile, pictures and slices are always byte-aligned, and are transmitted in an independent decodable unit (IDU) as described in Annex E. In the advanced profile, a new picture, or a slice, is detected via start-codes as outlined in Annex E. In the main and simple profiles, a new picture is byte-aligned. The pointer to the coded bitstream, and its size for each coded picture 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 10 through Figure 25 show the bitstream elements that make up each layer.

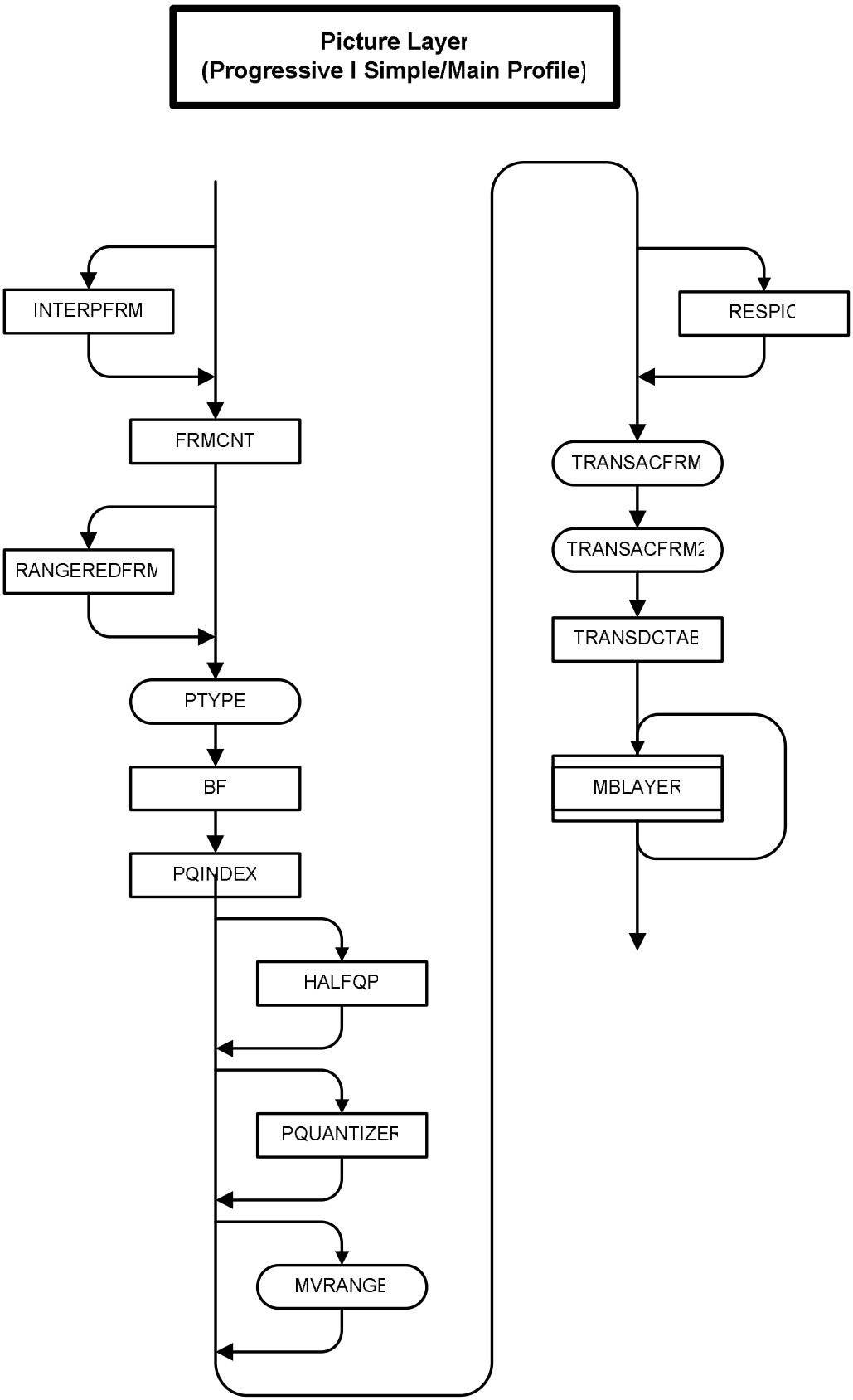


Figure 10: Syntax diagram for the Progressive I picture layer bitstream in simple/main profile

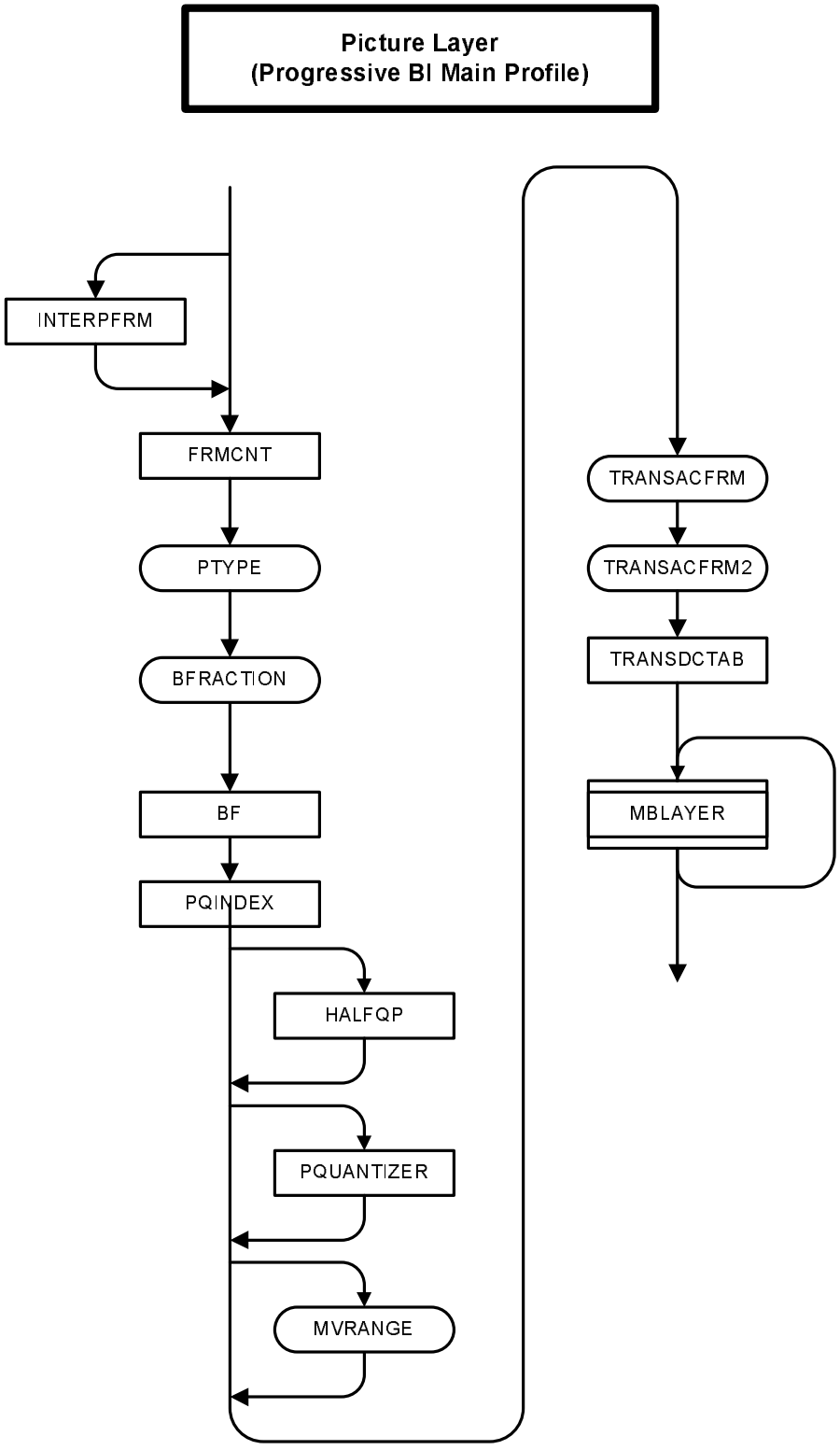


Figure 11: Syntax diagram for the Progressive BI picture layer bitstream in main profile

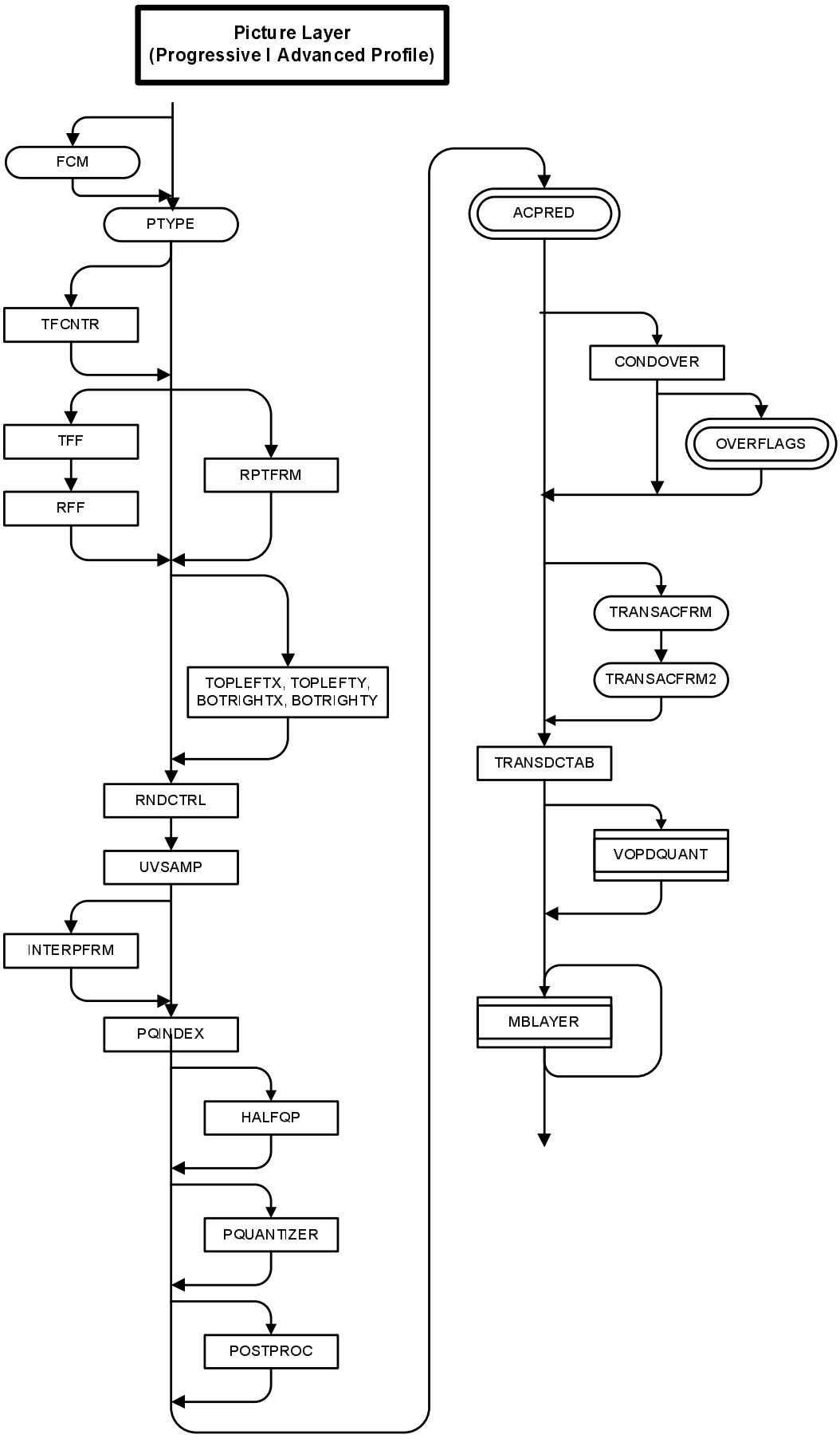

Picture Layer
(Progressive I Advanced Profile)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY,
BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
INTERPFRM
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
ACPRED
CONDOVER
OVERFLAGS
TRANSACFRM
TRANSACFRM2
TRANSDCTAB
VOPDQUANT
MBLAYER

Figure 12: Syntax diagram for the Progressive I picture layer bitstream in advanced profile.

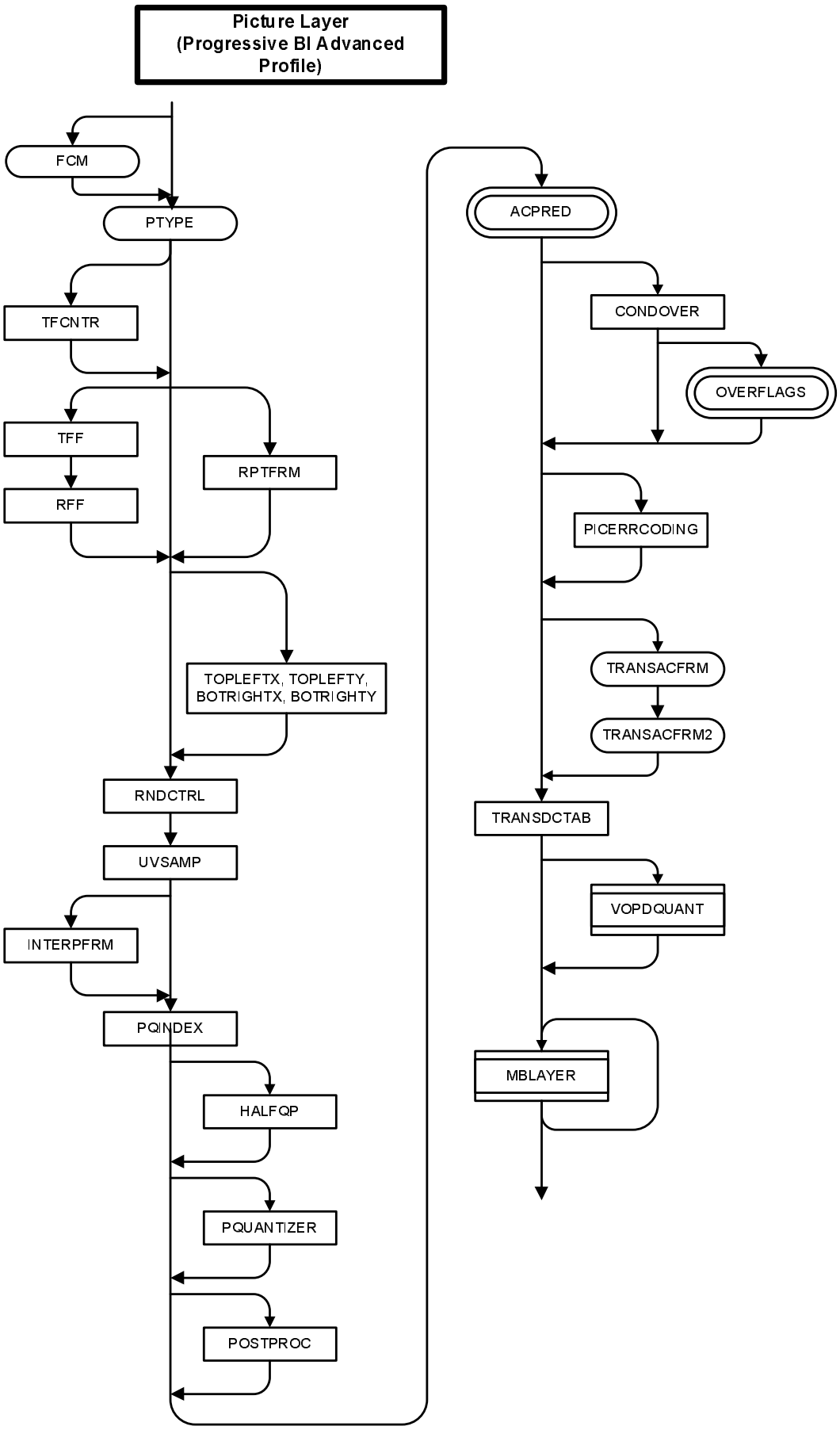

Picture Layer
(Progressive BI Advanced Profile)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY, BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
INTERPFRM
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
ACPRED
CONDOVER
OVERFLAGS
PICERRCODING
TRANSACFRM
TRANSACFRM2
TRANSDCTAB
VOPDQUANT
MBLAYER

Figure 13: Syntax diagram for the Progressive BI picture layer bitstream in advanced profile.

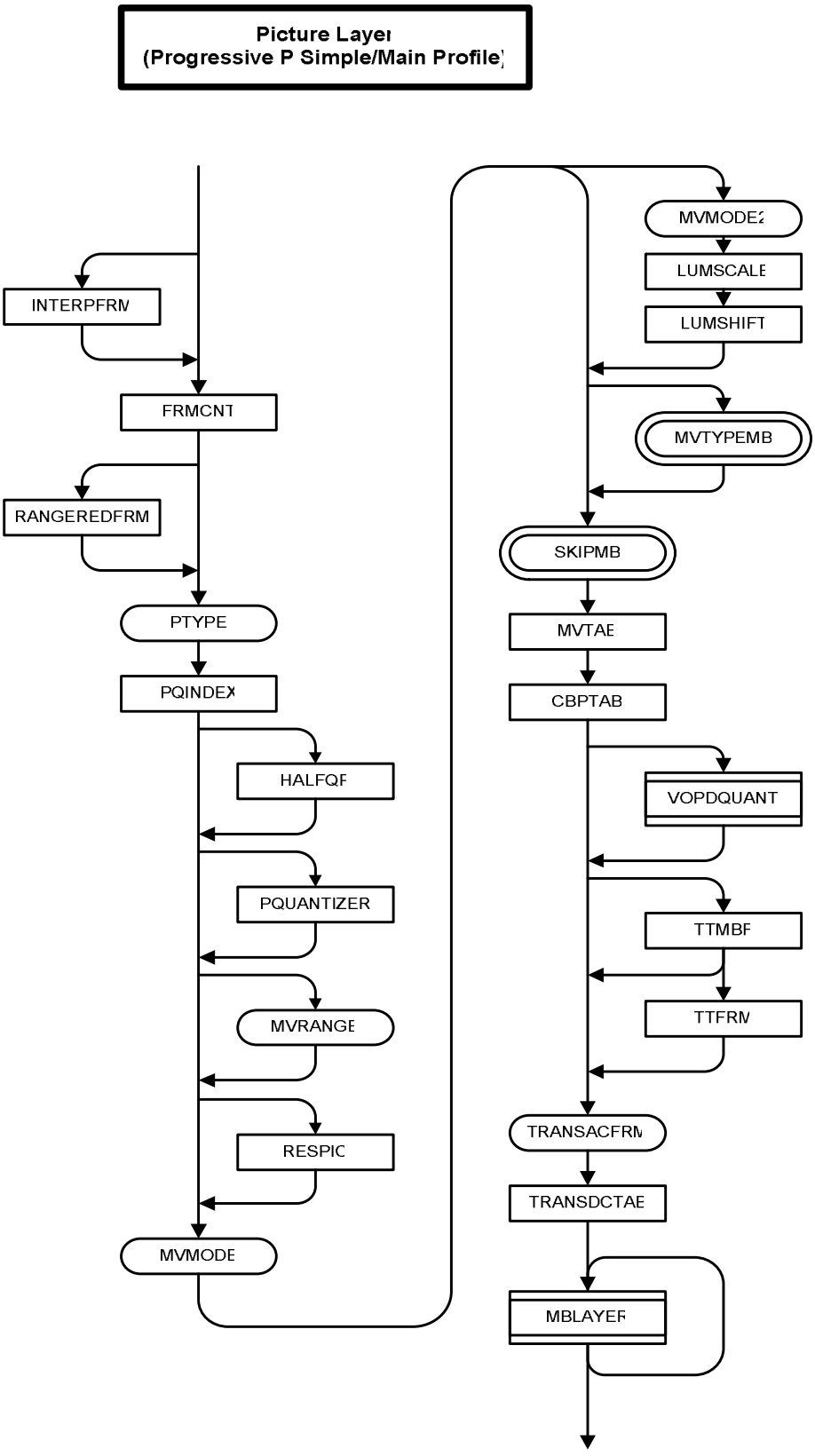


Figure 14: Syntax diagram for the Progressive P picture layer bitstream in Simple/Main Profile.

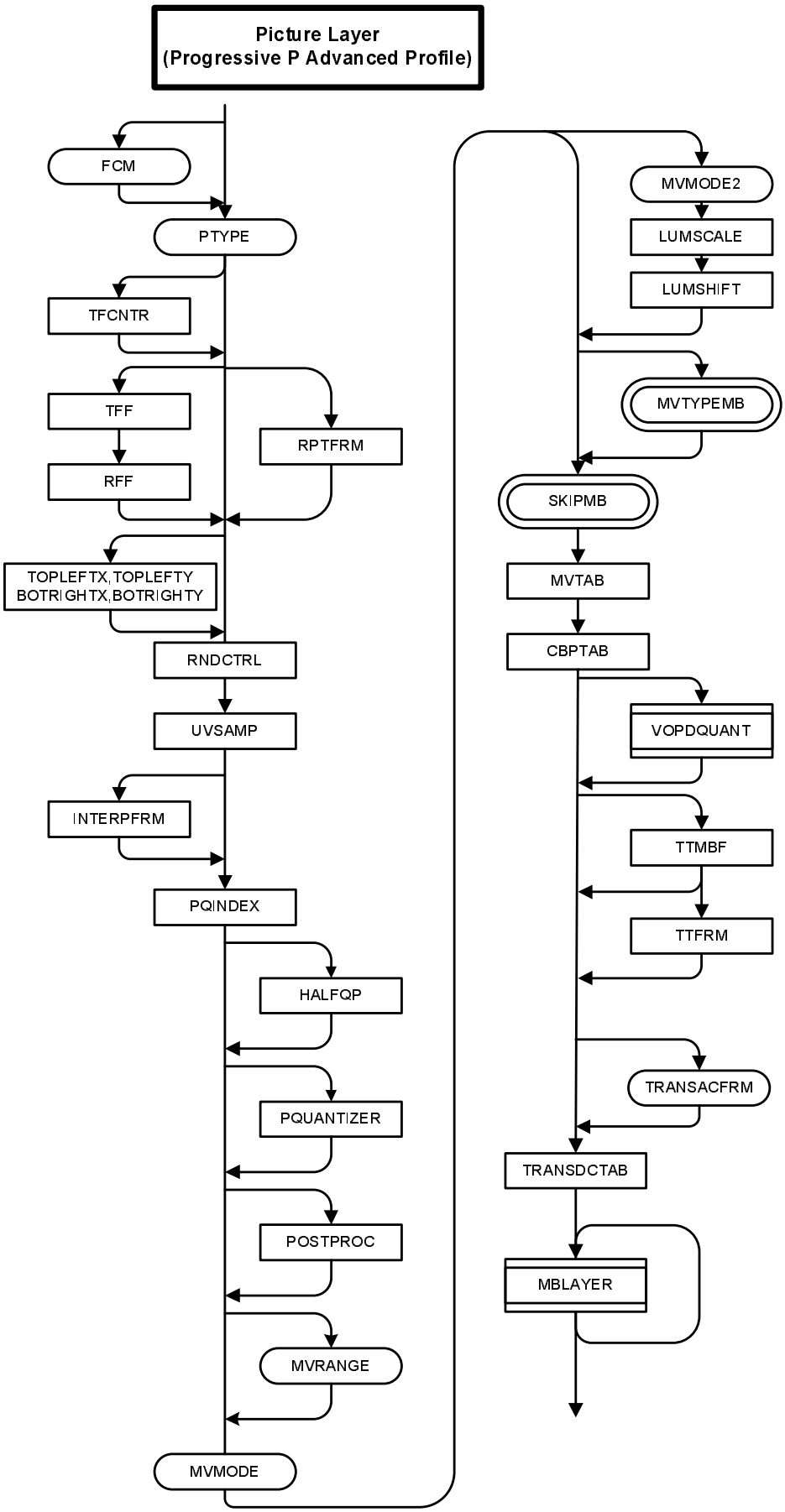

Picture Layer
(Progressive P Advanced Profile)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY
BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
INTERPFRM
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
MVRANGE
MVMODE
MVMODE2
LUMSCALE
LUMSHIFT
MVTYPEMB
SKIPMB
MVTAB
CBPTAB
VOPDQUANT
TTMBF
TTFRM
TRANSACFRM
TRANSDCTAB
MBLAYER

Figure 15: Syntax diagram for the Progressive P picture layer bitstream in Advanced Profile.

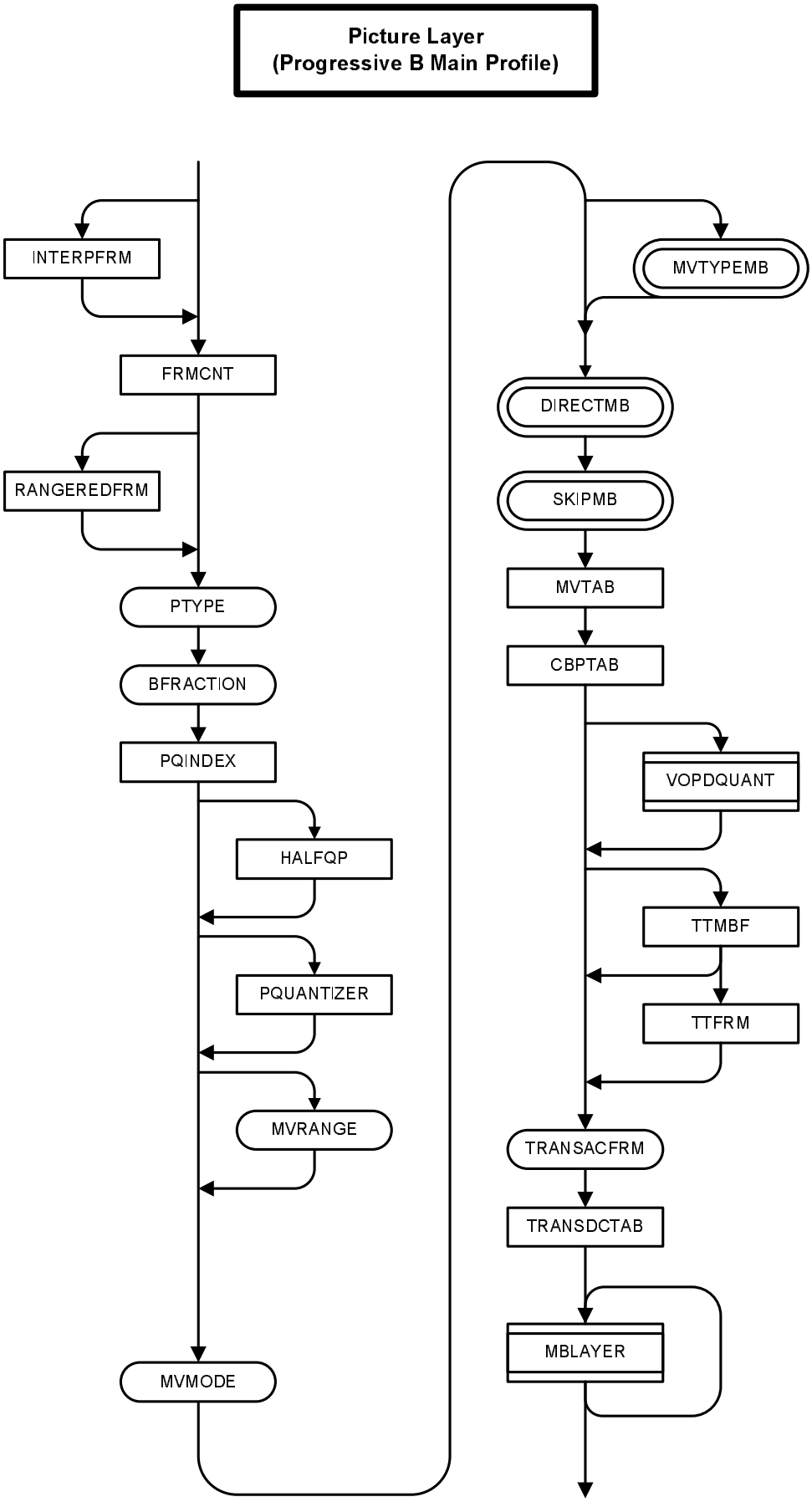

Picture Layer
(Progressive B Main Profile)
INTERPFRM
FRMCNT
RANGEREDFRM
PTYPE
BFRACTION
PQINDEX
HALFQP
PQUANTIZER
MVRANGE
MVMODE
MVTYPEMB
DIRECTMB
SKIPMB
MVTAB
CBPTAB
VOPDQUANT
TTMBF
TTFRM
TRANSACFRM
TRANSDCTAB
MBLAYER

Figure 16: Syntax diagram for the Progressive B picture layer bitstream in Main Profile.

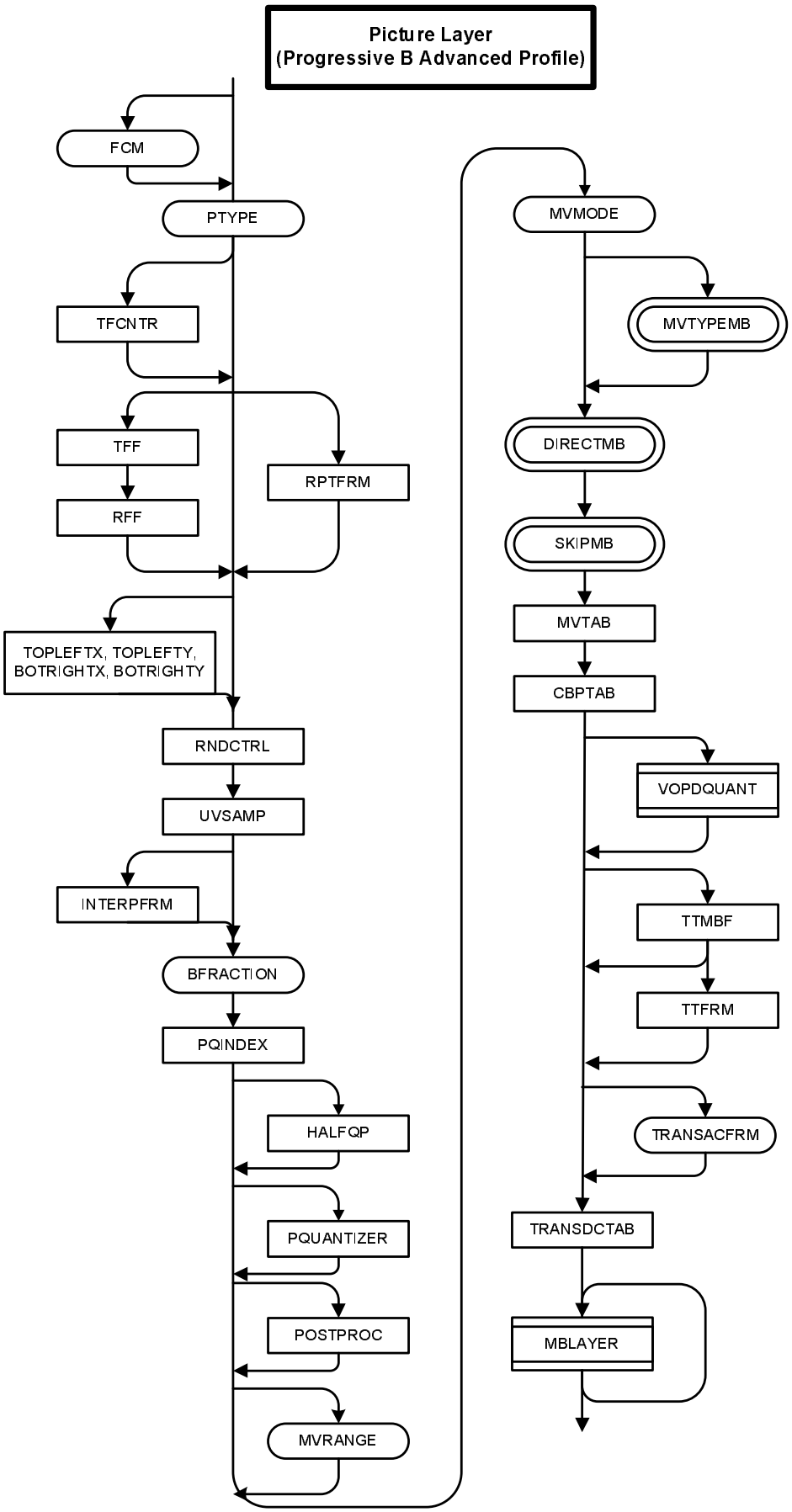

Picture Layer
(Progressive B Advanced Profile)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY,
BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
INTERPFRM
BFRACTION
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
MVRANGE
MVMODE
MVTYPEMB
DIRECTMB
SKIPMB
MVTAB
CBPTAB
VOPDQUANT
TTMBF
TTFRM
TRANSACFRM
TRANSDCTAB
MBLAYER

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

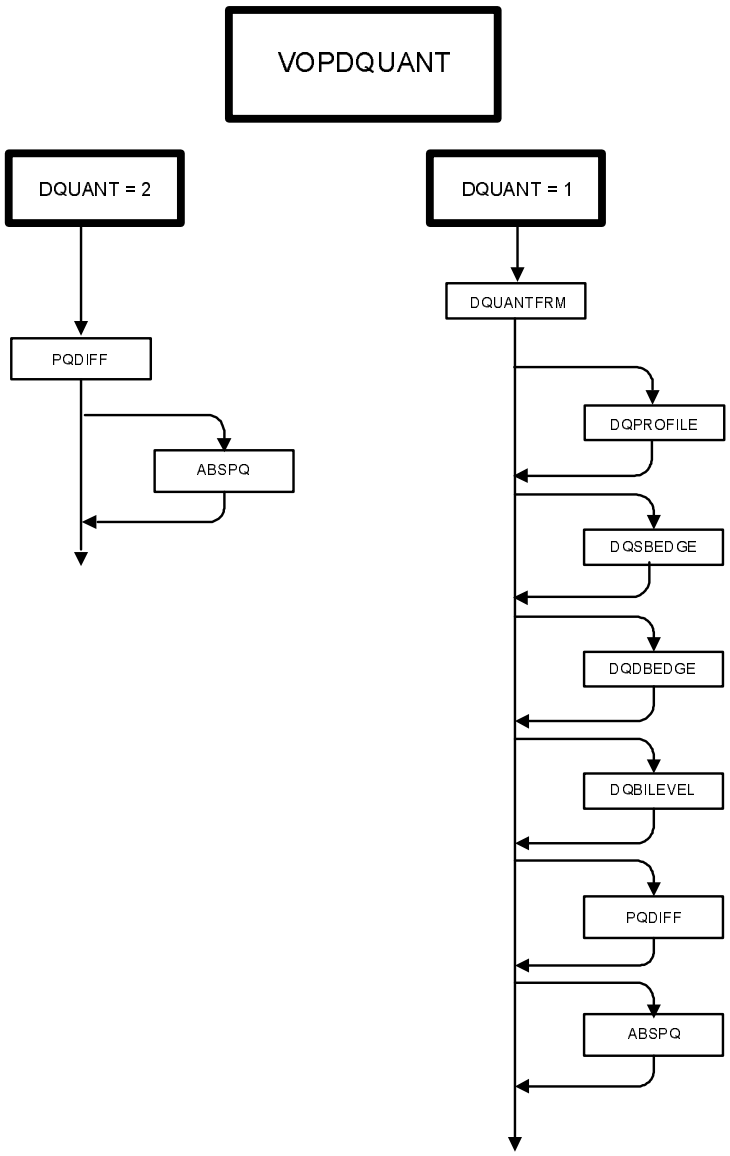


Figure 18: Syntax diagram for VOPDQUANT in picture header

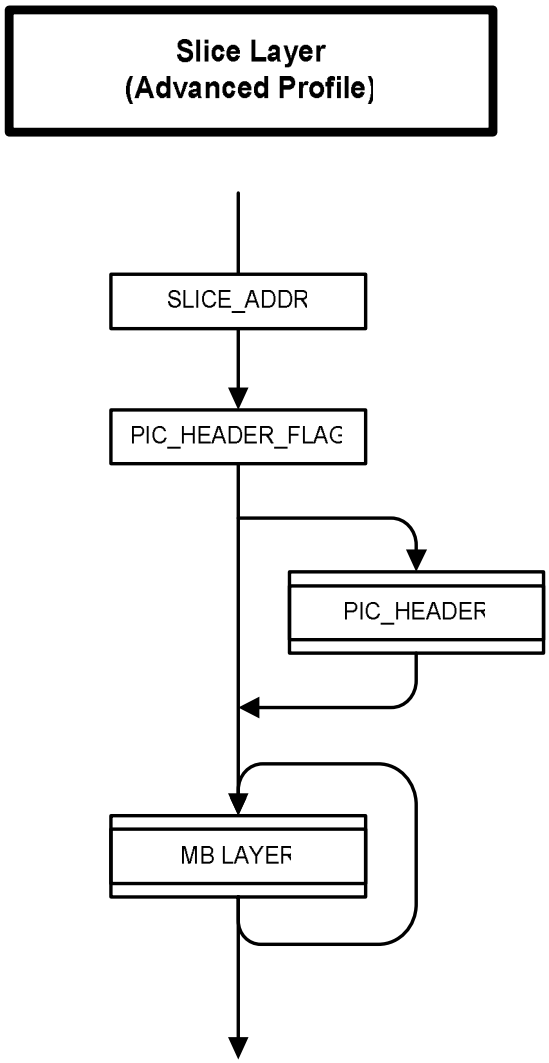


Figure 19: Syntax diagram for for the Slice-Layer bitstream in the Advanced Profile

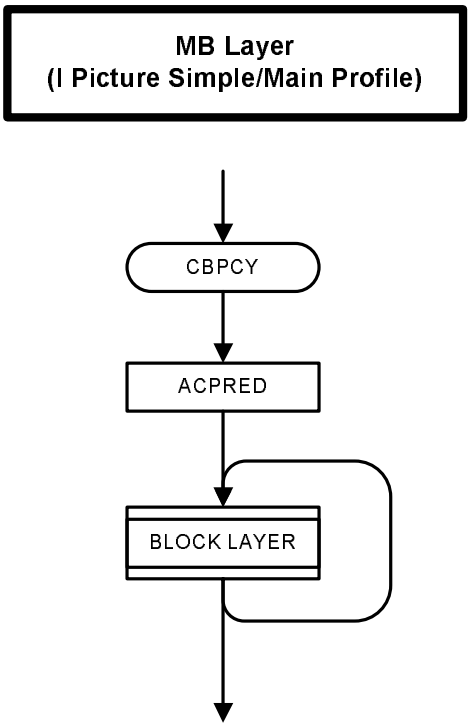


Figure 20: Syntax diagram for macroblock layer bitstream in Progressive I picture for simple/main profile

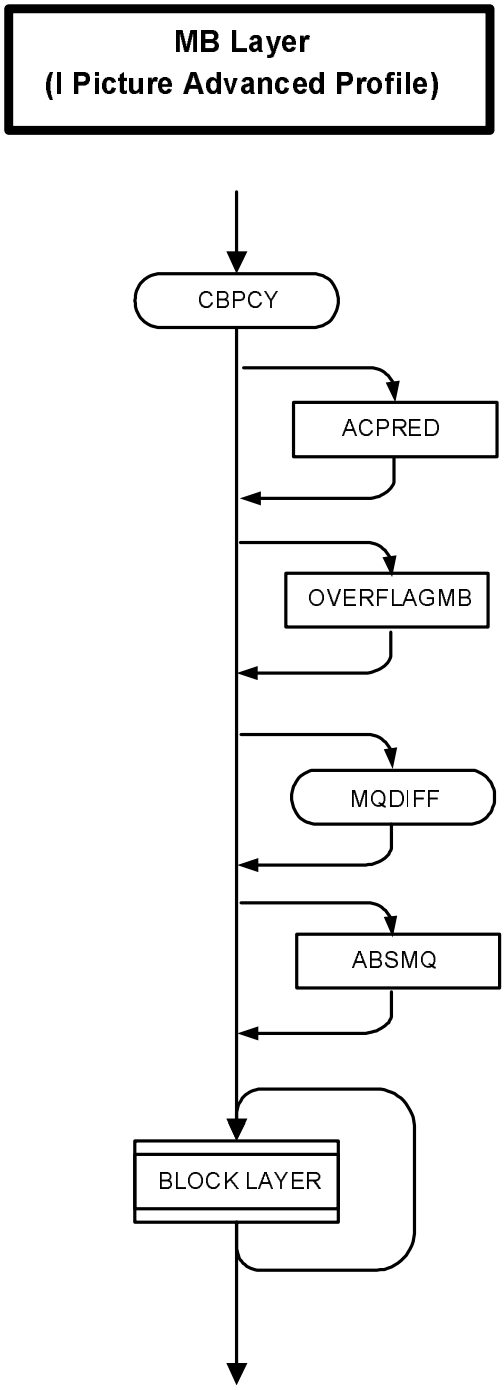


Figure 21: Syntax diagram for macroblock layer bitstream in progressive I picture for advanced profile

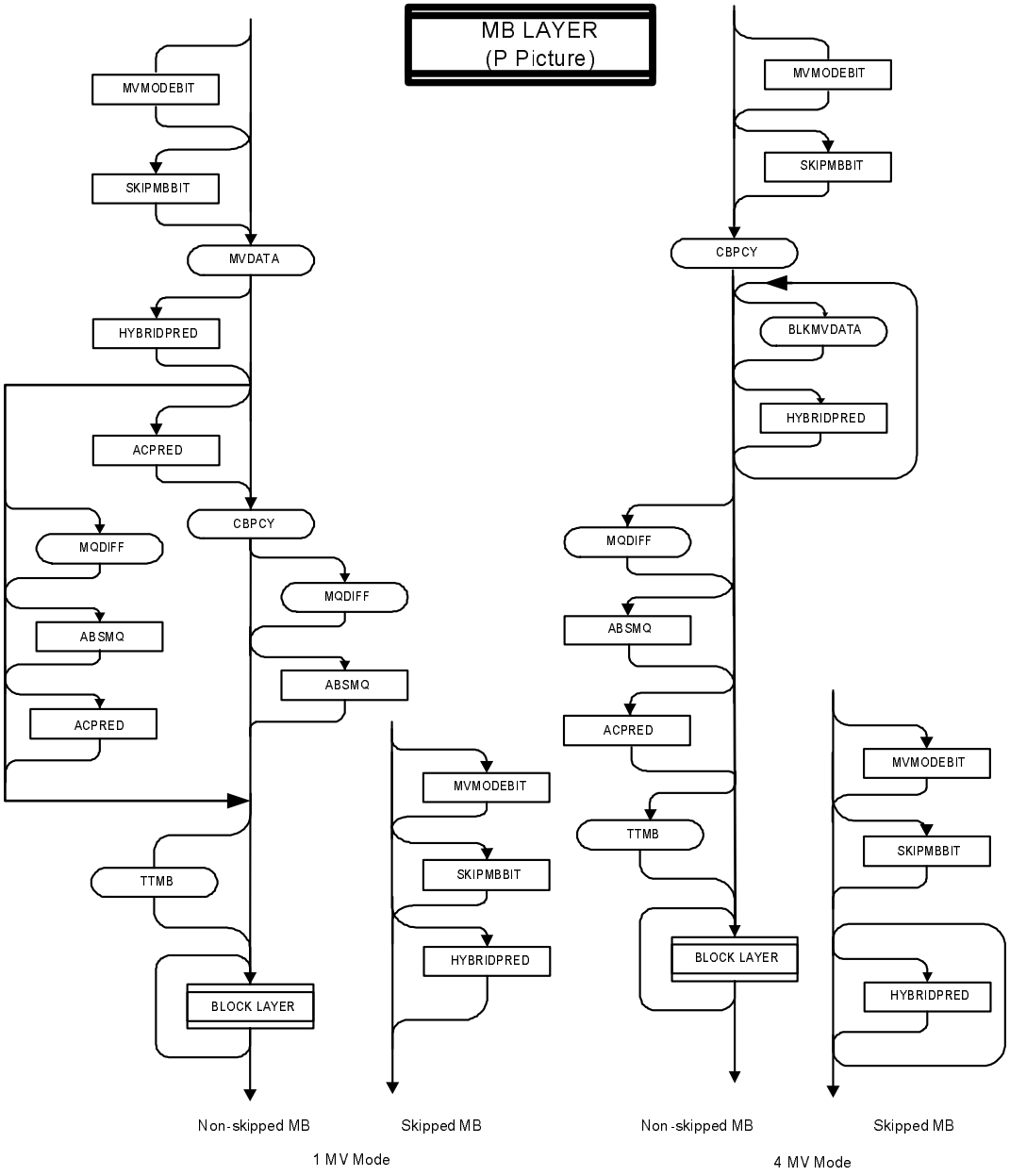


Figure 22: Syntax diagram for macroblock layer bitstream in Progressive-P picture for Simple/Main/Advanced Profiles

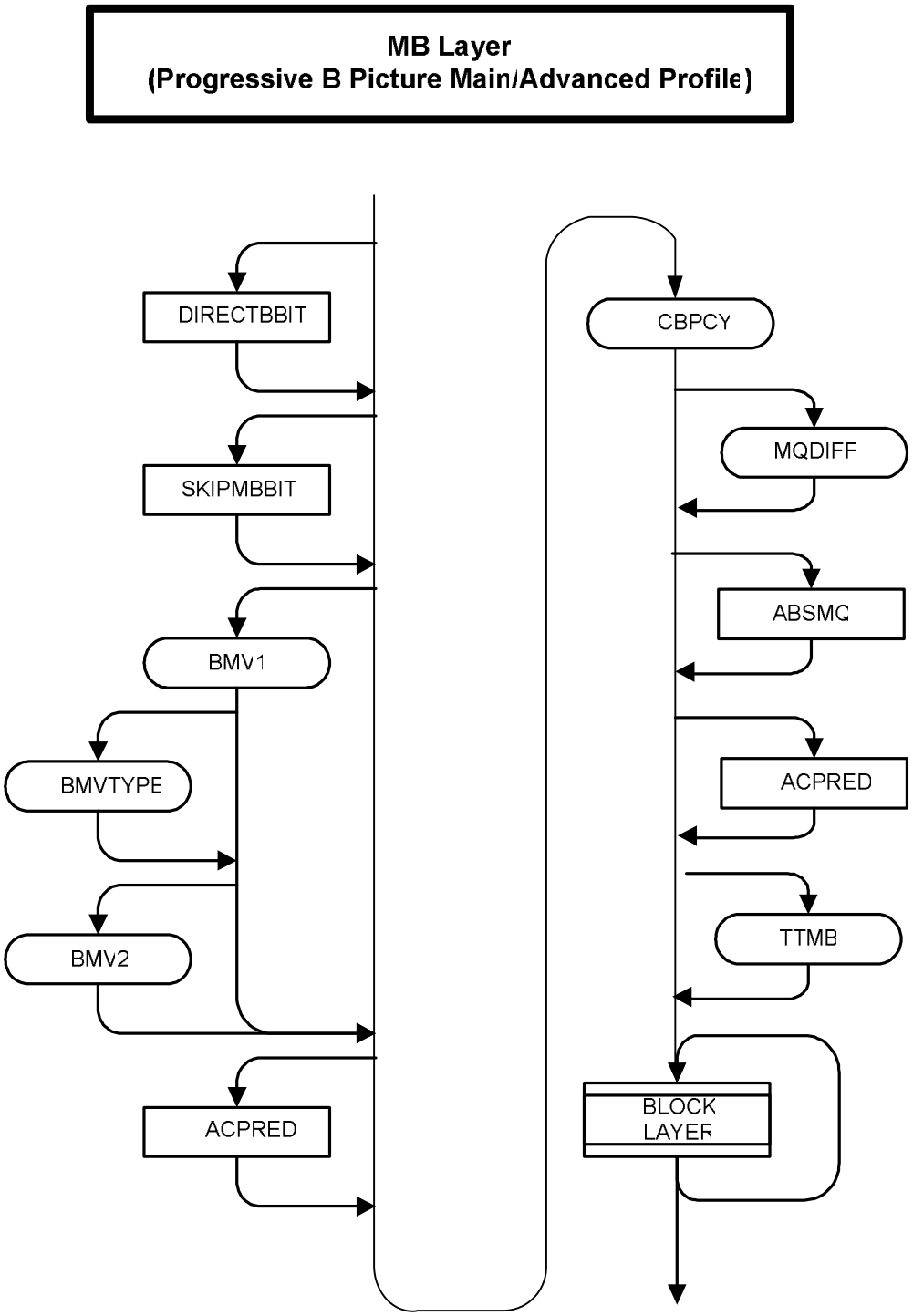


Figure 23: Syntax diagram for macroblock layer bitstream in Progressive B picture for Main/Advanced Profiles

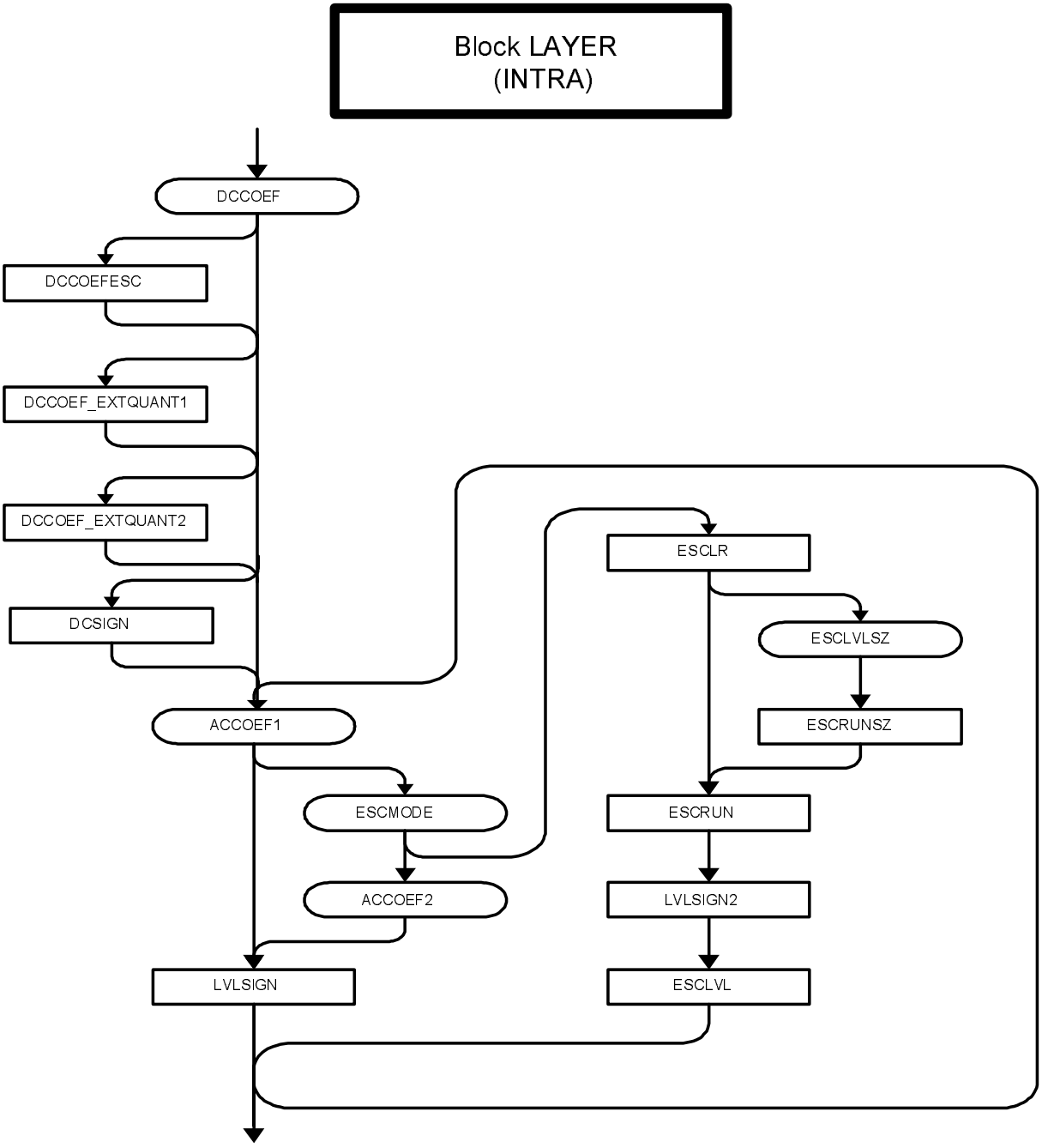


Figure 24: Syntax diagram for the Intra-coded block layer bitstream in Progressive mode.

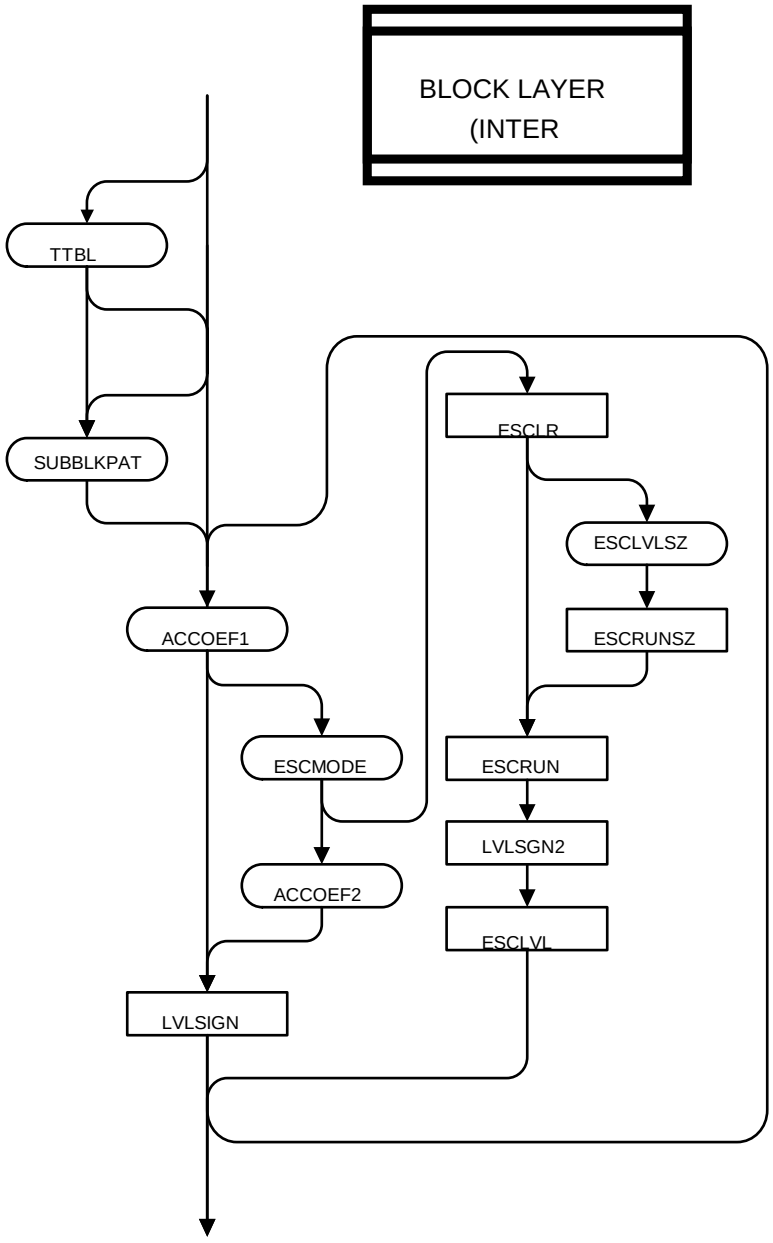


Figure 25: Syntax diagram for the Inter-coded block layer bitstream in Progressive mode.

The following tables show the syntax elements of the picture-layer and slice-layer bitstream.

Table 5: Progressive I picture layer bitstream for Simple and Main Profile

PICTURE LAYER( ) {	Number of bits	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
FRMCNT	2	
if (RANGERED == 1) {		

RANGEREDFRM	1	
}		
PTYPE	Var. size or 1	
BF	7	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	Only M.P.
}		
if (MULTIRES == 1) {		
RESPIC	2	Not B->I
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 6: Progressive BI picture layer bitstream for Main Profile

PICTURE LAYER( ) {	Number of bits	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
FRMCNT	2	
PTYPE	Var. size or 1	
BFRACTION	Var. size	
BF	7	
PQINDEX	5	

if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 7: Progressive I picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
If (INTERLACE == 1)		
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {		
TFCNTR	8	
}		
if (BROADCAST) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		

if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
ACPRED	Bitplane	
if (OVERLAP == 1 && 'PQUANT conditions') {		
CONDOVER	Variable size	
if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	

}		
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 8: Progressive BI picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
If (INTERLACE == 1)		
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {		
TFCNTR	8	
}		
if (BROADCAST) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	

}		
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
ACPRED	Bitplane	
if (OVERLAP == 1 && 'PQUANT conditions') {		
CONDOVER	Variable size	
if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 9: Progressive P picture layer bitstream for Simple and Main Profile

PICTURE LAYER( ) {	Number of bits	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
FRMCNT	2	

if (RANGERED == 1) {		
RANGEREDFRM	1	
}		
PTYPE	Var. size or 1	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	Not S.P.
}		
if (MULTIRES == 1) {		
RESPIC	2	
}		
MVMODE	Variable size	
if (MVMODE == 'intensity compensation') {		
MVMODE2	Variable size	
LUMSCALE	6	
LUMSHIFT	6	
}		
if (MVMODE == 'Mixed-MV' || (MVMODE == 'Intensity Compensation' && MVMODE2 == 'Mixed-MV') {		
MVTYPEMB	Bitplane	
}		
SKIPMB	Bitplane	
MVTAB	2	
CBPTAB	2	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	

if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 10: Progressive P picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
If (INTERLACE == 1)		
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {		
TFCNTR	8	
}		
if (BROADCAST) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		

}		
RNDCTRL	1	
UVSAMP	1	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
MVMODE	Variable size	
if (MVMODE == 'intensity compensation') {		
MVMODE2	Variable size	
LUMSCALE	6	
LUMSHIFT	6	
}		
if (MVMODE == 'Mixed-MV' || (MVMODE == 'Intensity Compensation' && MVMODE2 == 'Mixed-MV') {		
MVTYPEMB	Bitplane	
}		
SKIPMB	Bitplane	
MVTAB	2	
CBPTAB	2	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		

if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 11: Progressive B picture layer bitstream for Main Profile

PICTURE LAYER( ) {	Number of bits	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
FRMCNT	2	
if (RANGERED == 1) {		
RANGEREDFRM	1	
}		
PTYPE	Var. size or 1	
BFRACTION	Variable size	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	Not S.P.
}		
MVMODE	Variable size	

if (MVMODE == ‘Mixed-MV mode’) {		
MVTYPEMB	Bitplane	
}		
DIRECTMB	Bitplane	
SKIPMB	Bitplane	
MVTAB	2	
CBPTAB	2	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 12: Progressive B picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
If (INTERLACE == 1)		
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {		
TFCNTR	8	
}		
if (BROADCAST) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		

else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
if (FINTERPFLAG == 1) {		
INTERPFRM	1	
}		
BFRACTION	Variable size	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
MVMODE	Variable size	
if (MVMODE == ‘Mixed-MV mode’) {		
MVTYPEMB	Bitplane	

}		
DIRECTMB	Bitplane	
SKIPMB	Bitplane	
MVTAB	2	
CBPTAB	2	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 13: VOPDQUANT in Progressive picture header

VOPDQUANT( ) {	Number of bits	
if (DQUANT == 2) {		
PQDIFF	3	
if (PQDIFF == 7) {		
ABSPQ	5	
}		
}		
else {		
DQUANTFRM	1	
if (DQUANTFRM == 1) {		
DQPROFILE	2	
if (DQPROFILE == ‘single edge‘) {		
DQSBEDGE	2	

}		
if (DQPROFILE == ‘double edge‘) {		
DQDBEDGE	2	
}		
if (DQPROFILE == ‘all macroblock‘) {		
DQBILEVEL	1	
}		
if not (DQPROFILE == ‘all macroblock‘ and DQBILEVEL == 0) {		
PQDIFF	3	
if (PQDIFF == 7) {		
ABSPQ	5	
}		
}		
}		
}		
}		

Table 14: Slice-Layer bitstream in Advanced Profile

SLICE( ) {	Number of bits	
SLICE_ADDR	9	
PIC_HEADER_FLAG	1	
if (PIC_HEADER_FLAG == 1) {		
PICTURE_LAYER( )		
}		
for (‘all macroblocks’) {		
MB_LAYER ( )		
}		
}		

Table 15: Bitplane coding

BITPLANE( ) {		
INVERT	1	

IMODE	Variable size	
DATABITS	Variable size	
}		

Table 16: Macroblock layer bitstream in Progressive I picture for Simple/Main Profile

I PICTURE MB( ) {	Number of bits	
CBPCY	Variable size	
ACPRED	1	
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
}		

Table 17: Macroblock layer bitstream in Progressive I picture for Advanced Profile

I PICTURE MB( ) {	Number of bits	
CBPCY	Variable size	
if (‘ACPRED mode == RAW’) {		<
ACPRED	1	
}		
if (CONDOVER == 11b && OVERFLAGS == ‘raw mode’) {		<
OVERFLAGMB	1	
}		
if (DQUANTFRM) {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		

}		
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
}		

Table 18: Macroblock layer bitstream in Progressive P picture for Simple/Main/Advanced Profile

P PICTURE MB( ) {	Number of bits	
if (MVTYPEMB == ‘raw mode’) {		
MVMODEBIT	1	
}		
if (SKIPMB == ‘raw mode’) {		
SKIPMBBIT	1	
}		
if (1 MV mode) {		
if (non-skipped MB) {		
MVDATA	Variable size	
if (‘hybrid MV pred’) {		<
HYBRIDPRED	1	
}		
if (‘Intra MB’ && ‘last flag’ = = 0) {		
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
MQDIFF	Variable size	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
ACPRED	1	
}		
else if (‘last flag’ = = 1){		
if (‘Intra MB’) {		

ACPRED	1	
}		
CBPCY	Variable size	
if (DQUANTFRM) {		
if (DQPROFILE == ‘all macroblocks‘) {		
MQDIFF	Variable size	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
if (TTMBF == 0) {		
TTMB	Variable size	
}		
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
} /* non-skipped MB */		
else { /* skipped MB */		
if (‘hybrid MV pred’) {		<
HYBRIDPRED	1	
}		
} /* skipped MB */		
} /* 1 MV mode */		
else { /* 4 MV mode */		
if (non-skipped MB) {		
CBPCY	Variable size	
for (‘each of the 4 Y-blocks’) {		
if (‘CBPCY bit set for this block’) {		
BLKMVDATA	Variable size	
}		
if (‘hybrid MV pred’) {		<
HYBRIDPRED	1	
}		

}		
if (DQUANTFRM) {		
if (DQPROFILE == ‘all macroblocks‘) {		
MQDIFF	Variable size	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
if (‘any block is intra’ && ‘non-zero prediction for that block’) {		
ACPRED	1	
}		
if (TTMBF == 0) {		
TTMB	Variable size	
}		
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
} /* non-skipped MB */		
else { /* skipped MB */		
for (‘all 4 Y-blocks’) {		<
if (‘hybrid MV pred’) {		<
HYBRIDPRED	1	
}		
}		
} /* skipped MB */		
} /* 4 MV mode */		
}		

Table 19: Macroblock layer bitstream in Progressive B picture for Main/Advanced Profile

B PICTURE MB( ) {	Number of bits	
if (DIRECTMB == ‘raw mode’) {		

DIRECTBBIT	1	
}		
if (SKIPMB == ‘raw mode’) {		
SKIPMBBIT	1	
}		
if (!DIRECTBBIT) {		
if (SKIPMBBIT) {		
BMVTYPE	Variable size	
}		
else {		
BMV1	Variable size	
if (BMV1 does not indicate intra) {		
BMVTYPE	Variable size	
}		
}		
}		
If (SKIPMBBIT)		
goto End2:		
if (BMV1 == ‘last’) {		
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
MQDIFF	Variable size	
if (MQDIFF == 7)		
ABSMQ	5	
}		
}		
if (‘Intra MB’)		
ACPRED	1	
}		
else {		
if (FORWARDMB == ‘INTERPOLATE’)		
BMV2	Variable size	
If (BMV2 == ‘LAST’)		
goto End;		
if (‘Intra MB’)		

ACPRED	1	
CBPCY	Variable size	
if (DQUANTFRM) {		
if (DQPROFILE == ‘all macroblocks‘) {		
MQDIFF		
if (MQDIFF == 7)	Variable size	
ABSMQ	5	
}		
}	Variable size	
}		
End:		
if (TTMBF == 0) {		
TTMB	Variable size	
}		
End2:		
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
}		

The following tables show the syntax elements of the block-layer bitstream.

Table 20: Intra block layer bitstream in Progressive mode.

BLOCK( ) {	Number of bits	
DCCOEF	Variable size	
if (DCCOEF == ‘escape code’) {		
DCCOEFESC	Variable size	
}		
else if (DCCOEF != 0) {		
if (QUANT == 1)		
DCCOEF_EXTQUANT1	2	
else if (QUANT == 2)		

DCCOEF_EXTQUANT2	1	
DCSIGN	1	
}		
while (‘not last run’) {		
ACCOEF1	Variable size	
if (ACCOEF1 == ‘escape code’) {		
ESCMODE	Variable size	
if (‘escape mode 1 or 2’) {		
ACCOEF2	Variable size	
}		
else { /* ‘escape mode 3’ */		
ESCLR	1	
if (‘first escape mode 3 in this frame’) {		
ESCLVLSZ	Variable size	
ESCRUNSZ	2	
}		
ESCRUN	Calculated size	
LVLSGN2	1	
ESCLVL	Calculated size	
}		
} /* ‘escape mode’ */		
if (‘not escape mode 3’) {		
LVLSIGN	1	
}		
} /* while ( ) */		
}		

Table 21: Inter block layer bitstream in Progressive mode

BLOCK( ) {	Number of bits	
if (TTMB == ‘block’) {		
TTBLK	Variable size	
}		

if (‘transform type is 8*4, 4*8 or 4*4, and other cond’) {		
SUBBLKPAT	Variable size	
}		
while (‘not last run’) {		
ACCOEF1	Variable size	
if (ACCOEF1 == ‘escape code’) {		
ESCMODE	Variable size	
if (‘escape mode 1 or 2’) {		
ACCOEF2	Variable size	
}		
else { /* ‘escape mode 3’ */		
ESCLR	1	
if (‘first escape mode 3 in this frame’) {		
ESCLVLSZ	Variable size	
ESCRUNSZ	2	
}		
ESCRUN	Calculated size	
LVLSGN2	1	
ESCLVL	Calculated size	
}		
} /* ‘escape mode’ */		
if (‘not escape mode 3’) {		
LVLSIGN	1	
}		
} /* while ( ) */		
}		

7.1.1 Picture layer

Data for each picture consists of a picture header followed by data for the macroblock layer. Figure 10 and Figure 12 show the bitstream elements that make up the I progressive picture layer in simple/main profile and advanced profile, respectively, and Figure 14 and Figure 15 show the bitstream elements that make up the P progressive picture layer in simple/main profile and advanced profile, respectively. The following sections give a short description of each of the bitstream elements in the picture layer.

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

TFCNTR is present only in advanced profile, and only if the sequence level syntax element TFCNTRFLAG is 1. TFCNTR is an 8-bit fixed length field. When the sequence header includes a set of HRD parameters (HRD_flag set to '1'), TFCNTR of each coded frame shall increment by one modulo 256 when examined in display order at the output of the decoding process, except when a sequence header occurs. Among the frames coded immediately after the sequence header, the temporal reference of the coded frame that is displayed first 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.

7.1.1.2 Picture Coding Type (FCM) (Variable size)

FCM is present only in advanced profile, and only if the sequence level syntax element INTERLACE has the value 1, and it indicates whether the picture is coded as progressive, interlace-field or interlace-frame. Table 22 shows the VLC codewords used to indicate the picture coding type. NB: B pictures shall be constrained to be the same type (i.e. progressive, field-interlace or frame-interlace) as the first anchor frame that comes after them, i.e. all B pictures shall be of the same picture coding type as its backward reference picture.

Table 22: Picture Coding Type VLC

FCM	Picture Coding Type
0	Progressive
10	Frame-Interlace
11	Field-Interlace

7.1.1.3 Top Field First (TFF) (1 bit)

TFF is a one bit element present in advanced profile picture headers if the sequence header element BROADCAST is set to '1' and the sequence header element INTERLACE = 1. Note that TFF is not part of the decoding process, but it is used during display. TFF = 1 implies that the Top Field should be displayed first. If TFF = 0, the bottom field should be displayed first.

7.1.1.4 Repeat First Field (RFF) (1 bit)

RFF is a one bit element present in advanced profile picture headers if the sequence header element BROADCAST is set to '1' and the sequence header element INTERLACE = 1. Note that RFF is not part of the decoding process, but it is used during display. RFF = 1 implies that the first field should be repeated during display. RFF = 0 implies that no repetition is necessary.

7.1.1.5 Repeat Frame Count (RPTFRM) (2 bits)

RPTFRM is a two bit syntax element present in advanced profile picture headers if the sequence header element BROADCAST is set to '1' and the sequence header element INTERLACE = 0. RPTFRM takes value from 0 to 3 which are coded in binary using 2 bits. RPTFRM is not part of the decoding process, but it is used during display. It represents the number of a time is repeated during display.

7.1.1.6 Frame Interpolation Hint (INTERPFRM) (1 bit)

INTERPFRM is a 1-bit syntax element present in all progressive frame types for all profiles, if the syntax element FINTERPFLAG in the sequence header is 1. This bit is not used in the decoding process. Its intended purpose is to

provide a hint to the post-decoding (display) process that the current temporal region is suitable for temporal interpolation. The need to perform temporal frame interpolation or the method of doing so is outside the scope of this document. If INTERPFRM = 0 then the current temporal region is considered unsuitable for temporal frame interpolation. INTERPFRM = 1 then the current temporal region is considered suitable for temporal frame interpolation. For example, the renderer may use interpolation to increase the displayed frame-rate when INTERPFRM = 1. It is important to reemphasize that this interpolation is outside the decoding process.

7.1.1.7 Frame Count (FRMCNT) (2 bits)

FRMCNT is a 2-bit syntax element present in all picture headers for simple and main profiles. This syntax element may have any of the values from 0 to 3. FRMCNT has no effect on the decoding or display process.

7.1.1.8 Range Reduction Frame (RANGEREDFRM) (1 bit)

RANGEREDFRM is a 1-bit syntax element present in all frame types, for main profile, if the sequence level flag RANGERED = 1 (see section 6.1.23). If RANGEREDFRM = 1, then range reduction is used for the frame. If RANGEREDFRM = 0, then range reduction is not used for the frame. See sections 8.1.1.4 and 8.3.4.12 for a description of range reduction decoding.

7.1.1.9 Picture Type (PTYPE) (Variable size)

For simple and main profiles:

If the sequence level syntax element MAXBFRAMES = 0, then Table 23 is used to decode the PTYPE syntax element in the picture header.

Table 23: Simple/Main Profile Picture Type FLC if MAXBFRAMES = 0

PTYPE FLC	Picture Type
0	I
1	P

If MAXBFRAMES is greater than 0, then Table 24 is used to decoder the PTYPE syntax element in the picture header.

Table 24: Main Profile Picture Type VLC if MAXBFRAMES > 0

PTYPE VLC	Picture Type
1	P
01	I
00	B

For advanced profile:

Table 25 is used to decode the PTYPE syntax element in the picture header.

Table 25: Advanced Profile Picture Type VLC

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

1111	Skipped

A BI frame is treated as a B frame in the sense that it is coded out of order and is not used as a reference for other frames but it is coded using the I frame syntax.

If PTYPE indicates that the frame is skipped then the frame is treated 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. In simple and main profiles, if the size of any coded picture is less than or equal to one byte, that picture is treated as a skipped frame. Note that the size of coded picture is transmitted to the decoder via the Transport Layer in simple and main profiles.

7.1.1.10 B Picture Fraction (BFRACTION)(Variable size)

BFRACTION is variable-sized syntax element present in B picture headers. 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 is used to scale collocated motion vectors for deriving the direct motion vectors.

The mapping of BFRACTION is shown in Table 26. One symbol is unused in the codetable. When BFRACTION is 1111111, this means that the entire B frame is coded independent of its anchors, i.e. as an “Intra” frame. This frame is referred to as an *Intra B frame*. This is not a true I frame, since there is no temporal dependency on the Intra B frame, nor does this represent the start of an independently decodable segment.

Table 26: BFRACTION VLC Table

BFRACTION VLC	Fraction	BFRACTION VLC	Fraction
000	1/2	1110101	2/7
001	1/3	1110110	3/7
010	2/3	1110111	4/7
011	1/4	1111000	5/7
100	3/4	1111001	6/7
101	1/5	1111010	1/8
110	2/5	1111011	3/8
1110000	3/5	1111100	5/8
1110001	4/5	1111101	7/8
1110010	1/6	1111110	Invalid
1110011	5/6	1111111	See note below
1110100	1/7		

Note regarding last entry in Table 26: For simple and main profiles the 1111111 codeword indicates that the B frame is coded using the I frame syntax. For advanced profile this codeword is invalid since this case is coded using the PTYPE syntax element.

7.1.1.11 Buffer Fullness (BF) (7 bits)

BF is a 7-bit syntax element that is only present in simple and main profile I picture headers. BF shall be set to zero, and all other are forbidden.

7.1.1.12 Rounding Control Bit (RNDCTRL)(1 bit)

RNDCTRL is a 1 bit syntax element that is present in progressive advanced profile picture headers (I, P, B). The flag is used to indicate the type of rounding used for the current frame. If RNDCTRL = 1, the parameter R which controls rounding is set to 1. Otherwise, R is set to zero. See Section 8.3.7 for more details on the effect of R on rounding.

7.1.1.13 UV Sampling Format (UVSAMP)(1 bit)

UVSAMP is a 1 bit syntax element that is only present in advanced profile picture headers (I, P, B), when the sequence level field INTERLACE is 1. The flag is used to indicate the type of chroma subsampling used for the current frame. If UVSAMP = 1, then progressive subsampling of the chroma is used, otherwise, interlace subsampling of the chroma is used. This syntax element does not affect decoding of the bitstream.

7.1.1.14 Pan scan window coordinates (TOPLEFTX, TOPLEFTY, BOTRIGHTX, BOTRIGHTY)(4 X 16 bits)

TOPLEFTX, TOPLEFTY, BOTRIGHTX, BOTRIGHTY are four coordinates that specify each pan-scan window. Each occupies 16 bits, and is sent only in advanced profile when PANSCANFLAG = 1.

7.1.1.15 Picture Quantizer Index (PQINDEX) (5 bits)

PQINDEX is a 5-bit syntax element that signals 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 sequence syntax element QUANTIZER = 00, see section 6.1.25), then PQINDEX specifies both the picture quantizer scale (PQUANT) and the quantizer (uniform or nonuniform) used for the frame. See section 8.1.1.14 for details on dequantization using uniform as well as nonuniform quantizers. Table 27 shows how PQINDEX is translated to PQUANT and the quantizer for this case.

Table 27: PQINDEX to PQUANT/Quantizer Translation (Implicit Quantizer)

PQINDEX	PQUANT	Quantizer	PQINDEX	PQUANT	Quantizer
0	NA	NA	16	13	Nonuniform
1	1	Uniform	17	14	Nonuniform
2	2	Uniform	18	15	Nonuniform
3	3	Uniform	19	16	Nonuniform
4	4	Uniform	20	17	Nonuniform
5	5	Uniform	21	18	Nonuniform
6	6	Uniform	22	19	Nonuniform
7	7	Uniform	23	20	Nonuniform
8	8	Uniform	24	21	Nonuniform
9	6	Nonuniform	25	22	Nonuniform
10	7	Nonunifor	26	23	Nonunifor

		m			m
11	8	Nonuniform	27	24	Nonuniform
12	9	Nonuniform	28	25	Nonuniform
13	10	Nonuniform	29	27	Nonuniform
14	11	Nonuniform	30	29	Nonuniform
15	12	Nonuniform	31	31	Nonuniform

If the quantizer is signaled explicitly at the sequence or frame level (signaled by sequence syntax element QUANTIZER = 01, 10 or 11, see section 6.1.25), then PQINDEX is translated to the picture quantizer stepsize PQUANT as indicated by Table 28.

Table 28: PQINDEX to PQUANT Translation (Explicit Quantizer)

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

7.1.1.16 Half QP Step (HALFQP) (1 bit)

HALFQP is a 1-bit syntax element 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 stepsize is PQUANT + ½. If HALFQP = 0, then the picture quantizer stepize is PQUANT. Therefore, if the uniform quantizer is used, then half stepsizes are possible up to PQUANT = 9 (i.e., PQUANT = 1, 1.5, 2, 2.5 … 8.5, 9), and then only integer stepsizes are allowable above PQUANT = 9. For the nonuniform quantizer, half stepsizes are possible up to PQUANT = 7 (i.e., 1, 1.5, 2, 2.5 … 6.5, 7).

7.1.1.17 Picture Quantizer Type (PQUANTIZER) (1 bit)

PQUANTIZER is a 1 bit syntax element present in all frame types if the sequence level syntax element QUANTIZER = 01 (see section 6.1.25). In this case, the quantizer used for the frame is specified by PQUANTIZER. If PQUANTIZER = 0, then the nonuniform quantizer is used for the frame. If PQUANTIZER = 1, then the uniform quantizer is used.

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

MVRANGE is a variable-sized syntax element present for sequences coded using the main and advanced profiles when the sequence-layer EXTENDED_MV bit is set to 1. For the main profile, it is present in I, P and B pictures. For the advanced profile, it is present in P, and B pictures. 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. 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. The default range is chosen under Simple Profile, and when EXTENDED_MV is 0 under Main Profile encoding.

Table 29 lists the four possible binary codewords for MVRANGE and the corresponding motion vector range signaled by the codeword. Section 8.3.5.2 details the decoding of differential motion vectors for different ranges specified by MVRANGE.

Table 29: Motion Vector Range Signaled by MVRANGE

Codeword in binary	MV range in full pixel units (horiz x vert)
0 (also default)	[-64, 63.f] x [-32, 31.f]
10	[-128, 127.f] x [-64, 63.f]
110	[-512, 511.f] x [-128, 127.f]
111	[-1024,1023.f] x [-256, 255.f]

7.1.1.19 Picture Resolution Index (RESPIC) (2 bits)

The RESPIC syntax element is present in progressive I and P pictures, in simple and main profiles, if MULTIRES =1 in the sequence layer. This syntax element specifies the scaling factor of the current frame relative to the full resolution frame. Table 30 shows the possible values of the RESPIC syntax element. Refer to section 8.1.1.3 for a description of variable resolution coding. NOTE: The RESPIC syntax element of a P picture header shall carry the same value as the RESPIC syntax element of the closest temporally preceding I frame.

Table 30: **Progressive picture resolution code-table**

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

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

The SKIPMB syntax element is present only in P or B pictures. The SKIPMB syntax element encodes the skipped macroblocks using a bitplane coding method. Refer to section 8.7 for a description of the bitplane coding method.

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

The DIRECTMB syntax element is present only in B pictures. The DIRECTMB syntax element uses bitplane coding to indicate the macroblocks in the B picture that are coded in direct mode. The DIRECTMB syntax element may also signal that the direct mode is signaled in raw mode, in which case the direct mode is signaled at the macroblock level (see section 0). Refer to section 8.7 for a description of the bitplane coding method.

7.1.1.22 Motion Vector Mode (MVMODE) (Variable size)

The MVMODE syntax element is present in P and B picture headers. For P Pictures, the MVMODE syntax element signals one of four motion vector coding modes or one intensity compensation mode. In the bitstream corresponds to simple profile, the MVMODE syntax element shall not take the value corresponding to intensity compensation mode Depending on the value of PQUANT, either Table 31 or Table 32 is used to decode the MVMODE syntax element.

Table 31: P Picture Low rate (PQUANT > 12) MVMODE codetable

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

Table 32: P Picture High rate (PQUANT <= 12) MVMODE codetable

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

Intensity compensation is not signaled for B Pictures, and only two motion modes are valid. Table 33 and Table 34 show the tables used to code the motion vector mode for B Pictures.

Table 33: B Picture High rate (PQUANT <= 12) MVMODE codetable

MVMODE VLC	Mode
1	Quarter-pel Bicubic
000	Half-pel Bilinear

Table 34: B Picture Low rate (PQUANT > 12) MVMODE codetable

MVMODE VLC	Mode
1	Half-pel Bilinear
01	Quarter-pel Bicubic

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

The MVMODE2 syntax element is only present in P pictures, and only if the picture header syntax element MVMODE signals intensity compensation. Refer to section 8.3.4.3 for a description of motion vector mode/intensity compensation. Table 35 and Table 36 are used to decode the MVMODE2 syntax element.

Table 35: P Picture Low rate (PQUANT > 12) MVMODE2 codetable

MVMODE2 VLC	Mode
1	1 MV Half-pel bilinear
01	1 MV
001	1 MV Half-pel
000	Mixed MV

Table 36: P Picture High rate (PQUANT <= 12) MVMODE2 codetable

MVMODE2 VLC	Mode
1	1 MV
01	Mixed MV
001	1 MV Half-pel
000	1 MV Half-pel bilinear

7.1.1.24 Luminance Scale (LUMSCALE)(6 bits)

The LUMSCALE syntax element is only present in P pictures, and only if the picture header syntax element MVMODE signals intensity compensation. Refer to section 8.3.8 for a description of intensity compensation.

7.1.1.25 Luminance Shift (LUMSHIFT)(6 bits)

The LUMSHIFT syntax element is only present in P pictures, and only if the picture header syntax element MVMODE signals intensity compensation. Refer to section 8.3.8 for a description of intensity compensation.

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

The MVTYPEMB syntax element is present in P and B 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. 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 decode process.

7.1.1.27 Motion Vector Table (MVTAB) (2 bits)

The MVTAB syntax element is a 2-bit value present only in P and B frames. The MVTAB syntax element indicates which of four Huffman tables is used to encode the motion vector data. Refer to section 8.3.5.2 for a description of the motion vector decoding process.

Table 37: **MVTAB code-table**

FLC	Motion Vector Huffman Table
00	Huffman Table 0
10	Huffman Table 1
01	Huffman Table 2
11	Huffman Table 3

The motion vector Huffman tables are listed in section 11.10.

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

The CBPTAB syntax element is a 2-bit value present only in P and B frames. This syntax element signals the Huffman table used to decode the CBPCY syntax element (described in section 7.2.5.5) for each coded macroblock in P-pictures. Refer to section 8.3.4.6 for a description of how the CBP Huffman table is used in the decoding process.

The CBPCY Huffman tables are listed in sections 11.5 and 11.6.

7.1.1.29 Macroblock Quantization (VOPDQUANT) (Variable size)

The VOPDQUANT syntax element is made up of several bitstream syntax elements as shown in Figure 18. VOPDQUANT is present in Progressive P and B pictures when the sequence header DQUANT syntax element is nonzero. VOPDQUANT is present in Progressive I pictures only in advanced profile when the sequence header DQUANT syntax element is nonzero. The syntax of VOPDQUANT is dependent on the value of DQUANT.

Case 1: DQUANT = 1.

There are four possibilities in this case:

1. The macroblocks located on the boundary are quantized with a second quantization step size (ALTPQUANT), while the rest of the macroblocks are quantized with the frame quantization step size (PQUANT). Here boundary macroblocks are those macroblocks along the picture edges.
2. Two adjacent edges are signaled (see Table 40), and those macroblocks located on the two edges are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.
3. One edge is signaled and those macroblock located on the 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 select from two quantization steps (PQUANT or ALTPQUANT), or each macroblock may be arbitrarily quantized using any step size.

Case 2: DQUANT = 2.

The macroblocks located on the boundary (picture edges) are quantized with ALTPQUANT while the rest of the macroblocks are quantized with PQUANT.

7.1.1.30 VOPDQUANT Syntax Elements

The syntax elements are as follows:

- **Differential Quantizer Frame DQUANTFRM** (1 bit)

  The DQUANTFRM syntax element is a 1-bit value that is present only when DQUANT = 1. If DQUANTFRM = 0, then the current picture is only quantized with PQUANT.

- **Differential Quantizer Profile DQPROFILE** (2 bits)

  The DQPROFILE syntax element is a 2-bits value that is present only when DQUANT = 1 and DQUANTFRM = 1. It indicates where it is allowable to change quantization step sizes within the current picture.

Table 38: Macroblock Quantization Profile (DQPROFILE) Code Table

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

- **Differential Quantizer Single Boundary Edge DQSBEDGE** (2 bits)

  The DQSBEDGE syntax element is a 2-bits value that is present when DQPROFILE = Single Edge. It indicates which edge will be quantized with ALTPQUANT.

Table 39: Single Boundary Edge Selection (DQSBEDGE) Code Table

FLC	Boundary Edge
00	Left
01	Top
10	Right
11	Bottom

- **Differential Quantizer Double Boundary Edge DQDBEDGE** (2 bits)

  The DQDBEDGE syntax element is a 2-bits value that is present when DQPROFILE = Double Edge. It indicates which two edges will be quantized with ALTPQUANT.

Table 40: Double Boundary Edges Selection (DQDBEDGE) Code Table

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

- **Differential Quantizer Binary Level DQBILEVEL** (1 bit)

  The DQBILEVEL syntax element is a 1-bit value that is present when DQPROFILE = All Macroblock. If DQBILEVEL = 1, then each macroblock in the picture may take one of two possible values (PQUANT or ALTPQUANT). If DQBILEVEL = 0, then each macroblock in the picture may take on any quantization step size.

- **Picture Quantizer Differential PQDIFF** (3 bits)

  PQDIFF is a 3 bit syntax element that encodes either the PQUANT differential or an escape code.

  If PQDIFF does not equal 7, then PQDIFF encodes the differential, and the ABSPQ syntax element does not follow in the bitstream. In this case:

  ALTPQUANT = PQUANT + PQDIFF + 1

  Note that the value of ALTPQUANT has to be in the range of 1 to 31 for the bitstream to be valid. If PQDIFF equals 7, then the ABSPQ syntax element follows in the bitstream, and ALTPQUANT is decoded as:

  ALTPQUANT = ABSPQ

- **Absolute Picture Quantizer ABSPQ** (5 bits)

  ABSPQ is present in the bitstream if PQDIFF equals 7. In this case, ABSPQ directly encodes the value of ALTPQUANT as described above.

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

This syntax element is present only in P- and B- picture headers and only if the sequence-level syntax element VSTRANSFORM = 1 (described in section 6.1.19). If TTMBF = 1, then the TTFRM syntax element is also present in the picture layer. See section 8.3.4.8 for a description.

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

This syntax element is present in P- and B- picture headers if VSTRANSFORM = 1 and TTMBF = 1. The TTFRM syntax element is decoded using Table 41. See section 8.3.4.9 for a description.

Table 41: **Transform type select code-table**

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

7.1.1.33 AC Prediction (ACPRED)(Variable size)

For advanced profile I and BI pictures, the 1-bit ACPRED syntax elements present in all macroblocks are jointly coded using a bitplane coded syntax element that indicates the AC prediction status for each macroblock in the picture. The decoded bitplane represents the AC prediction status for each macroblock as a syntax element of 1-bit values in raster scan order from upper left to lower right. Refer to section 7.2 for a description of the bitplane coding. See section 8.1.1.8 for a description of AC prediction.

7.1.1.34 Conditional Overlap Flag (CONDOVER) (Variable size)

This syntax element is present only in I pictures, and only in advanced profile, and only when OVERLAP is on and PQUANT is less than or equal to 8 (regardless of HALFQP). CONDOVER may take the values 0 binary, 10 binary, and 11 binary. For the meaning of these values, and how CONDOVER affects overlap smoothing in advanced profile, see section 8.5.2.

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

This syntax element is present only in I pictures, and only in advanced profile, and only when CONDOVER has the binary value 11. OVERFLAGS is coded as a bitplane, which in raw mode requires that each macroblock carry its local information, OVERFLAGMB.

7.1.1.36 Post Processing (POSTPROC)(2 bits)

POSTPROC is a 2-bits syntax element that occurs in all pictures in advanced profile when the sequence level flag POSTPROCFLAG is set to 1. It is used to suggest to the decoder on the level of post processing that should be used for the current frame. The four suggested modes are tabulated below:

POSTPROC	Suggested Post processing
00	No Post Processing
01	De-blocking
10	De-ringing
11	De-block + De-ringing

Post processing is outside the decoding loop and is therefore not a normative part of the VC-9 specification[1].

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

This syntax element is present in all progressive frame types and all interlaced frame types in Advanced. Table 42 is used to decode the TRANSACFRM syntax element. See section 8.3.4.10 for a description of the TRANSACFRM syntax element.

Table 42: **Transform AC coding set index code-table**

VLC	Coding set index
0	0
10	1
11	2

See section 8.1.1.10 for a description of the Transform AC coding sets.

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

This syntax element is present in progressive I frames and interlaced I frames in Advanced Profile. Table 42 is used to decode the TRANSACFRM2 syntax element. See section 8.1.1.10 for a description of the Transform AC coding sets.

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

This syntax element is present in all frame types. See section 8.1.1.2 for a description.

7.1.2 Slice Layer

A *slice* represents one or more contiguous rows of macroblocks that are scanned in raster-scan order. Slice-layer is present only in the advanced profile. Even in advanced profile, slice layer is optional, and may be skipped by coding a

[1] An informative comment: MPEG-4 style deblocking and deringing operations may be used as post processing filters by a VC-9 decoder.

picture as a single independent decodable unit (IDU). When a picture is coded in multiple IDUs, slices are used. Note that a slice always begins at the first macroblock in a row, and ends at the last macroblock in the same or another row. Thus, a slice contains an integer number of complete rows. A slice is always byte-aligned, and each slice is transmitted in a different IDU. The beginning of a new slice is detected through search for start-codes as outlined in Annex E.

When a new slice begins, motion vector predictors, predictors for AC and DC coefficients, and the predictors for quantization parameters are reset. In other words, with respect to prediction, the first row of macroblocks in the slice is 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 is carried in raw mode which ensures that each macroblock carries its own local information. For the purposes of deblocking, each slice is 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 is no loop-filtering across slices.

Figure 19 shows the structure for the slice layer. The elements that make up the slice layer are described in the following sections.

7.1.2.1 Slice Address (SLICE_ADDR)(9 bits)

SLICE_ADDR is a fixed-length 9-bit syntax element. The row address of the first macroblock row in the slice is binary encoded in this syntax element. The range of this syntax element is from 0 to 511. (Informative – The maximum picture size of 8192 corresponds to a maximum of 512 macroblock rows).

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

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

7.1.3 Macroblock Layer

Data for each macroblock consists of a macroblock header followed by the block layer. Figure 20 – Figure 23, and Table 16 - Table 19 show the macroblock layer structure for I picture and P picture macroblocks. The elements that make up the macroblock layer are described in the following sections. The picture types that the macroblock layer syntax elements occur in, are indicated in the square brackets.

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

This syntax element is present only in I pictures, only in advanced profile, and only when CONDOVER has the binary value 11, and when the raw mode is chosen to encode the OVERFLAGS plane. In this case, one bit is sent 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.

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

MVMODEBIT is a 1-bit syntax element present in P frame macroblocks if the frame level syntax element MVTYPEMB (see section 7.1.1.26) indicates that raw mode is used. For definition of raw mode, see section 7.2. If MVMODEBIT = 0, then the macroblock is coded in 1MV mode, and if MVMODEBIT = 1, then the macroblock is coded in 4MV mode.

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

SKIPMBBIT is a 1-bit syntax element present in P and B frame macroblocks if the frame level syntax element SKIPMB (see section 7.1.1.20) indicates that raw mode is used. For definition of raw mode, see section 7.2. If SKIPMBBIT = 1, then the macroblock is skipped. See section 8.3.4.4 for details on skipped macroblocks.

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

CBPCY is a variable-length syntax element present in I picture, P picture and B macroblock layers. Section 8.1.1.5 describes the CBPCY syntax element in I picture macroblocks and section 7.2.5.5 describes the CBPCY syntax element in P picture and B picture macroblocks.

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

The ACPRED syntax element is present in all I picture macroblocks and in Intra macroblocks in P pictures and B Pictures. The (see section 8.3.5.1 for a description of the macroblock types). ACPRED is also present in a 4MV macroblock in P pictures, if atleast one of the blocks in that macroblock is intra-coded, and if that block(s) has a non-zero predictor. (See section 8.3.6.1.2 for details on determining if a block has a non-zero predictor). This is a 1-bit syntax element that specifies whether the blocks were coded using AC prediction. ACPRED = 0 indicates that AC prediction is not used. ACPRED = 1 indicates that AC prediction is used. In advanced profile I pictures, this bit could be jointly coded using bitplane coding and sent at the picture layer instead of being sent at the macroblock level. See section 8.1.1.6 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.

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

MQDIFF is a variable-sized syntax element present in Progressive P and B pictures. MQDIFF is also present in progressive I pictures in advanced profile. It is present only if the picture layer syntax element DQPROFILE = All Macroblocks. The syntax depends on the DQBILEVEL syntax element as described below.

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

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

If MQDIFF does not equal 7, then MQDIFF encodes the differential, and the ABSMQ syntax element does not follow in the bitstream. In this case:

MQUANT = PQUANT + MQDIFF

Note that MQUANT has to be in the range of 1 to 31 for the bitstream to be valid. If MQDIFF equals 7, then the ABSMQ syntax element follows in the bitstream, and MQUANT is decoded as:

MQUANT = ABSMQ

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

ABSMQ is present in the bitstream if MQDIFF equals 7. In this case, ABSMQ directly encodes the value of MQUANT as described above.

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

MVDATA is a variable sized syntax element present in P picture macroblocks. This syntax element encodes the motion vector(s) for the macroblock. The Huffman table used to decode this syntax element is specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.27. See section 8.3.5.2.1 for a description of the motion vector decode process.

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

BLKMVDATA is a syntax element that contains motion information for the block. It is a variable sized syntax element and is only present in certain situations. The Huffman table used to decode this syntax element is specified by the MVTAB syntax element in the picture layer as specified in section 7.1.1.27. See section 8.3.5.2.1 for a description of when the BLKMVDATA syntax element is present and how it is used.

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

HYBRIDPRED is a 1-bit syntax element per motion vector, present in P picture macroblocks. Section 8.3.5.3.4 describes how HYBRIDPRED is used in the decoding process.

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

The TTMB syntax element is a variable syntax element present in P and B picture macroblocks, if the picture layer syntax element TTMBF = 0. As shown in Table 43, Table 44, and

Table **45**, the TTMB syntax element specifies the transform type, the signal level and the subblock pattern. If the signal type specifies macroblock mode, the transform type decoded from the TTMB syntax element is used to decode all coded blocks in the macroblock. If the signal type signals block mode, then the transform type decoded from the TTMB syntax element is used to decode the first coded block in the macroblock. The transform type of the remaining blocks is coded at the block level. If the transform type is 8x4 or 4x8, then the subblock pattern indicates the subblock pattern of the first block.

The table used to decode the TTMB syntax element depends on the value of PQUANT. For PQUANT less than or equal to 4, Table 43 is used. For PQUANT greater than 4 and less than or equal to 12, Table 44 is used. For PQUANT greater than 12,

Table 45 is used.The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

Table 43: High Rate (PQUANT < 5) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
11	8x8	Block	NA
101110	8x4	Block	Bottom
1011111	8x4	Block	Top
00	8x4	Block	Both
10110	4x8	Block	Right
10101	4x8	Block	Left
01	4x8	Block	Both
100	4x4	Block	NA
10100	8x8	Macroblock	NA
1011110001	8x4	Macroblock	Bottom
101111001	8x4	Macroblock	Top
101111011	8x4	Macroblock	Both
101111000000	4x8	Macroblock	Right
101111000001	4x8	Macroblock	Left
10111100001	4x8	Macroblock	Both
101111010	4x4	Macroblock	NA

Table 44: Medium Rate (5 <= PQUANT < 13) TTMB VLC Table

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
110	8x8	Block	NA

0110	8x4	Block	Bottom
0011	8x4	Block	Top
0111	8x4	Block	Both
1111	4x8	Block	Right
1110	4x8	Block	Left
000	4x8	Block	Both
010	4x4	Block	NA
10	8x8	Macroblock	NA
0010100	8x4	Macroblock	Bottom
0010001	8x4	Macroblock	Top
001011	8x4	Macroblock	Both
001001	4x8	Macroblock	Right
00100001	4x8	Macroblock	Left
0010101	4x8	Macroblock	Both
00100000	4x4	Macroblock	NA

Table 45: Low Rate (PQUANT >= 13) TTMB VLC Table

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

001000001	4x8	Macroblock	Left
0010001	4x8	Macroblock	Both
00100000000	4x4	Macroblock	NA

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

DIRECTBBIT is a 1-bit syntax element present in B frame macroblocks if the frame level syntax element DIRECTMB (see section 7.1.1.21) indicates that raw mode is used. If DIRECTBBIT = 1, then the macroblock is coded using direct mode. See section 8.4.3.2 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 present in B picture macroblocks. This syntax element encodes the first motion vector for the macroblock. See section 8.3.5.2 for a description of the motion vector decode process.

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

BMV2 is a variable sized syntax element present in B picture macroblocks if the Interpolation mode is used. This syntax element encodes the second motion vector for the macroblock. See section 8.3.5.2 for a description of the motion vector decode process.

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

BMVTYPE is a variable sized syntax element present in B frame macroblocks that indicates whether the macroblock uses forward, backward or interpolated prediction. As Table 46 shows, the value of BFRACTION (in the picture header, see section 7.1.1.10) along with BMVTYPE determine which type is used.

Table 46: B Frame Motion Prediction Type

BMVTYPE VLC	Motion Prediction Type	
	BFRACTION <= 1/2	BFRACTION > 1/2
0	Backward	Forward
10	Forward	Backward
11	Interpolated	Interpolated

7.1.4 Block Layer

Figure 24 and Figure 25 show the block layer syntax elements for intra and inter-coded blocks. 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 Block-level Transform Type (TTBLK)(Variable size)[inter]

The TTBLK syntax element is present only in inter-coded blocks and only if the macroblock level syntax element TTMB (see section 7.1.3.11) 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 codes the transform type for the 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 is less than or equal to 4, then Table 47 is used. If PQUANT is greater than 4 and less than or equal to 12, then Table 48 is used. If PQUANT is greater than 12, then Table 49 is used. The TTBLK syntax element is not present for the first block in each macroblock since the transform type and subblock pattern decoded in TTMB is used for the first block. TTBLK is present for each coded block after the first. The subblock pattern indicates which of 8x4 or 4x8 subblocks have at least one non-zero coefficient.

Table 47: High Rate (PQUANT < 5) TTBLK VLC Table

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

Table 48: Medium Rate (5 =< PQUANT < 13) TTBLK VLC Table

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

Table 49: Low Rate (PQUANT >= 13) TTBLK VLC Table

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

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

The SUBBLKPAT syntax element is only present in inter-coded blocks and only if the transform type for the block is 8x4, 4x8 or 4x4.

For 4x4 transform types, the SUBBLKPAT syntax element indicates which of the 4 4x4 subblocks have at least one non-zero coefficient.

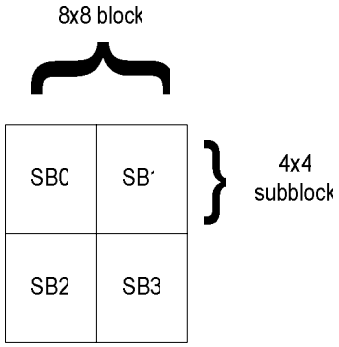


Figure 26: 4x4 Subblocks

The subblock pattern is coded as a 4 bit syntax element where each bit indicates whether the corresponding subblock contains at least one non-zero coefficient. Figure 26 shows the labeling of the 4 subblocks that make up an 8x8 block. The subblock pattern is coded is coded 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 encode the subblock pattern. The table used depends on the value of PQUANT. If PQUANT is less than or equal to 4, then Table 50 is used. If PQUANT is greater than 4 and less than or equal to 12, then Table 51 is used. If PQUANT is greater than 12, then Table 52 is used.

Table 50: High Rate (PQUANT < 5) SUBBLKPAT VLC Table

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

Table 51: Medium Rate (5 =< PQUANT < 13) SUBBLKPAT VLC Table

SUBBLKPAT	Subblock	SUBBLKPAT	Subblock

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

Table 52: Low Rate (PQUANT >= 13) SUBBLKPAT VLC Table

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

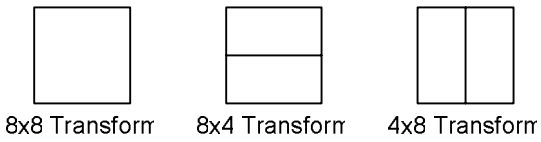


Figure 27: 8x4 and 4x8 Subblocks

For 8x4 or 4x8 transform types, the SUBBLKPAT syntax element specifies which of the two sub-blocks have at least one non-zero coefficient. The data is encoded with the following VLC table (an X indicates that the sub-block contains at least one non-zero coefficient):

Table 53: **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
0		X		X
10	X	X	X	X
11	X		X	

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

The DCCOEF syntax element is only present in intra-coded blocks. This is a variable-length codeword that encodes the Transform DC differential. Refer to section 8.1.1.7 for a description of the Transform DC decoding process. One of two code tables is used to encode the DC differentials (the table is signaled in the TRANSDCTAB syntax element in the picture header as described in section 8.1.1.2). Section 11.7 lists the DC Huffman tables.

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

The DCCOEFESC syntax element is only present in intra-coded blocks and only if DCCOEF decodes to the escape code. 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 8.1.1.7 for a description of the Transform DC decoding process.

7.1.4.5 Transform DC Coefficient Extension for Quant1 (DCCOEF_EXTQUANT1)(Variable size)[intra]

The DCCOEF_EXTQUANT1 is a 2-bit syntax element is only 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 is 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 8.1.1.7 for a description of the Transform DC decoding process.

7.1.4.6 Transform DC Coefficient Extension for Quant2 (DCCOEF_EXTQUANT2)(Variable size)[intra]

The DCCOEF_EXTQUANT2 is a 1-bit syntax element is only 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 2. This syntax element is 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 8.1.1.7 for a description of the Transform DC decoding process.

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

DCSIGN is a one-bit value that indicates the sign of the DC differential. It is present only 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.

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

ACCOEF1 is present in both intra and inter blocks. This is a variable-length codeword that encodes the run, level and last_flag for each non-zero AC coefficient. Refer to section 8.1.1.10 for a description of the Transform AC decoding process. One of three code tables is used to encode ACCOEF1. The table is signaled in the picture or macroblock headers. Section 11.8 lists the AC Huffman tables.

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

ACCOEF2 may be present in both intra and inter blocks. It is only present if ACCOEF1 decodes to the escape code and if the ESCMODE syntax element (described in section 7.1.4.10) specifies AC decoding escape mode 1 or 2 (refer to section 8.1.1.10 for a description of the Transform AC decoding process). One of three code tables is used to encode ACCOEF2. The table is signaled in the picture or macroblock headers. Section 11.8 lists the AC Huffman tables.

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

ESCMODE may be present in both intra and inter blocks. It is only present if ACCOEF1 decodes to the escape code. ESCMODE is a variable-length codeword that signals which of three escape decoding methods are used to decode the AC coefficient. Table 54 shows the code-table used to encode the escape modes.

Table 54: **AC escape decoding mode code-table**

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

If mode 1 or mode 2 decoding is specified, then the bitstream contains the ACCOEF2 element as described in section 7.1.4.9. If mode 3 is specified, then the bitstream contains the ESCLR, ESCRUN, ESCLVL and LVLSIGN2 elements and may contain the ESCLVLSZ and ESCRUNSZ elements, as described in sections 6.1.4.9 - 7.1.4.17.

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

LVLSIGN may be present in both intra and inter blocks. It will always be present unless ESCMODE specifies AC decoding mode 3. LVLSIGN is a one-bit value that specifies the sign of the AC level. Refer to section 8.1.1.10 for a description of the Transform AC decoding process. If LVLSIGN = 0, then the level is positive. If LVLSIGN = 1, then the level is negative.

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

ESCLR may be present in both intra and inter blocks. It is only present if ESCMODE specifies AC decoding escape mode 3. ESCLR is a one-bit value that specifies whether this coefficient is the last non-zero coefficient in the block. If ESCLR = 1, then this is the last non-zero coefficient. If ESCLR = 0, then this is not the last non-zero coefficient.

7.1.4.13 Escape Mode 3 Run (ESCRUN)(Calculated size)[both]

ESCRUN may be present in both intra and inter blocks. It is only present if ESCMODE specifies AC decoding escape mode 3. The size of the ESCRUN codeword is fixed throughout the frame, with the size being specified in the ESCRUNSZ syntax element described in section 7.1.4.17. ESCRUN directly encodes the run value for the coefficient. For example, if the size (from ESCRUNSZ) is 4 bits and the value is [0101], then the run is decoded as 5.

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

ESCLVL may be present in both intra and inter blocks. It is only 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 7.1.4.16. ESCLVL directly encodes the level value for the coefficient. For example, if the size (from ESCLVLSZ) is 3 bits and the value is [110], then the run is decoded as 6.

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

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

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

ESCLVLSZ may be present in both intra and inter blocks. It is only present if ESCMODE specifies AC decoding escape mode 3 and if this is the first time mode 3 has been signaled within the current frame (in other words, all subsequent instances of escape mode 3 coding within this frame do not have this syntax element). ESCLVLSZ is used

to specify the codeword size for the mode 3 escape-coded level values for the entire frame. Two different VLC tables are used to encode ESCLVLSZ, depending on the value of PQUANT, and other information.

The *conservative* table is used when PQUANT is between 1 and 7, both values inclusive. The *efficient* table is used when PQUANT is 8 and higher. 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. When DQUANT is used within a frame, the conservative table is used regardless of PQUANT.

The two tables are as follows:

Table 55: Escape mode 3 level codeword size *conservative* code-table (used typically for 1 <= PQUANT <= 7)

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

Table 56: **Escape mode 3 level codeword size *efficient* code-table (used typically for 8 <= PQUANT <= 31)**

8 <= PQUANT <= 31	
ESCLVLSZ VLC	Level codeword size
1	2
01	3
001	4

0001	5
00001	6
000001	7
000000	8

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

ESCRUNSZ may be present in both intra and inter blocks. It is only present if ESCMODE specifies AC decoding escape mode 3 and is only present the first time escape mode 3 is signaled within the frame. ESCRUNSZ is used to specify the codeword size for the mode 3 escape-coded run values for the entire frame. The run codeword size is encoded according to Table 57:

Table 57: **Escape mode 3 run codeword size code-table**

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

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 0 and the presence of a non-skipped macroblock is signaled with a bit set to 1. These bits are coded as a frame-level bitplane. The following diagram shows the elements that make up the bitplane.

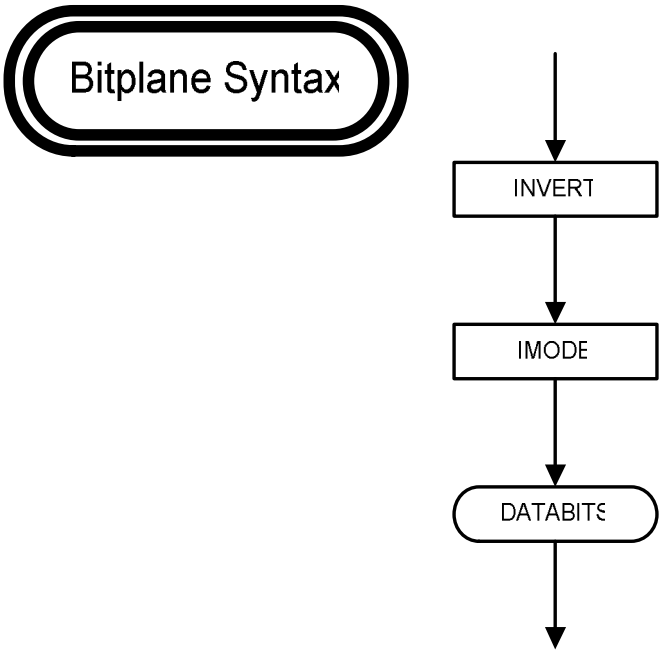


Figure 28: Syntax diagram for the bitplane coding

7.2.1 Invert Flag (INVERT)

The INVERT syntax element is a 1-bit value. 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)

The IMODE syntax element is a variable length value that indicates the coding mode used to encode the bitplane. Table 58 shows the codetable used to encode the IMODE syntax element. Refer to section 7.6.2 for a description of how the IMODE value is used in decoding the bitplane.

Table 58: **IMODE VLC Codetable**

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

7.2.3 Bitplane Coding Bits (DATABITS)

The DATABITS syntax element is variable sized syntax element that encodes the bitplane. The method used to encode the bitplane is 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 process for progressive I pictures, and Progressive P pictures.

8.1 Progressive I Frame Decoding

The following sections describe the process for decoding progressive I pictures.

8.1.1 Progressive I Picture Layer Decode

Figure 10 shows the elements that make up the I picture layer header for simple and main profiles. Figure 12 shows the elements that make up the I picture layer header for advanced profile. Some of the elements are self-explanatory. The following sections provide extra detail for some of the elements.

8.1.1.1 Frame-level Transform AC Table Index

TRANSACFRM and TRANSACFRM2 are variable-length syntax elements that are present in the picture layer. The TRANSACFRM and TRANSACFRM2 syntax elements provide the indices that select the coding sets used to decode the Transform AC coefficients for the Y and U/V blocks, respectively. Table 42 is used to decode the TRANSACFRM and TRANSACFRM2 syntax elements. Refer to section 8.1.1.10 for a description of AC coefficient decoding.

8.1.1.2 Intra Transform DC Table

TRANSDCTAB is a one-bit syntax element that signals which of two Huffman tables is used to decode the Transform DC coefficients in intra-coded blocks. If TRANSDCTAB = 0, then the low motion huffman table is used. If TRANSDCTAB = 1, then the high motion huffman table is used. Section 11.7 lists the Transform DC Huffman tables.

8.1.1.3 Picture Resolution Index

The RESPIC syntax element in I pictures, in simple and main profiles, specifies 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. Table 30 shows how the scaling factor is encoded in the RESPIC syntax element.

The resolution encoded in the I picture RESPIC syntax element also applies to all subsequent P pictures until the next I picture. In other words, all P pictures are encoded at the same resolution as the first 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 following pseudo-code illustrates how the new frame dimensions are calculated if a downsampled resolution is indicated.

X = full resolution horizontal dimension in samples
Y = full resolution vertical dimension in samples
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 29: Calculation of Frame Dimensions in Multires Downsampling Pseudo-code

If the decoded picture is one of the subsampled resolutions, then it shall be upsampled to full resolution prior to display. Since this upsampling process is outside the decoding loop, the implementer is free to use any upsampling filter. However, attention should be paid to the relative spatial positioning of the samples produced from the upsampling and downsampling processes. In particular, spatial alignment of the video samples of the downsampled frame with respect to the video samples of the frame at the original resolution should follow the specification in Annex B.

8.1.1.4 Range Reduction Frame - I Frame (RANGEREDFRM)

The RANGEREDFRM is only signaled when RANGERED is signaled at the sequence level.

When RANGEREDFRM is signaled for the current I Frame, the current decoded frame shall be scaled up prior to display while keeping the original reconstructed frame for use in future motion compensation. Let Y, U, V denote the YCbCr planes of the output frame. The pixels are scaled up according to the following formula:

Y[n] = CLIP ((Y[n] – 128) * 2 + 128);

U[n] = CLIP ((U[n] – 128) * 2 + 128);

V[n] = CLIP ((V[n] – 128) * 2 + 128);

Macroblock Layer Decode

Figure 3 shows how the frame is composed of macroblocks. The macroblocks are coded in raster scan order form left to right. Figure 20 shows the elements that make up the I picture macroblock layer.

8.1.1.5 Coded Block Pattern

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 Huffman table used to decode CBPCY is listed in section 11.5). The coded block pattern (*cbpcy*) is 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:

predicted_Y0 =
L1, if *LT3* equals *T2*
T2 otherwise
predicted_Y0 ^= ((decoded_cbpcy >> 5) & 0x01);

predicted_Y1 =
predicted_Y0, if *T2* equals *T3*
T3 otherwise
predicted_Y1 ^= ((decoded_cbpcy >> 4) & 0x01);

predicted_Y2 =
L3, if *L1* equals *predicted_Y0*
predicted_Y0 otherwise
predicted_Y2 ^= ((decoded_cbpcy >> 3) & 0x01);

predicted_Y3 =
predicted_Y2, if *predicted_Y0* equals *predicted_Y1*
predicted_Y1 otherwise
predicted_Y3 ^= ((decoded_cbpcy >> 2) & 0x01);

L0, L1, L2, L3, LT3, T0, T1, T2 and T3 are one-bit values representing the coded status of the neighboring luminance blocks as illustrated in Figure 30. If the neighboring block is outside the frame boundaries, or if the neighboring block belongs to a different slice, its coded status is set to zero. The figure shows the four luminance 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 indicate 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.

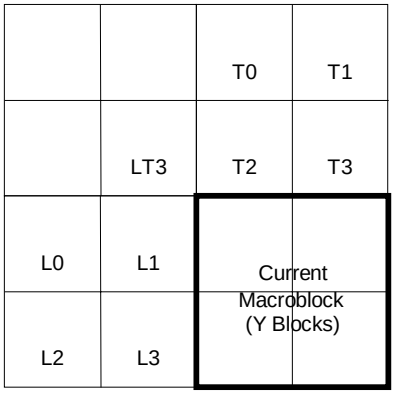


Figure 30: CBP encoding using neighboring blocks

The six-bit coded block pattern (*cbpcy*) specifies 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 correspond to the six blocks as shown in Table 59 (bit position 0 is the rightmost bit):

Table 59: **Coded block pattern bit position**

	Coded Block Pattern Bit Position					
	5	4	3	2	1	0
Block	Y1	Y2	Y3	Y4	Cb	Cr

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

8.1.1.6 AC Prediction Flag

The ACPRED syntax element in the macroblock header is a one-bit syntax element that specifies whether AC prediction is used to decode the AC coefficients for all the blocks in the macroblock. Section 8.1.1.13 describes the AC prediction process. If ACPRED is 1, then AC prediction is used, otherwise it is not used.

Block Decode

Figure 3 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 coded first followed by the Cb and Cr blocks. This section describes the process used to reconstruct the blocks.

Figure 4 shows the forward intra-coding steps used to encode the 8x8 pixel blocks. Figure 31 shows the inverse process used to reconstruct the 8x8 blocks.

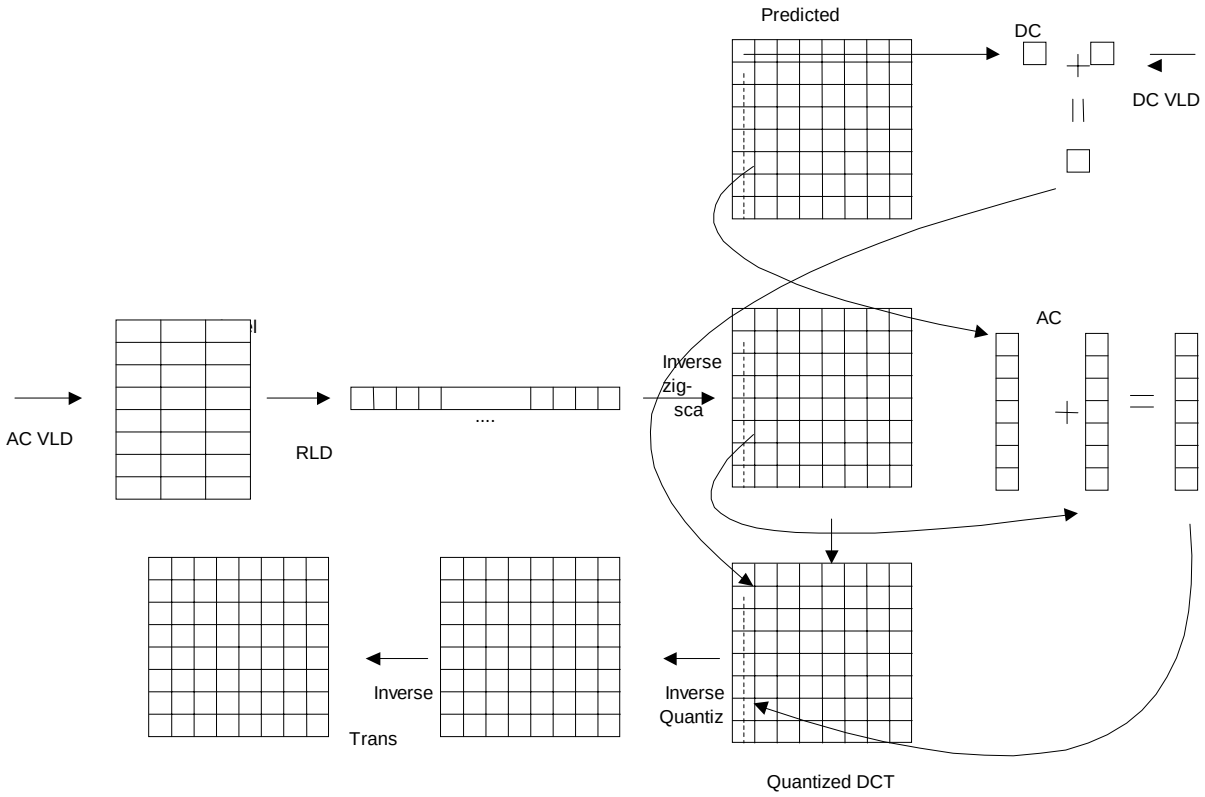


Figure 31: Intra block reconstruction

As Figure 31 shows, the DC and AC Transform coefficients are coded using separate techniques. The DC coefficient is coded 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 in I pictures

8.1.1.7 DC Differential Bitstream Decode

The DC coefficient is coded 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 24 shows the bitstream elements used to encode the DC differential. DCCOEF is decoded using one of two VLC code tables. The table is 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 Huffman tables listed in section 11.7 is used to decode DCCOEF. This will yield either:

1) Zero, or
2) the absolute value of the DC differential, or
3) The escape code.

If DCCOEF decodes to zero, the value of the DC differential is also zero. Other wise, further decoding is necessary to determine the value of DC differential. If the DCCOEF decodes to the escape code, the absolute value of the DC differential is encoded in the DCCOEFESC syntax element (section 7.1.4.4). The size of the DCCOEFESC syntax element may be 8, 9 or 10 bits depending on the quantization step size of the block. If the DCCOEF does not decode to the escape code, and the quantizer step size is 1, an additional 2-bit syntax element DCCOEF_EXTQUANT1 is decoded, and this is used to refine the value of DC differential. If the DCCOEF does not decode to the escape code, and the quantizer stepsize is 2, an additional 1-bit syntax element DCCOEF_EXTQUANT2 is decoded, and this quantizer step size is used to refined the value of the DC differential. The sign of the DC differential is obtained from the DCSIGN syntax element (section 7.1.4.7).

The following pseudo-code, where QUANT refers to MQUANT, illustrates the DC differential decoding process:

```
DCDifferential = vlc_decode()
if(DCDifferential != 0) {
    if(DCDifferential == ESCAPECODE) {
          if(QUANT == 1)
              DCDifferential = flc_decode(10);
          else if(QUANT == 2)
              DCDifferential = flc_decode(9);
           else // QUANT is > 2
              DCDifferential = flc_decode(8);
         }
   else {   // DCDifferential is not ESCAPECODE
        if(QUANT == 1)
            DCDifferential = DCDifferential*4 + flc_decode(2) – 3;
        else if(QUANT == 2)
            DCDifferential = DCDifferential*2 + flc_decode(1) – 1;
         }
    DCSign = flc_decode(1)
    if (DCSign == 1)
     DCDifferential = -DCDifferential
}
```

Figure 32: DC Differential Decoding Pseudo-code

8.1.1.8 DC Predictor

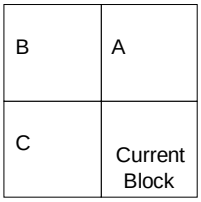


Figure 33: DC predictor candidates

The quantized DC value for the current block is obtained by adding the DC predictor to the DC differential obtained as described in section 8.1.1.7. In the simple and main profiles, the DC predictor is obtained directly from the DC coefficients of one of the previously decoded adjacent blocks. In the advanced profile, there is an additional coefficient scaling step if the macroblocks quantizers of the neighbouring 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 8.1.1.15, and these scaled DC coefficients are used for computing the prediction direction, as well as the actual prediction. Figure 33 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.

For these cases the DC predictor is set to either 0 (Rule A) or the following (Rule B):

```
DCPredictor = (1024 + (DCStepSize >> 1)) / DCStepSize
```

Rule B is used only for Intra (and BI) frames coded in the simple/main profile with overlap filtering turned off. All other cases use the value of 0 for the default DC predictor. Since DQUANT is not enabled for I and BI frames in the simple and main profiles, Rule B is never applied with DQUANT.

Refer to section 8.1.1.9 for a description of how to compute DCStepSize.

A prediction direction is formed based on the values of A, B and C and either the A or C predictor is chosen. The prediction direction is calculated for Rule A and Rule B as follows:

Rule A

If there are no adjacent intra blocks, the DC predictor is set to zero. If there is only one adjacent block, this block is used for prediction and its DC value used as the DC predictor (appropriately scaled as specified in section 8.1.1.15 if necessary). If only the left and top blocks are intra, and top-left block (block B in Figure 33) is inter coded, then the left block C is used as the predictor. If all three neighbors A, B and C are available, then the procedure of Figure 34 is applied.

Rule B

When Rule B is used, the DC component of unavailable neighbors is assumed to be DCPredictor (specified in the above equation). If the absolute value of (B - A) is less than or equal to the absolute value of (B – C), then the prediction is made from the left (C is the predictor). Otherwise the prediction is made from the top (A is the predictor). In pseudo-code, the process is as follows:

```
if (|B – A| <= |B – C|)
{
    PredDirection = left;
    DCPredictor = C;
```

```
}
else
{
    PredDirection = top;
    DCPredictor = A;
}
```

Figure 34: Prediction selection pseudo-code

The quantized DC coefficient is then calculated by adding the DC differential and the DC predictor as follows:

```
DCCoeffQ = DCPredictor + DCDifferential
```

8.1.1.9 DC Inverse-quantization

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

```
DCCoefficient = DCCoeffQ * DCStepSize
```

The value of DCStepSize is based on the value of MQUANT (obtained in the picture header 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
```

8.1.1.10 AC Coefficient Bitstream Decode

The non-zero quantized AC coefficients are 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.

Huffman table (HuffTable): 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 35. Used in escape coding mode 2.

Last delta run table (LastDeltaRunTable): The table of delta *run* values indexed by the *level* value as illustrated in pseudo-code of Figure 35. Used in escape coding mode 2.

Not-last delta level table (NotLastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 35. Used in escape coding mode 1.

Last delta level table (LastDeltaLevelTable): The table of delta *level* values indexed by the *run* value as illustrated in pseudo-code of Figure 35. Used in escape coding mode 1.

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

Constants

Start index of last coefficient (StartIndexOfLast): The HuffTable 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 correspond 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 HuffTable. See the description above and the pseudo-code of Figure 35 for a description of how this constant is used.

The following pseudo-code illustrates how the tables and constants are used to decode a run, level and last-flag triplet.

```
last_flag = 0;
index = vlc_decode();    ## Use HuffTable 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 54    to decode ESCMODE syntax element
      if (escape_mode == mode1)
      {
            index = vlc_decode();    ## Use HuffTable to 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 HuffTable to decode VLC codeword (ACCOEF2)
        run = RunTable[index];
        level = LevelTable[index];
        if (index >= StartIndexOfLast)
            last_flag = 1;
        if (last_flag == 0)
            run = run + NotLastDeltaRunTable[level];
        else
            run = run + LastDeltaRunTable[level];
        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 55 or Table 56 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 35: Coefficient decode pseudo-code

The process illustrated in Figure 35 above for decoding the non-zero AC is 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 is used. For Cb and Cr blocks one of the four inter coding sets is used. Section 11.8 lists the tables that make up each coding set. The particular set used to decode a block is signaled by an index value in either the picture header. The following two tables shows how the index corresponds to the coding set for Y and Cb/Cr blocks. As the tables show, if the value of PQINDEX (see section 7.1.1.15) is less than or equal to 7, then the high rate coding set is used for index 0. If PQINDEX is greater than 7, then the low motion coding set is used for index 0.

Table 60: Coding Set Correspondence for PQINDEX <= 7

Y blocks	Cb and Cr blocks

Index	Table	Index	Table
0	High Rate Intra	0	HighRate Inter
1	High Motion Intra	1	High Motion Inter
2	Mid Rate Intra	2	Mid Rate Inter

Table 61: Coding Set Correspondence for PQINDEX > 7

Y blocks		Cb and Cr blocks	
Index	Table	Index	Table
0	Low Motion Intra	0	Low Motion Inter
1	High Motion Intra	1	High Motion Inter
2	Mid Rate Intra	2	Mid Rate Inter

The value decoded from the TRANSACFRM2 syntax element is used as the coding set index for Y blocks and the value decoded from the TRANSACFRM syntax element is used as the coding set index for Cb and Cr blocks.

8.1.1.11 AC Run-level Decode

The ordered run and level pairs obtained as described in section 8.1.1.10 are used to form an array of 63 elements by employing a run-level decode process as illustrated in the pseudo-code of Figure 36.

```
array[63] = {0};       ## 63 element array initialized to zero.
curr_position = 0;
do {
    decode_symbol(&run, &level, &last_flag);    ## decode the bitstream as described in Figure 35 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 36: Run-level decode pseudo-code

8.1.1.12 Zig-zag Scan of AC Coefficients

Decoding the run-level pairs as described in section 8.1.1.11 produces a one-dimensional array of 63 quantized AC coefficients. The elements in the array are scanned out into an 8x8 two-dimension array in preparation for the Inverse Transform. Figure 37 shows the elements in an 8x8 array labeled in raster scan order from 0 to 63. The DC coefficient is in position 0. A mapping array is used to scan out the remaining 63 AC coefficients in the one-dimensional array to the 8x8 array. As an example,

Figure 39 shows the mapping array used to produce the one-dimensional to two-dimensional scan out pattern shown in Figure 38.

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 37: 8x8 array with positions labeled

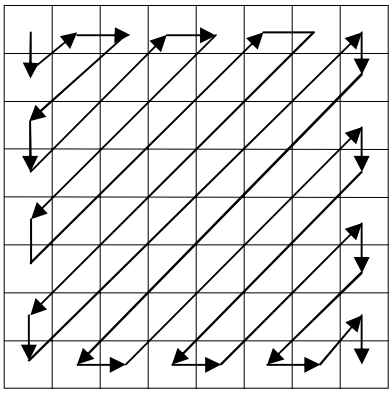


Figure 38: Example zig-zag scanning pattern

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 39: Zig-zag scan mapping array

One of three scan arrays is used to scan out the one-dimensional array depending on the AC prediction status for the block (see section 8.1.1.13 for description of AC prediction). Table 62 shows how the AC prediction status determines which scan array is used.

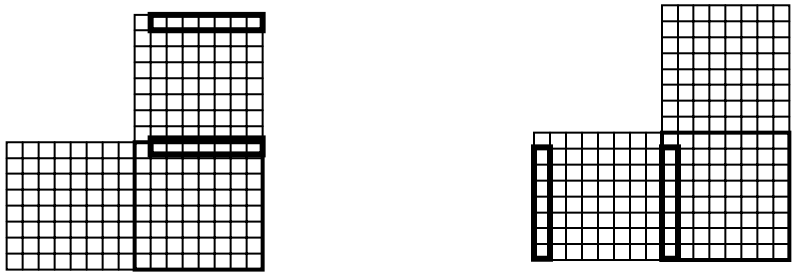
Table 62: **Scan Array Selection**

AC Prediction	**AC Scan Array**
Top prediction	Horizontal scan
Left prediction	Vertical scan
No prediction	Normal scan

The tables for the horizontal, vertical and normal scan arrays are listed in section 11.9.1.

8.1.1.13 AC Prediction

If the ACPRED syntax element in the macroblock layer specifies that AC prediction is used for the blocks, then the top row or left column of AC coefficients in the decoded block are treated as differential values from the coefficients in the corresponding row or column in a predicted block. The predictor block is either the block immediately above or to the left of the current block. For each block, the direction chosen for the DC predictor is used for the AC predictor (see section 8.1.1.8). Figure 40 shows that for top prediction the first row of AC coefficients in the block immediately above is 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 is used as the predictor for the first column of AC coefficients in the current block. In the simple profile, these AC coefficients are used directly for prediction. In the main and advanced profiles, there is an additional coefficient scaling step if the macroblocks 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 8.1.1.15, and these scaled AC coefficients are used for prediction.



Top Prediction Left Prediction

Figure 40: AC prediction candidates

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

The AC coefficients in the predicted row or column are added to the corresponding decoded AC coefficients in the current block to produce the fully reconstructed quantized Transform coefficient block.

8.1.1.14 Inverse AC Coefficient Quantization

Depending on whether the uniform or nonuniform quantizer is used (see section 7.1.1.15 and 7.1.1.17), the non-zero quantized AC coefficients reconstructed as described in the sections above are inverse quantized according to the following formula:

dequant_coeff = *quant_coeff* * *double_quant* (if uniform quantizer), or

dequant_coeff = *quant_coeff* * *double_quant* + sign(*quant_coeff*) * *quant_scale* (if nonuniform quantizer)

where:

quant_coeff is the quantized coefficient

dequant_coeff is the inverse quantized coefficient

double_quant = 2 * MQUANT + HalfStep

quant_scale = MQUANT

MQUANT is encoded in the macroblock layer, or may be derived from the picture layer PQUANT, as described in sections 7.1.1.15, 7.1.1.29, 7.1.3.6 and 7.1.3.7. HalfStep is encoded in the picture layer as described in section 7.1.1.16.

8.1.1.15 Coefficient Scaling

For DC and AC prediction, in the advanced profile, the coefficients in the predicted blocks are 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 MQUANT in the predictor block

$STEP_c$ is the MQUANT in the current block

$DQScale$ is an integer look up table with inputs from 1 to 31.

Table 63: 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

8.1.1.16 Inverse TRANSFORM

After reconstruction of the TRANSFORM coefficients, the resulting 8×8 blocks are processed by a separable two-dimensional inverse transform of size 8 by 8. The inverse transform output has a dynamic range of 10 bits. See section 8.8 regarding INVERSETRANSFORM conformance.

Subsequent to the inverse transform, the process of overlap smoothing is carried out if signaled. This is covered in Section 7.5. Finally, the constant value of 128 is added to the reconstructed and possibly overlap smoothed intra block. This result is clamped to the range [0 255] and forms the reconstruction prior to loop filtering.

For simple and main profile I frames, the constant 128 is not added prior to clamping to [0 255].

8.2 Progressive BI Frame Decoding

When B frames are used (in main and advanced profiles only), we code a special type of frame that is in some ways a hybrid of I and B frames. The syntax of BI frames is almost identical to that of I, but they are usually coded at higher QP's and can never be used as an anchor or reference frame to predict other frames.

We will only mention the few differences between BI and I frames in this section. The remainder of BI decoding is identical to I decoding.

8.2.1 BFRACTION following picture type (main profile only)

The BFRACTION syntax element (section 7.1.1.10) immediately follows the picture type in a BI frame sent in main profile. The specific code word that indicates that this I frame is to be re-interpreted as BI- is the BFRACTION codeword: “1111111”.

8.2.2 No picture resolution index (RESPIC)

RESPIC is not sent in a BI frame. BI frames, along with B frames are constrained to operate at the same resolution as its neighboring anchor frames.

8.2.3 No range reduction (RANGEREDFRM)

RANGEREDFRM is not sent in a BI frame. BI frames, along with B frames are constrained to operate at the same dynamic range as the neighboring anchor frames.

8.3 Progressive P Frame Decoding

Figure 51 shows the steps required to decode and reconstruct blocks in P frames. The following sections describe the process for decoding P pictures.

8.3.1 Skipped P Frames

In the main and simple profiles, frame skipping is to be signaled through additional means. If ASF Transport Layer is used for this purpose, frame skipping may be signaled via the total length of data comprising a compressed frame. As a coded frame will always contain 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). Note that the decoder does not decode the 8 bits used for frameskipping as they do not represent an actual codeword.

In the advanced profile, a skipped frame is signaled by the PTYPE syntax element in the picture header. If a frame is signaled as skipped then it is treated as if it were a P frame which was identical to the reference frame. Therefore, the reconstruction of the skipped frame may be treated conceptually as copying the reference frame.

8.3.2 Out-of-bounds Reference Pixels

The previously decoded frame is 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 are the replicated values of the edge pixel. Figure 41 illustrates pixel replication for the upper-left corner of the frame. Note that in advanced profile, “frame edge”, “frame corner” and “outside the boundary” refer to the true frame dimensions, not the dimensions right or top/bottom justified to the edge of the macroblock. In other words, the right and bottom pixels that are repeated to infinity for a 200 x 300 image begin at column 304 and row 208 for the simple and main profiles. However, for the advanced profile, these begin respectively at column 300 and row 200.

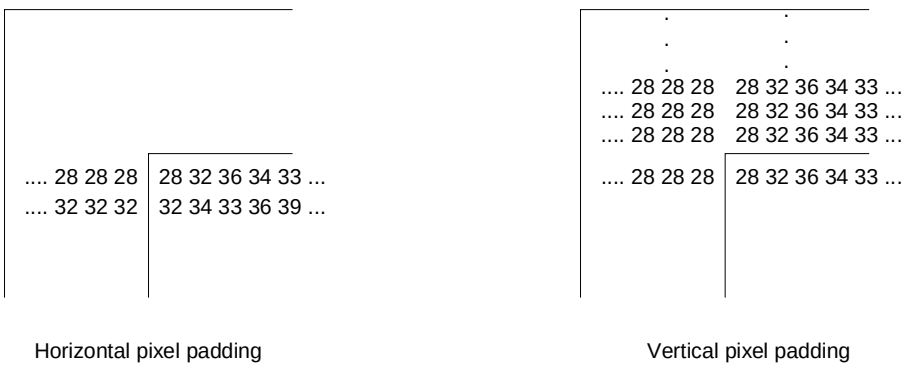


Figure 41: Horizontal and vertical pixel replication for out-of-bounds reference

8.3.3 P Picture Types

P pictures may 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 is 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 encoded as a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luminance blocks has 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 14 shows the elements that make up the progressive P picture layer header. Some of the elements are self-explanatory. The following sections provide extra detail for some of the elements.

8.3.4.1 Picture-level Quantizer Scale

The frame level quantizer scale PQUANT is decoded from the 5-bit picture layer syntax element PQINDEX as described in section 7.1.1.15. PQUANT specifies the frame level quantizer scale (a value between 1 and 31) for the macroblocks in the current picture. When the seq. header DQUANT = 0, then PQUANT is used as the quantization step size for every macroblock in the current picture. When DQUANT != 0, then PQUANT is used as described in section 7.1.1.29. The PQINDEX syntax element also specifies whether the uniform or nonuniform quantizer is used for all macroblocks in the frame.

8.3.4.2 Picture Resolution Index

The RESPIC syntax element in P pictures, in simple and main profiles, 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 speficies a resolution index of 1 (full vertical resolution, half horizontal resolution), then all subsequent P pictures will specify the same resolution until the next I picture.

All P pictures that are coded at less than full resolution shall be upsampled to full resolution prior to display. Since this upsampling process is outside the reconstruction loop the implementer is free to use whatever upsampling process he or she choses. The spatial alignment of video samples of the downsampled 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 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, LUMSCALE and LUMSHIFT syntax elements follow in the picture layer. In this case, MVMODE2 signals the motion vector mode and LUMSCALE and LUMSHIFT are 6-bit values which specify parameters used in the intensity compensation process. Refer to section 8.3.8 for a description of intensity compensation decode.

Table 31 and Table 32 show the codetables used to decode the MVMODE syntax element. Table 31 is used if PQUANT is greater than 12 and Table 32 is used if PQUANT is less than or equal to 12. In a similar fashion, Table 35 and Table 36 are used to decode the MVMODE2 syntax element. Either MVODE or MVMODE2 will signal one of four motion vector modes. If the motion vector mode is mixed MV mode, then the MVTYPEMB syntax element 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 in raster scan order from upper left to lower right. Refer to section 8.7

for a description of the bitplane coding. A value of 0 indicates that the macroblock is coded in 1-MV mode. A value of 1 indicates that the macroblock is coded in 4-MV mode. Refer to section 8.3.5.2 for a description of the motion vector decoding process.

8.3.4.4 Skipped Macroblock Decoding

The P picture layer contains the SKIPMB syntax element which is a bitplane coded syntax element that indicates 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 in raster scan order from upper left to lower right. Refer to section 8.7 for a description of the bitplane coding. A value of 0 indicates that the macroblock is not skipped. A value of 1 indicates that the macroblock is coded as skipped. 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). Note that a skipped macroblock does not contain any prediction error information. Refer to section 8.3.5.2 for a description of how the HYBRIDPRED syntax element is used in the decoding process.

8.3.4.5 Motion Vector Huffman Table

MVTAB is 2-bit syntax element in the picture layer that indicates the Huffman table used to decode the motion vector differentials for the macroblocks in the picture. The Huffman tables are encoded as shown in Table 64. Section 11.10 contains the Motion Vector Differential Huffman tables. Refer to section 8.3.5.2 for a description of the motion vector decode process.

Table 64: Motion vector Huffman table

MVTAB FLC	Huffman table
00	Motion Vector Table 0
01	Motion Vector Table 1
10	Motion Vector Table 2
11	Motion Vector Table 3

8.3.4.6 Coded Block Pattern Huffman Table

CBPTAB is 2-bit syntax element in the picture layer that indicates the Huffman table used to decode the coded block pattern (CBPCY) for the macroblocks in the picture. The Huffman tables are encoded as shown in Table 65. Section 11.6 contains the CBP Huffman tables. See section 8.3.5.2 for a description of how CBPCY is used.

Table 65: CBP Huffman table

CBPTAB FLC	Huffman table
00	CBP Table 0
01	CBP Table 1
10	CBP Table 2
11	CBP Table 3

8.3.4.7 Macroblock-level Quantizer Mode Flag

See section 7.1.3.6.

8.3.4.8 Macroblock-level Transform Type Flag

TTMBF 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 is used for all blocks in the frame. In this case, the transform type is signaled in the TTFRM syntax element that follows. If TTMBF = 0, then the transform type may vary throughout the frame and is signaled at the macroblock or block levels.

8.3.4.9 Frame-level Transform Type

TTFRM is a variable-length syntax element that is present in the picture layer if TTMBF = 1. TTFRM is decoded using Table 41 and signals 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.

8.3.4.10 Frame-level Transform AC Coding Set Index

TRANSACFRM is a variable-length syntax element that is 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 42 is used to decode the TRANSACFRM syntax element.

8.3.4.11 Intra Transform DC Table

The TRANSDCTAB syntax element has the same meaning as the TRANSDCTAB syntax element in I pictures. See section 8.1.1.2 for a description.

8.3.4.12 Range Reduction Frame - P Frame (RANGEREDFRM)

The RANGEREDFRM is only signaled when RANGERED is signaled at the sequence level.

When RANGEREDFRM is signaled for the current P Frame, the current decoded frame shall be scaled up prior to display, similar to I Frame, while keeping the current reconstructed frame intact. Let Y, U, V denote the YCbCr planes of the output frame. The pixels are scaled up according to the following formula:

Y[n] = CLIP ((Y[n] – 128) * 2 + 128);

U[n] = CLIP ((U[n] – 128) * 2 + 128);

V[n] = CLIP ((V[n] – 128) * 2 + 128);

In addition, the previously reconstructed frame shall be scaled up prior to using it for motion compensation if the current frame and previous frame are operating at different range. The process will be applied to 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 the previous reconstructed frame. Let Y, U, V denote the YCbCr planes of the previously reconstructed frame.

- Current frame's RANGEREDFRM is signaled and the previous frame's RANGEREDFRM is not signaled. In this case, the previously reconstructed frame is scaled down as follows:

  Y[n] = ((Y[n] – 128) >> 1) + 128;

  U[n] = ((U[n] – 128) >> 1) + 128;

  V[n] = ((V[n] – 128) >> 1) + 128;

- Current frame's RANGEREDFRM is not signaled and the previous frame's RANGEREDFRM is signaled. In this case, the previous reconstructed frame is scaled as follows:

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

  U[n] = CLIP((U[n] – 128) * 2 + 128);

  V[n] = CLIP((V[n] – 128) * 2 + 128);

8.3.5 Macroblock Layer Decode

8.3.5.1 Macroblock Types

Macroblocks in P pictures may be one of 3 possible types: 1MV, 4MV, and Skipped. The macroblock type is indicated by a combination of picture and macroblock layer syntax elements. The following sections describe each type and how they are signaled.

8.3.5.1.1 1MV Macroblocks

1MV macroblocks may occur in 1-MV and Mixed-MV P pictures. A 1MV macroblock is one where a single MVDATA syntax element 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 also indicates 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.

If the P picture is of type 1MV, then all the macroblocks in the picture are of type 1MV so there is no need to individually signal the macroblock type.

If the P picture is of type Mixed-MV, then the macroblocks in the picture may be of type 1MV or 4MV. In this case the macroblock type (1MV or 4MV) is signaled in the MVTYPEMB syntax element in the picture layer. See section 8.3.4.3 for a description of how the MVTYPEMB syntax element signals the 1MV/4MV macroblock type.

8.3.5.1.2 4MV Macroblocks

4MV macroblocks may only occur in Mixed-MV P pictures. A 4MV macroblock is indicated by signaling that the macroblock is 4-MV in the MVTYPEMB picture layer syntax element. Individual blocks within a 4MV macroblock may be coded as Intra blocks. For the 4 luminance blocks, the Intra/Inter state is signaled by the BLKMVDATA syntax element associated with that block. The CBPCY syntax element that indicates which blocks have BLKMVDATA syntax elements present in the bitstream. See section 8.3.5.2 for a description of how the CBPCY syntax element is used in 4MV macroblocks.

The Inter/Intra state for the chroma blocks is derived from the luminance Inter/Intra states. If 3 or 4 of the luminance blocks are coded as Intra, then the chroma blocks are also coded as Intra.

8.3.5.1.3 Skipped Macroblocks

Skipped macroblocks may occur in 1-MV, and Mixed-MV P pictures. In all cases, a skipped macroblock is signaled by the SKIPMB bitplane syntax element in the picture layer. See section 8.3.4.4 for a description of the SKIPMB syntax element.

8.3.5.2 Macroblock Decoding Process

The following sections describe the macroblock layer decoding process for P picture macroblocks.

Refer to section 8.3.6.2 for a description of the inverse quantization process.

8.3.5.2.1 Decoding Motion Vector Differential

The MVDATA or BLKMVDATA syntax elements encode motion information for the blocks in the macroblock. 1MV macroblocks have a single MVDATA syntax element, and 4MV macroblocks may have between zero and four BLKMVDATA syntax elements (see section 8.3.5.2 for a description of how the CBPCY syntax element is used to encode the number of MVDATA syntax elements in 4MV macroblocks).

Each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly encodes three things: 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 4MV) 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 indicates that the block is actually intra-coded. See the pseudo-code for decoding MV in section 8.3.5.2.1.

The MVDATA or BLKMVDATA syntax element is a variable length Huffman codeword followed by a fixed length codeword. The value of the Huffman codeword determines the size of the fixed length codeword. The MVTAB syntax element in the picture layer specifies the Huffman table used to decode the variable sized codeword.

The following psuedocode illustrates how the motion vector differential, Inter/Intra type and last-flag information are decoded. Note that the motion vector differentials decoded in this pseudocode are modulo differentials. The computation of motion vectors from these differentials is shown in section 8.3.5.4.1.

The values: '**last flag', intra_flag, dmv_x** and **dmv_y** are computed in the following pseudocode. The values are defined as follows:

'last flag': binary flag indicating whether any Transform coefficients are present (1 = coefficients present, 0 = coefficients not present)

intra_flag: binary flag indicating whether the block or macroblock is intra-coded (0 = inter-coded, 1 = intra-coded)

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 7.1.1.16) according to Table 66.

Table 66: 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** used in the following pseudocode 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 1MV or Mixed MV, then **halfpel_flag** = 0 and quarter-pel precision is used. If MVODE specifies the mode as 1MV Half-pel or 1MV Half-pel Bilinear, then **halfpel_flag** = 1 and half-pel precision is used.

The tables **size_table** and **offset_table** are arrays used in the following pseudocode and are 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 Huffman table indicated by MVTAB in the picture layer
index = index + 1
if (index >= 37)
{
      'last flag' = 1
      index = index - 37
}
```

```
else
    'last 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 42: Decoding MV Differential in Progressive Pictures: Pseudo-code

8.3.5.3 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. If a neighbouring block is intra-coded, its motion vector is set to be zero for the purposes of prediction. The following sections describe how the predictors are calculated for macroblocks in 1MV P pictures, and Mixed-MV P pictures.

8.3.5.3.1 Motion Vector Predictors In 1MV P Pictures

Figure 43 shows the three motion vectors used to compute the predictor for the current macroblock. As the figure shows, the predictor is 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 is taken from the top-left macroblock instead of the top-right.

For the special case where the frame is one macroblock wide, the predictor is always Predictor A (the top predictor).

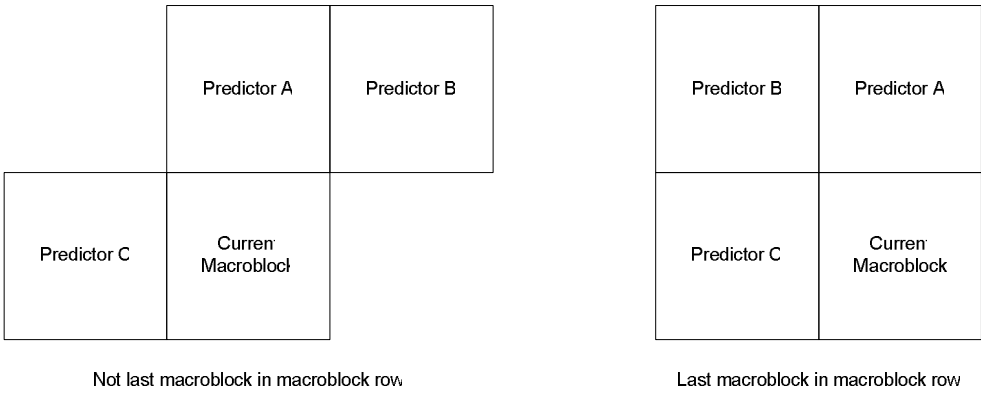


Figure 43: Candidate Motion Vector Predictors in 1MV P Pictures

8.3.5.3.2 Motion Vector Predictors In Mixed-MV P Pictures

Figure 44 and Figure 45 show the 3 candidate motion vectors for 1MV and 4MV 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 is always Predictor A (the top predictor).

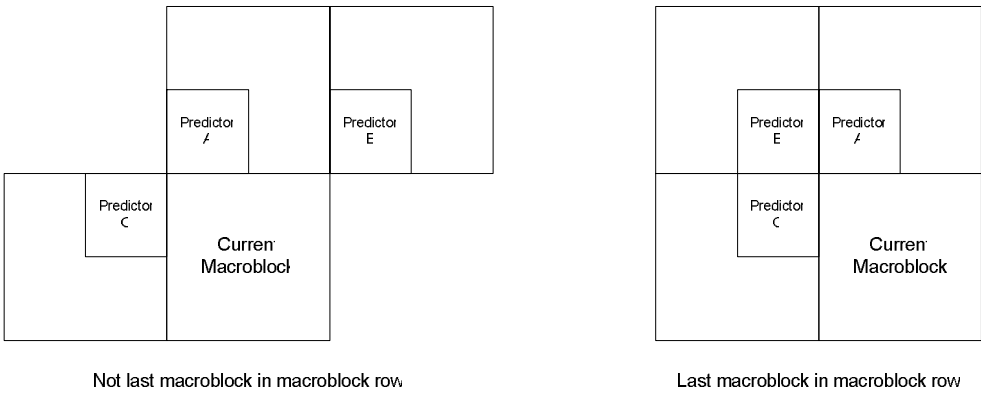


Figure 44: Candidate Motion Vectors for 1MV Macroblocks in Mixed-MV P Pictures

Figure 44 shows the candidate motion vectors for 1MV macroblocks. The neighboring macroblocks may 1MV or 4 MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4MV (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 1MV macroblocks, then the motion vector predictors shown in Figure 44 are 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 is from block 3 of the top-left macroblock instead of from block 2 in the top-right macroblock as is the case otherwise.

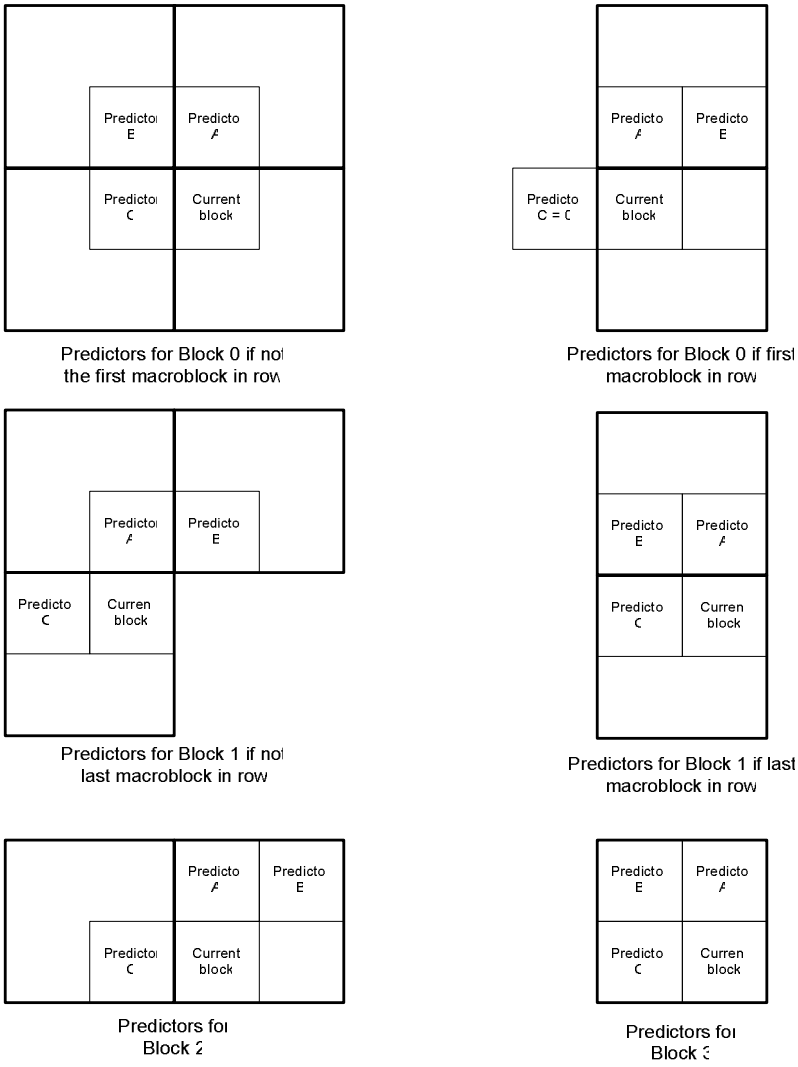


Figure 45: Candidate Motion Vectors for 4MV Macroblocks in Mixed-MV P Pictures

Figure 45 shows the predictors for each of the 4 luminance blocks in a 4MV 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 is 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 is 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 are set equal to 0.

8.3.5.3.3 Calculating the Preliminary Motion Vector Predictor

Given the 3 motion vector predictor candidates, the following pseudocode illustrates the process for calculating the preliminary motion vector predictors. These preliminary predictors are used in the next section to compute the actual motion vector predictors.

```
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) {
            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_x;
} else {
      predictor_pre_x = predictor_pre_y = 0;
}
```

Figure 46: Calculating MV Predictior: Pseudo-code

See section 4.9 for the definition of median3.

Note that a predictor candidate is considered to be out of bounds, if either the corresponding block is outside the frame boundary, or if the corresponding neighbouring block is part of a different slice.

After the predicted motion vectors are computed, a “pull-back’ operation is performed, if necessary, on its values. The pull-back operation consists of follows: The predicted motion vector is checked to see if the block/macroblock referenced by it lies outside of the reference frame. If yes, the predictor motion vectors (MV) are clipped such that at least one line of the reference frame is inside the block/macroblock referenced by the predictor. This goal is achieved by adjusting the horizontal component MVx (if needed), followed by adjusting the vertical component MVy (if needed). As an example, consider a block N pixels wide and M pixels high.

Horizontal Adjustment: If the top left pixel of the block (pointed by the predicted MV) is to the left of $(N-1)^{th}$ column in the reference frame, then adjust predicted MVx so that the top left point lies along the $(N-1)^{th}$ column. Similarly, if the top left point of the block (pointed by the predicted MV) is to the right of (picture_width – 1)th column, then adjust predicted MVx so that the top left point lies along the (picture_width – 1)th column.

Vertical Adjustment: If the top left point of the block (pointed by the predicted MV) is to the top of $–(M-1)^{th}$ row in the reference frame, then adjust predicted MVy so that the top left point of the block lies along to $–(M-1)^{th}$ row. Similarly if the top left point of the block is to the bottom of (picture_height – 1)th row, then adjust predicted MVy so that the top left point lies along the (picture_height – 1)th row.

As a result of this pull-back operation, the predicted motion vectors obey the following constraints:

1. For 16x16 mv: restrict the top-left point of the 16x16 area pointed to by the predicted MV to be -15 to picture width -1.
2. For 8x8 mv: restrict the top-left point of the 8x8 area pointed to by the predicted MV to be -7 to picture width – 1.

8.3.5.3.4 Hybrid Motion Vector Prediction

If the P picture is 1MV or Mixed-MV, then the motion predictor calculated in the previous section is tested relative to the A and C predictors to see if the predictor is explicitly coded in the bitstream. If so, then a bit is decoded that indicates 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 following pseudocode illustrates hybrid motion vector prediction decoding.

The variables are defined as follows in the pseudocode:

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 hybdrid motion vector prediction
predictor_post_y: The vertical motion vector predictor after checking for hybdrid 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) == 0)       // 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) == 0)
            {
                // 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 47: Hybrid Motion Vector: Preliminary Prediction

8.3.5.3.5 Motion Vector Predictors in Skipped Macroblocks

If a macroblock is coded as skipped, then the predicted motion vector computed as described above is used as the motion vector for the block, macroblock or syntax element. The block, syntax element or macroblock referenced by the motion vector is used as the current block or macroblock in the current picture. A single bit may be present in the macroblock layer indicating which of the predictor candidates to use. A macroblock with 4 motion vectors may have upto 4 hybrid motion predictor (HYBRIDPRED) syntax elements, i.e. upto 4 bits.

8.3.5.4 Reconstructing Motion Vectors

The following sections describe how to reconstruct the luminance and chroma motion vectors for 1MV and 4MV macroblocks.

8.3.5.4.1 Luminance Motion Vector Reconstruction

In all cases (1MV and 4MV macroblocks) the luminance motion vector is reconstructed by adding the differential to the predictor as follows:

mv_x = (dmv_x + predictor_x) smod range_x

mv_y = (dmv_y + predictor_y) smod range_y

The modulus operation “smod” is a signed modulus, defined as follows:

A smod b = ((A + b) % 2 b) - b

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 66.

Following are the notes about luminance motion vectors in 1MV and 4MV macroblocks.

1MV Macroblock Notes

In 1MV macroblocks there will be a single motion vector for the 4 blocks that make up the luminance component of the macroblock.

If dmv_x decodes to indicate that the macroblock is Intra-coded (as described in the section “Decoding Motion Vector Differential” above), then no motion vectors are associated with the 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_x and mv_y = predictor_y).

4MV Macroblock Notes

Each of the Inter-coded luminance blocks in a macroblock will have its own motion vector. Therefore there will be between 0 and 4 luminance motion vectors in each 4MV macroblock.

A non-coded block in 4MV macroblocks may occur in one of two ways: 1) if the SKIPMB syntax element in the picture layer indicates that the macroblock is skipped and the MVTYPEMB syntax element in the picture layer indicates that the macroblock is 4MV. All blocks in the macroblock are skipped in this case, or 2) if the CBPCY syntax element (described in the next section) in the macroblock indicates that the block is non-coded. If a block is not coded, then dmv_x = 0 and dmv_y (mv_x = predictor_x and mv_y = predictor_y).

8.3.5.4.2 Chroma Motion Vector Reconstruction

The chroma motion vectors are derived from the luminance motion vectors. Also, for 4MV macroblocks, the decision of whether to code the chroma blocks as Inter or Intra is made based on the status of the luminance blocks or syntax elements. The following sections describe how to reconstruct the chroma motion vectors for 1MV and 4MV macroblocks. The chroma vectors are reconstructed in two steps.

As a first step, the nominal chroma motion vector is obtained by combining and scaling the luminance 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, a sequence level 1-bit syntax element FASTUVMC syntax element is used to determine if further rounding of chroma motion vectors is necessary. The purpose of this mode is speed optimization of the decoder. If FASTUVMC = 0, no rounding is performed in the second stage. If FASTUVMC = 1, the chroma motion vectors that are at quarter pel offsets will be rounded to the nearest half or full pel positions as described in Section 8.3.5.4.5.

Only bilinear filtering will be used for all chroma interpolation

In the sections below cmv_x and cmv_y denote the chroma motion vector components and lmv_x and lmv_y denote the luminance motion vector components.

8.3.5.4.3 First-stage Chroma Motion Vector Reconstruction - 1MV Chroma Motion Vector Case:

If a MV macroblock, the chroma motion vectors are derived from the luminance 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[1mv_x & 3]) >> 1
cmv_y = (lmv_y  + s_RndTbl[1mv_y & 3]) >> 1
```

8.3.5.4.4 First-stage Chroma Motion Vector Reconstruction - 4MV Chroma Motion Vector Case:

The following pseudocode illustrates how the chroma motion vectors are derived from the motion information in the 4 luminance blocks in 4MV macroblocks. In this section, ix and iy are temporary variables.

```
if (all 4 luminance 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 luminance 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 luminance 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
    Chroma 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 48: Chroma MV Reconstruction for Progressive: Pseudo-Code

See section 4.9 for the definition of median3 and median4.

8.3.5.4.5 Second Stage Chroma Rounding

If the sequence level bit FASTUVMC = 1, then a second level of rounding is done on the chroma 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 chroma motion vector in units of quarter pels. % represents the modulus (or remainder) operation, which is defined thus: (x % a) = -(-x % a), i.e. the modulus of a negative number is equal to the negative of the modulus of the corresponding positive number. Thus, when cmv_x (or cmv_y) is divisible by 4, there is an integer 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 are being disallowed by rounding the chroma motion vector to the nearest integer or half pel position towards zero (half pel positions are left unaltered).

This forces the chroma co-ordinates to be remapped to integer and half pel positions. Second stage rounding is not performed if FASTUVMC = 0.

8.3.5.5 Coded Block Pattern

Figure 22 shows the position of the CBPCY syntax element within the P picture macroblock layer. The CBPCY syntax element is a variable-length code that decodes to a 6-bit syntax element.

The Huffman codetable used to decode CBPCY is specified by the CBPTAB syntax element in the picture layer. See section 8.3.4.6 for a description of the CBPTAB syntax element.

The CBPCY syntax element is used differently depending on whether the macroblock is 1MV or 4MV. The following sections describe how CBPCY is used in each macroblock type.

8.3.5.5.1 CBPCY in 1MV Macroblocks

The CBPCY syntax element is present in the 1MV macroblock layer if:

1) The MVDATA syntax element indicates that the macroblock is Intra-coded or,

2) The MVDATA syntax element indicates the macroblock is inter-coded, and that at least one block contains coefficient information. This is indicated by the ‘last’ 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 decodes to a 6-bit syntax element indicating which of the blocks contain at least one non-zero coefficient. Figure 49 shows how the CBPCY bitsyntax element corresponds to the block numbers.

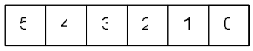


Figure 49: Bit-position/block correspondence for CBPCY

A ‘1’ in one of the positions indicates that the corresponding block has at least one non-zero AC coefficient if the macroblock is Intra-coded or at least one non-zero DC or AC coefficient if the macroblock is Inter-coded.

A ‘0’ in one of the positions indicates that the corresponding block does not contain any non-zero AC coefficients if the macroblock is Intra-coded or any non-zero DC or AC coefficients if the macroblock is Inter-coded.

8.3.5.5.2 CBPCY in 4MV Macroblocks

The CBPCY syntax element is always present in the 4MV macroblock layer. The CBPCY bit positions for the luminance blocks (bits 0-3) have a slightly different meaning than the bit positions for chroma blocks (bits 4 and 5).

For the luminance blocks:

A ‘0’ indicates that the corresponding block does not contain motion vector information or any non-zero coefficients. In this case, the BLKMVDATA syntax element is not present for that block and the predicted motion vector is used as the motion vector and there is no residual data. If the motion vector predictors indicate that hybrid motion vector prediction is used, then a single bit is present indicating the motion vector predictor candidate to use. Refer to section 8.3.5.2 for a description of computing the motion vector predictor.

A ‘1’ indicates that the BLKMVDATA syntax element is present for the block. The BLKMVDATA syntax element indicates whether the block is Inter or Intra-coded and whether there is coefficient data for the block. If it

is Inter coded, the BLKMVDATA syntax element also contains the motion vector differential. If the 'last flag' decoded from BLKMVDATA (described in section 8.3.5.2) decodes to 0, then no AC coefficient information is present if the block is Intra-coded or no DC or AC coefficient is present if the block is Inter-coded. If 'last flag' decodes to 1, then there is 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.

For the chroma blocks:

A '0' indicates that the block does 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' indicates that the corresponding block has 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 is present only in Inter macroblocks. As described in section 7.1.3.11 TTMB encodes 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 is the same for all blocks in the macroblock. If the transform type is 8x4 or 4x8, then a subblock pattern is also decoded from the TTMB syntax element. In this case, the subblock pattern applies to the first coded block in the macroblock. If the transform type is 4x4, then the subblock pattern in encoded in the SUBBLKPAT syntax element at the block level. If the transform type is 8x4 or 4x8, then the subblock patterns for all the blocks after the first one are coded 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 is applied to the first coded block in the macroblock and the TTBLK syntax element is not present for the first coded block. For the remaining coded blocks, the TTBLK syntax element indicates the transform type for that block. If TTMB syntax element indicates that the first transform type is 8x4 or 4x8, then a subblock pattern is also decoded from the TTMB syntax element. In this case, the subblock pattern applies to the first coded block in the macroblock.

8.3.6 Block Layer Decode

8.3.6.1 Intra Coded Block Decode

The process for decoding Intra blocks in P pictures is similar to the process for decoding Intra blocks in I pictures as described in section 0 with the following differences.

8.3.6.1.1 Coefficient Scaling

The process for coefficient scaling is same as described in section 8.1.1.15.

8.3.6.1.2 AC Prediction in Intra blocks in 4MV.

Refer to section 8.1.1.13 for a description of AC prediction. AC prediction in Intra-coded blocks within 4MV macroblocks is similar. The following sections describe how AC prediction is performed in 4MV macroblocks.

If the top predictor is selected, then the top row of AC coefficients from the block above the current block are 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 are used as the predictors for the left column of AC coefficients from the current block. The pseudocode below illustrates the process for deciding the AC predictors in Intra blocks in 4MV macroblocks.

In the pseudocode, predictorA is the block immediately above the current block, predictorC is the block 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 pseudocode is that the variable *use_ac_prediction* determines whether AC prediction is used and *prediction_direction* determines which block is used as the predictor.

Note that in the following pseudocode, the coefficients in the predictor blocks are scaled if the macroblock quantizer scales are different. The previous section describes the scaling operation.

```
use_ac_prediction = FALSE
is_nonzero_predictor = FALSE
if ((predictorA is Intra) && (predictorC is Intra))
{
    Is_nonzero_predicor = 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 = UP
     }
     else
        prediction_direction = LEFT
}
else if ((predictorA is Intra) || (predictorC is Intra))
{
     is_nonzero_predictor = TRUE
     if (predictorA is Intra)
        prediction_direction = UP
     else
        prediction_direction = LEFT
}
```

Figure 50: Calculating DC Predictor Direction: Pseudo-Code

After all the upto six predictors are computed (for the upto six intra blocks in the 4MV macroblock), and only when at least one of the blocks has the flag is_nonzeropredictor set to the value TRUE, the one bit element defining ACPRED is read.

```
if (get_bits(1) == 0)          // ACPRED syntax element
    use_ac_prediction = TRUE
```

8.3.6.1.3 AC Prediction in Intra blocks in 1MV macroblocks

AC prediction in Intra blocks within 1MV macroblocks is the same as Intra blocks in I pictures as described in section 8.1.1.13. The exception is if the top predictor block and left predictor block are not Intra-coded, then AC prediction is not 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 is used as the predictor. If both are Intra-coded, then the method described in section 8.1.1.13 is used. In this case, if the top-left block is not Intra, then the DC value is assumed to be 0.

8.3.6.1.4 Zig-zag Scan

The zig-zag scan order used to scan the run-length decoded Transform coefficients into the 8x8 array is the same as that used for the 8x8 Inter block as described in section 8.3.6.2. This differs from Intra blocks in I pictures which use one of 3 zig-zag scans depending on the prediction direction.

8.3.6.1.5 Coding Sets

If the coding set used to decode the AC coefficients is signaled at the frame level, then the TRANSACFRM syntax element is used to specify the coding set index used for decoding the Y and Cb/Cr AC coefficients (see section 8.1.1.10 for a description of the AC coding sets). The index decoded from the TRANSACFRM syntax element is used to select the intra coding set used to decode the Y blocks and is used to select the inter coding set used to decode the Cb/Cr 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 specifies the index for the intra coding set. The P picture header does not contain the TRANSACFRM2 syntax element. The correspondence between the coding set index and the coding set depends on the value of PQINDEX. Tables Table 67 and Table 68 below show the correspondence for PQINDEX <= 7 and PQINDEX > 7. Section 11.8 contains the table information.

Table 67: Index/Coding Set Correspondence for PQINDEX <= 7

Y blocks		Cb and Cr blocks
Index	Table	Table
0	High Rate Intra	High Rate Inter
1	High Motion Intra	High Motion Inter
2	Mid Rate Intra	Mid Rate Inter

Table 68: Index/Coding Set Correspondence for PQINDEX > 7

Y blocks		Cb and Cr blocks
Index	Table	Table
0	Low Motion Intra	Low Motion Inter
1	High Motion Intra	High Motion Inter
2	Mid Rate Intra	Mid Rate Inter

8.3.6.2 Inter Coded Block Decode

Figure 51 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, 2) sub-block pattern decode, 3) coefficient decode, 4) inverse Transform, 5) obtain predicted block and 6) motion compensation (add predicted and error blocks). The following sections describe these steps.

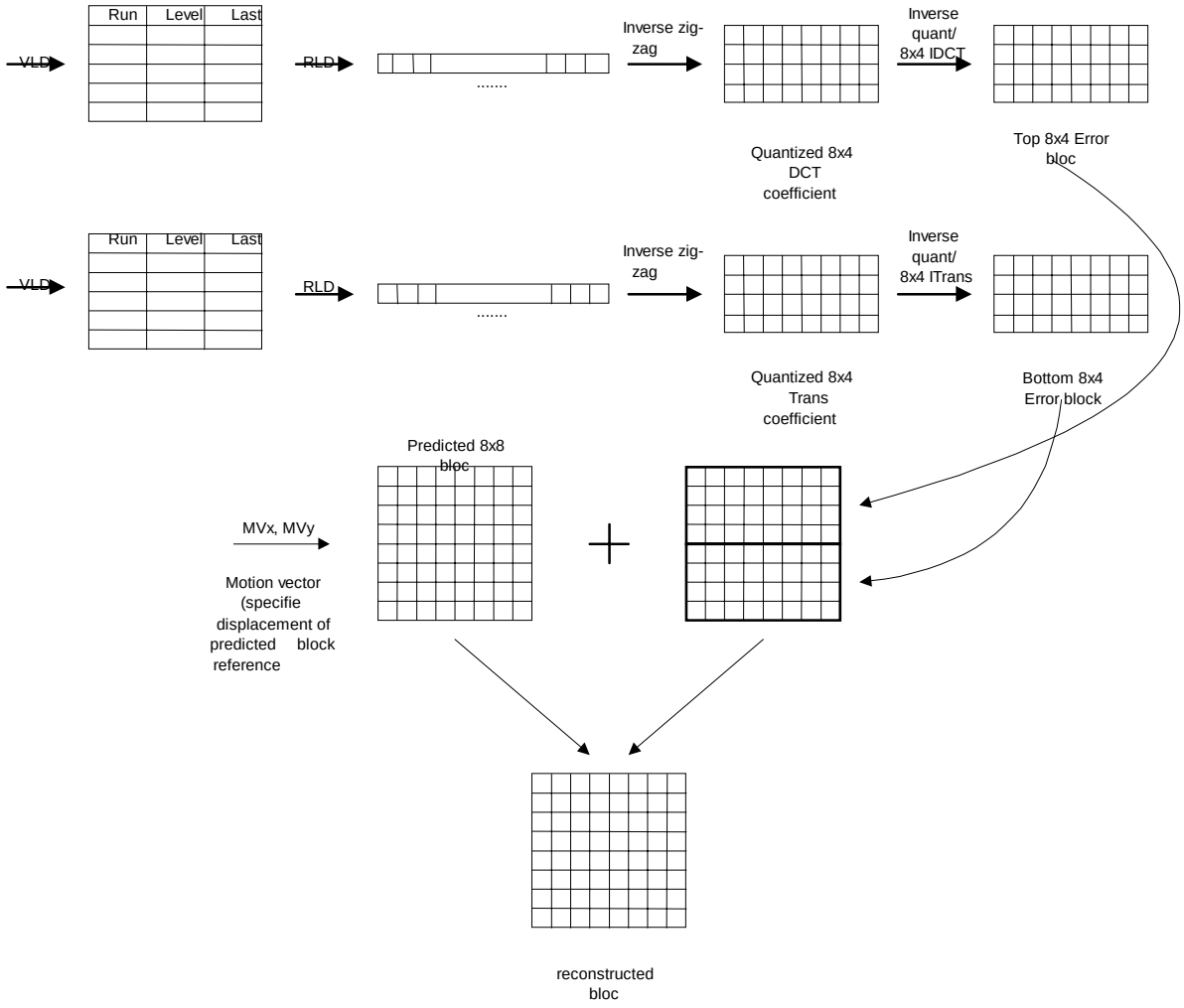


Figure 51: Inter block reconstruction

8.3.6.2.1 Transform Type Selection

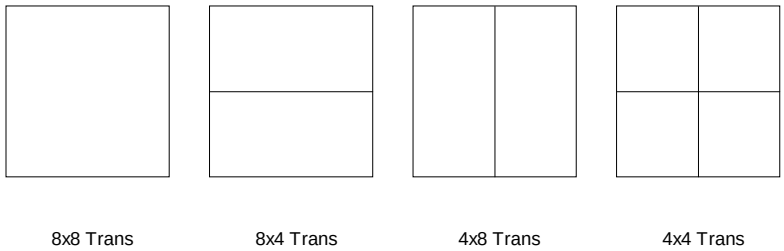


Figure 52: Transform Types

If variable-sized transform coding is enabled (signaled by the sequence-level syntax element VSTRANSFORM = 1 as described in section 6.1.14), then the 8x8 error block may be transformed using one 8x8 Transform, or as shown in Figure 52, 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 4 4x4 Transforms. The transform type is signaled at the picture, macroblock or block level. As shown in Tables Table 43,

TTMB VLC	Transform	Signal Level	Subblock

	Type		Pattern
11	8x8	Block	NA
101110	8x4	Block	Bottom
1011111	8x4	Block	Top
00	8x4	Block	Both
10110	4x8	Block	Right
10101	4x8	Block	Left
01	4x8	Block	Both
100	4x4	Block	NA
10100	8x8	Macroblock	NA
1011110001	8x4	Macroblock	Bottom
101111001	8x4	Macroblock	Top
101111011	8x4	Macroblock	Both
101111000000	4x8	Macroblock	Right
101111000001	4x8	Macroblock	Left
10111100001	4x8	Macroblock	Both
101111010	4x4	Macroblock	NA

Table 44: Medium Rate (5 <= PQUANT < 13) TTMB VLC Table

and

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

001011	8x4	Macroblock	Both
001001	4x8	Macroblock	Right
00100001	4x8	Macroblock	Left
0010101	4x8	Macroblock	Both
00100000	4x4	Macroblock	NA

Table 45 if TTMB indicates that the signal level is Block, then the transform type is signaled at the block level. If the transform type is specified at the block level, then the TTBLK syntax element is present within the bitstream as shown in Figure 25. This syntax element indicates the transform type used for the block. Tables Table 47Table 48Table 49 show the code tables used to encode the transform types if block mode signaling is used.

If variable-sized transform coding is not enabled, then the 8x8 Transform is 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 receive information about which of the subblocks have non-zero coefficients. For 8x4 and 4x8 transform types, the subblock pattern is decoded as part of the TTMB or TTBLK syntax element. If the transform type is 4x4, then the SUBBLKPAT syntax element is present in the bitstream as shown in Figure 25. Section 7.1.4.2 describes the SUBBLKPAT syntax element.

If the subblock pattern indicates that no non-zero coefficients are present for the subblock, then no other information for that subblock is present in the bitstream. For the 8x4 transform type, the data for the top subblock (if present) is coded first followed by the bottom subblock. For the 4x8 transform type, the data for the left subblock (if present) is coded first followed by the right subblock. For the 4x4 transform type, the data for the upper left subblock is 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 Tranform 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 8.1.1.10 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 are decoded using the same method.

2) Unlike the decoding process for intra blocks in I pictures (described in section 8.1.1.10) where the Y block coefficients are decoded using one of the three intra coding sets and the Cb and Cr block coefficients are decoded using one of the three inter coding sets, the Y and Cb/Cr inter blocks all use the same inter coding set.

3) The correspondence between the coding set index and the coding set depends on the value of PQINDEX. The following tables show the correspondence for PQINDEX <= 6 and PQINDEX > 6.

Table 69: Index/Coding Set Correspondence for PQINDEX <= 6

Y, Cb and Cr blocks	
Index	Table
0	High Rate Inter

1	High Motion Inter
2	Mid Rate Inter

Table 70: Index/Coding Set Correspondence for PQINDEX > 6

Y, Cb and Cr blocks	
Index	Table
0	Low Motion Inter
1	High Motion Inter
2	Mid Rate Inter

8.3.6.2.4 Run-level Decode

The process for decoding the run-level pairs obtained in the coefficient decoding process described above is nearly the same as described in section 8.1.1.11. 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 Zig-zag Scan of Coefficients

The one-dimensional array of quantized coefficients produced in the run-level decode process described above are scanned out into a two-dimensional array in preparation for the Inverse Transform. The process is similar to that described in section 8.1.1.12 for intra blocks. The differences are:

1) Each Transform type has an associated zig-zag scan array.
2) The scan arrays for some transform types are different in the interlace mode and progressive mode of advanced profile.
3) Unlike the zig-zag scanning process for intra blocks where one of three arrays are used depending on the DC prediction direction, only one array is used for inter blocks.

The zigzag scan arrays for Inter blocks in simple and main profiles are given in Table 226 to Table 229 in section 11.9.2. The scan arrays in the advanced profile depend on whether interlace or progressive mode is used.

In progressive mode of advanced profile, the scan arrays for Inter 8x8 and 4x4 blocks are identical to those for simple and main profiles, as in Table 226 and Table 229. The scan arrays for Inter 8x4 and 4x8 blocks are defined in Table 230 and Table 231. In interlaced mode of the advanced profile, the scan arrays for Inter 8x8, 8x4, 4x8 and 4x4 blocks are defined in Table 232 to Table 235.

8.3.6.3 Inverse Quantization

The non-zero quantized coefficients reconstructed as described in the sections above are inverse quantized in one of two ways depending on the value of PQUANT.

If the uniform quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep)$$

If the nonuniform quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep) + \text{sign}(quant\_coeff) * quant\_scale$$

where:

quant_coeff is the quantized coefficient

dequant_coeff is the inverse quantized coefficient

quant_scale = The quantizer scale for the block (either PQUANT or MQUANT)

halfstep = The half step encoded in the picture layer as described in section 7.1.1.16.

PQUANT is encoded in the picture layer as described in section 7.1.1.15.

MQUANT is encoded as described in section 8.3.5.2.

8.3.6.4 Inverse TRANSFORM

After reconstruction of the TRANSFORM coefficients, the resulting 8x8, 8x4, 4x8 or 4x4 blocks are processed by the appropriate two-dimensional inverse transforms (INVERSETRANSFORM). The 8x8 blocks are transformed using the 8x8 INVERSETRANSFORM, the 8x4 blocks are transformed using the 8x4 INVERSETRANSFORM, the 4x8 blocks are transformed using the 4x8 INVERSETRANSFORM and the 4x4 blocks are transformed using the 4x4 INVERSETRANSFORM. The inverse transforms output ranges from -256 to +255 after clipping to be represented with 9 bits.

See section 8.8 regarding INVERSETRANSFORM implementation and conformance.

8.3.6.5 Motion Compensation

The 8x8, 8x4, 4x8 or 4x4 error block or blocks are 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) is 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.

If the picture layer syntax element MVMODE (see section 7.1.1.22) indicates that 1MV Halfpel or 1MV Halfpel Bilinear is used as the motion compensation mode, then all motion vectors are expressed in half-pixel resolution. For example, a horizontal motion component of 4 would indicate a position 2 pixels to the right of the current position and a value of 5 would indicate a position of 2 ½ pixels to the right. If the picture layer syntax element MVMODE (see section 7.1.1.22) indicates that 1MV or Mixed MV is used as the motion compensation mode, then all motion vectors are expressed in quarter-pixel resolution. For example, a horizontal motion component of 4 would indicate a position 1 pixel to the right of the current position and a value of 5 would indicate a position of 1 1/4 pixels to the right. In 1MV Halfpel Bilinear mode, all non-integer pixel motion vector offsets use a bilinear filter to compute the interpolated pixels. Otherwise, all non-integer pixel motion vector offsets use a bicubic filter to compute the interpolated pixels

8.3.6.5.1 Bilinear Interpolation

The following sections describe the bilinear filter operations. The bilinear filter operates as shown in Figure 53.

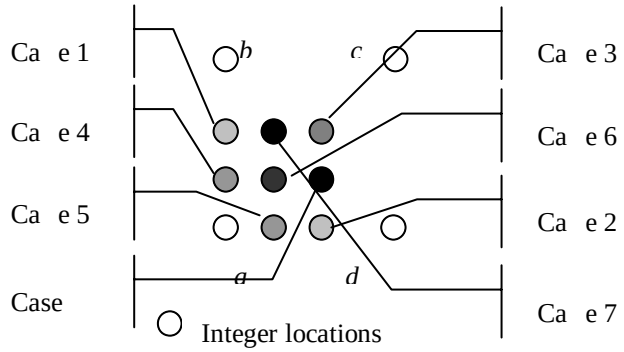


Figure 53: Bilinear filter operation

Figure 53 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

Although the bilinear interpolator is defined for quarter pixel motion vector resolution, only half pixel motion is allowed for the luminance blocks. In other words, in Figure 53, only cases 1, 2 and 3 are permitted for luminance. Cases 4 through 8 are used only for chrominance. The reference pixels *a*, *b*, *c*, *d* are used to generate the interpolated pixel as follows:

For the cases 1, 2 and 3, the interpolated pixel *p* is given by the following equations:

$$p = (a + b + 1 - R) >> 1 \quad \text{:case 1}$$

$$p = (a + d + 1 - R) >> 1 \quad \text{:case 2}$$

$$p = (a + b + c + d + 2 - R) >> 2 \quad \text{:case 3}$$

where *R* 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 and vertical directions, multiplied by 4. Their values range from 0 through 4 within the area bounded by the four pixels shown in Figure 53, with the origin located at *a*. Tables F and F' are the filter coefficients. F[] = { 4, 3, 2, 1, 0 } and G[] = { 0, 1, 2, 3, 4 }. The interpolated value *p* is given by:

$$p = (F[x] F[y] a + F[x] G[y] b + G[x] G[y] c + G[x] F[y] d + 8 - R) >> 4$$

For example, consider case 8. The subpixel shifts for case 8 are x=2, y=1. *p* is therefore:

$$p = (6 a + 2 b + 2 c + 6 d + 8 - R) >> 4$$

The above expression is identical to

$$p = (3 a + b + c + 3 d + 4 - R) >> 3$$

when *R* is 0 or 1, which is the case with the rounding control value. Similarly, it may be shown that cases 1 through 3 simplify to their earlier definitions.

Integer precision of 12 bits or more is required to implement the the bilinear interpolator.

8.3.6.5.2 Bicubic Interpolation

The following section describes the bicubic filter operations.

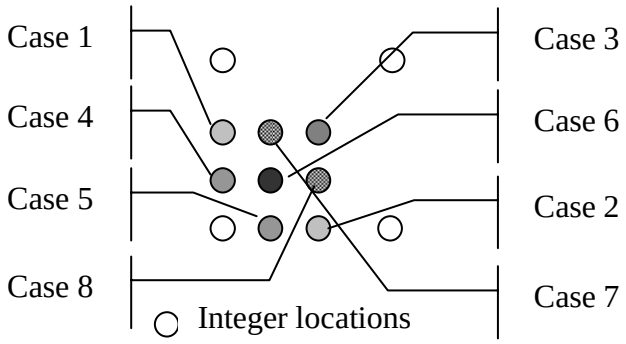


Figure 54: Quarter pel bicubic filter cases

Figure 54 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

One-dimensional Bicubic Interpolation (Cases 1, 2, 4 and 5)

In Figure 54, cases 1, 2, 4 and 5 represent the cases where interpolation occurs in only one dimension – either horizontal or vertical. The following filters are used for the possible shift locations:

½ pel shift F1: [-1 9 9 -1]

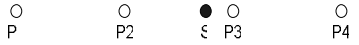
¼ pel shift F2: [-4 53 18 -3]

¾ pel shift F3: [-3 18 53 -4]

Figure 55 shows the pixels that are 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 applies to vertical interpolation.

½ Pixel Shift

¼ Pixel Shift



¾ Pixel Shift

Figure 55: Pixel Shifts

The following equations show the filtering operation for each case:

(-1*P1 + 9*P2 + 9*P3 -1*P4 + 8 – *r*) >> 4 (1/2 pixel shift)

(-4*P1 + 53*P2 + 18*P3 – 3*P4 + 32 – *r*) >> 6 (1/4 pixel shift)

(-3*P1 + 18*P2 + 53*P3 -4*P4 +32 – *r*) >> 6 (3/4 pixel shift)

The value *r* in the equations above depends on *R*, the frame-level round control value (see section 8.3.7 for a description) and the interpolation direction as follows:

$$r = \begin{cases} 1-R & (vertical\ direction-cases 1 and\ 4) \\ R & (horizontal\ direction-cases\ 2 and 5) \end{cases}$$

Two-dimensional Bicubic Interpolation

In Figure 54, cases 3, 6, 7 and 8 are the cases where interpolation occurs in both the horizontal and vertical directions.

Two-dimensionally interpolated pixel locations first interpolate along the vertical direction, and then along the horizontal direction using the appropriate filter among F1, F2 and F3 specified above. Rounding is applied after vertical filtering and after horizontal filtering. The rounding rule ensures retention of maximum precision permitted by 16 bit arithmetic in the intermediate results.

The rounding rule after vertical filtering is defined as

$$(S + rndCtrlV) >> shiftV$$

where

S = vertically filtered result, 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.

$rndCtrlV = 2^{shiftV-1} - 1 + R$ (see section 8.3.7 for a description of *R*)

The rounding rule after horizontal filtering is:

$$(S + 64 - R) >> 7.$$

where

S = horizontally filtered result

R = frame level round control value (see section 8.3.7)

All of the bicubic filtering cases may potentially produce an interpolated pixel whose value is negative, or larger than the maximum range (255). In these cases, the output is clipped to lie within the range – underflows are set to 0 and overflows to 255.

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 56 illustrates this process.

```
for (row= 0; row < 8; row++)
{
  for (col = 0; col < 8; col++)
        reconblock[row*8 + col] = clip(predblock[row*8 + col] + errorblock[row*8 + col])
}

where:
clip(n) =
            0 if n < 0
            255 if n > 255
            n otherwise
```

Figure 56: Inter block reconstruction pseudo-code

8.3.7 Rounding Control

Section 8.3.6.2 describes the interpolation operations used to generate subpixel values in the reference blocks. Rounding is controlled by a value *R* called the rounding control value.

In simple and main profiles, the value of *R* toggles back and forth between 0 and 1 at each P frame. At each I frame, the value of *R* is reset to 0. Therefore, the value of *R* for the first P frame following an I frame is 0.

In advanced profile, the value is R is decoded from the RNDCTRL syntax element in the picture header.

8.3.8 Intensity Compensation

If the picture layer syntax element MVMODE indicates that intensity compensation is used for the frame, then the pixels in the reference frame are remapped prior to using them as predictors for the current frame. As section 8.3.4.3 describes, when intensity compensation is used, the LUMSCALE and LUMSHIFT syntax elements are present in the picture bitstream. The following pseudocode illustrates how the LUMSCALE and LUMSHIFT values are used to build the lookup table used to remap the reference frame pixels.

```
if (LUMSCALE == 0)
{
      iScale = - 64
      iShift = 255 * 64 + 32 - LUMSHIFT *2 * 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
}
```

The Y component of the reference frame is remapped using the LUTY[] table generated above and the Cb/Cr components are 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 luminance pixel value in the reference frame and $\overline{p}_Y$ is the remapped luminance pixel value in the reference frame and p_{UV} is the original Cb or Cr pixel value in the reference frame and $\overline{p}_{UV}$ is the remapped Cb or Cr pixel value in the reference frame.

8.4 Progressive B Frame Decoding

At the top level, most B frames are coded as bidirectionally predicted frames as the name suggests. When using B frames, both forward and backward frames are needed for motion compensation. In certain cases however it is more economical to code a frame independent of its anchors – in other words as an intra B frame. An intra B frame has the same frame level syntax an inter B frame, but its macro block level decoding follows that of a I frame.

Normal B frames are coded by coding 16x16 tiles of the image (macro blocks). Unlike P frames there is no “4MV” motion compensation mode. At the frame level, only two choices for motion vector resolution are permitted – quarter pel bicubic and half pel bilinear.

8.4.1 Skipped Anchor Frames

If an anchor frame is coded as skipped then it is treated as a P frame which is identical to its reference frame. Therefore, the reconstruction of that frame may be treated conceptually as a copy of the reference frame. In this case, both anchor frames are 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).

See section 8.3.1 for a description of skipped P frames. The method used to signal skipped P frames is the same method used to signal skipped B frames.

8.4.2 B Picture Layer Decode

Some B frame specific information is transmitted at the frame level. Apart from the frame type (PTYPE), a symbol called the B frame fraction (BFRACTION) is sent at the frame header. For main profile this indicates whether the B frame is coded using the I picture syntax, and if not, the scaling factor used to derive the direct motion vectors (explained in section 7.1.1.10). For advanced profile, BFRACTION only signals the scaling factor. The RANGEREDFRM syntax element is present in main profile B picture header if enabled at the sequence layer. It is not allowable to change the range at a B frame so this element is ignored. The remainder of the section deals only with predicted or "normal" B frames, i.e. those that aren't coded using the I picture syntax.

8.4.2.1 Bitplane Coding

As in P frames, some information is coded as a compressed bitplane that is sent at the frame level. For progressive B frames, two such bitplanes are sent – one denoting skipped macro blocks and the other denoting direct coded macro blocks. This information is sent at the macro block level when the raw coding mode is chosen. See section 7.2 for a description of bitplane coding.

8.4.2.2 Rounding Control

The rounding control parameter used by B frames is identical to that used by the previously decoded anchor.

8.4.2.3 Sync Markers

B frames do not contain sync markers.

8.4.2.4 Picture Resolution

If variable resolution coding is enabled for the sequence (signaled by the MULTIRES flag in the sequence header, see section 6.1.13), then the resolution of the B frame is determined by the resolution of the two reference frames. See section 8.1.1.3 for a description of how the current resolution is signaled. The resolution of the B frame is the same as the resolution of the two reference frames. The two reference frames shall always have the same resolution. This will always be the case for B frames that occur temporally between two P frames since a resolution change may only occur at I frames. This restriction means that B frames may never occur temporally between an I and P or two I frames where the I frame is a different resolution than the preceding I or P frame.

8.4.2.5 Range Reduction Frame (RANGEREDFRM)

The RANGEREDFRM is only signaled when RANGERED is signaled at the sequence level. The RANGEREDFRM shall carry the same value as the RANGEREDFRM syntax element of the temporally subsequent anchor frame. This implies that no scaling shall be performed for performing motion compensation from the temporally subsequent frame. However, scaling shall be performed for prediction from the temporally preceding anchor frame. This scaling is identical to the scaling performed for predicting P frames, and is described in section 8.3.4.12.

8.4.3 B Macroblock Layer Decode

Macro blocks in B frames are identified as belonging to one of four modes, viz. *backward*, *forward*, *direct* and *interpolated*. The forward mode is akin to conventional P picture prediction. In the forward mode, the B macro block is interpolated from its temporally previous anchor frame only. Likewise, backward mode macro blocks are entirely interpolated from their temporally subsequent anchor frame.

8.4.3.1 Long and Short Types

When a B frame is closer to its temporally previous reference, it may be 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 may be expected that it references the future anchor 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.

8.4.3.2 Direct and Interpolated Modes

Macroblocks for which BMVTYPE is direct or interpolated use both the anchors for prediction. They use two sets of motion vectors (MV's), one each to reference into the previous and next anchor frame. In both cases the pixels are interpolated from the two reference frames 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 are explicitly coded within the bitstream. In direct mode the forward and backward motion vectors are derived by scaling the corresponding motion vectors from the backward reference frame.

We buffer both forward and backward motion vectors for the current (B) frame, as well as all the motion vectors from the next anchor frame (P) to use with the direct mode. All direct mode MV's (motion vectors) are treated as (0, 0) when the co-located macroblock (of the next anchor frame) is INTRA. This is also true when the previously decoded frame (i.e. the temporally "next" anchor frame) is an I-frame.

If the P frame's co-located MV was 1 MV then that MV is simply buffered for the next B to be coded, else we take median4 to average the 4 MV's. [Special cases are when we use average (integer division with truncation towards zero) of 2 non-intra's if 2 out of 4 MV's are intra, and median3 non-intra's if 1 of the 4 MV's is intra]. If all 4 MV's are intra then the direct mode's MV is also intra (treated as 0,0 for the direct mode). NB: median3 and median4 operations are defined in the document.

Given that the subsequent anchor frame was a P frame (in case the next frame was I, all the motion vectors are assumed to be (0,0)), and the collocated macroblock contained a motion vector MV (MV_X, MV_Y), the direct mode computes 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 subsequent anchor frame, (MV_X_B, MV_Y_B) in the following manner –

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 (Half pel units) {
        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;
    }
End Scale_Direct_MV
```

If the collocated macroblock is an intra macroblock, we set (MV_X, MV_Y) to be zero. If the collocated macroblock contains 4 motion vectors, (MV_X, MV_Y) is computed as the "median" of the 4 motion vectors. If all 4 blocks were non-intra blocks, the median4 operator is used on the 4 motion vectors. If three of the blocks are non-intra, the median3 operator is used on the 3 motion vectors. If two of them were non intra, the arithmetic mean of the two motion vectors is used.

"ScaleFactor" is computed at the start of decoding each B frame, as follows –

```
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 };

Frame_Initialization(code word)

    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
```

And "code word" is obtained by decoding the frame level syntax element BFRACTION, as shown in Table 26. In this VLC table, the code-words 000 through 110 are known as the "short" code words, and the others are known as the "long" code words.

Figure 57 shows how direct mode scales the motion vectors from the next P frame.

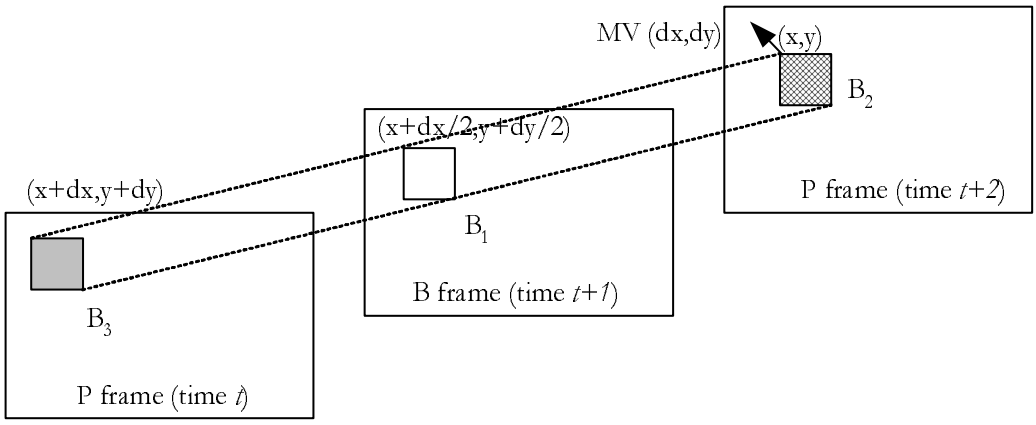


Figure 57: Direct Mode Prediction

8.4.3.3 Motion Vector Prediction

MV prediction for frame B pictures follows exactly the same rules as in P frames, and will not be repeated here. The only additional point to note is that two separate MV buffers are kept for forward and backward MV's, and the MV prediction rules are applied on each of these while decoding an MV of the like type, i.e. forward MV's are used to predict an incoming forward MV, and backward MV's are used to predict an incoming backward MV. In the interpolated mode we use both forward and backward prediction to predict the two incoming MV's, and in the direct mode we scale the next field P's collocated MV.

Macro blocks that use the direct or interpolate modes have valid forward and backward motion vectors associated with them. Macro blocks that are coded as forward or backward do not have valid backward and forward components respectively. For these cases, the direct mode motion vectors used in backward and forward directions respectively are used to fill in (see 8.4.3.2).

For intra coded macro blocks, the "intra motion vector" is used to fill in both forward and backward motion prediction planes. An "intra motion vector" is simply a unique large constant (0X4000 which exceeds the range of valid MV's) that is filled into the MV arrays to indicate that the MB was coded as intra.

In progressive B frames the direct mode motion vector has to be calculated even for an MB that is only forward (or backward predicted). If it is forward predicted we calculate the backward pointing direct mode MV and put that in the buffer of backward MV's (the reconstructed forward MV which is obtained from the bit stream would go into the buffer of forward MV's), and if the MB is backward predicted you will calculate the forward pointing direct mode MV and put that in the buffer of forward MV's (the reconstructed backward MV is inserted into the backward MV buffer as usual).

8.4.3.4 Motion Vector Transmission

Whether or not a macro block is coded as direct is known at the start of decoding macro block level information. Non-direct macro blocks have one or two associated *residual* motion vectors, direct macro blocks have none. Skipped macro blocks that are direct coded have zero residual Transform coefficients (Transform AC for intra macro blocks), and skipped non-direct coded macro blocks have zero residual motion as well. If an MB is "skipped", the MB mode shall be signaled, to identify whether the "skipped" MB shall use direct, forward, backward or interpolated prediction. Skipping in the context of B frames means shall imply that the MV prediction error is zero, i.e. decoding is carried on as usual (treating each mode with the appropriate decoding rules), and the predicted MV's will be exactly the ones we use.

It is possible to gain some efficiency by coding the mode of the non-direct macro block *after* sending the first motion vector. Since VC-9 jointly codes motion vector information with the intra flag, intra macro blocks are identified after decoding the first motion vector. It is not necessary to send any mode information subsequently after an intra motion vector is received.

When the first motion vector is non-intra, the macro block type is sent. This is based on the efficient remapping of forward and backward into short and long types as explained earlier. The second motion vector is sent only if the macro block is interpolated, and if the **last flag** (see section 8.3.5.2.1) component of the first motion vector is nonzero. If the '**last flag'** component is zero for an interpolated macro block, it is implied that the second residual motion vector of the interpolated block is zero, and so are the residual Transform terms.

8.4.3.5 Subpixel Interpolation

Subpixel interpolation of B frames is performed in the same manner as interpolation of P frames. The valid modes are quarter pel bicubic and half pel bilinear.

8.4.3.6 Pixel Averaging

Macroblocks that are coded using the direct or interpolated modes have two associated predictions drawn from the two reference anchors. These predictions are merged into one by averaging. A pixelwise mean operation with upwards rounding is employed to perform averaging.

8.4.3.7 Reconstructing and Adding Error

The decoding, dequantization, Inverse Transform, error addition and clamping of residuals to the predicted macro blocks is performed in a manner identical to that used in P frames. Intra macro blocks are also coded as they would be in P frames. However, the overlapped smoothing operation applied to edges between intra blocks in P frames is not performed in B frames. The exception is with a B frame encoded as an I frame. In this case, the overlap smoothing operation is performed.

8.4.4 B Block Layer Decode

Block decoding syntax and operations are the same as for P pictures and will not be repeated. I-MB's in B frames are also the same as those in P frames.

8.5 Overlapped Transform

If the sequence layer syntax element OVERLAP is set to 1, then a filtering operation is conditionally performed across edges of two neighboring Intra blocks, for both the luminance and chrominance channels. This filtering operation (referred to as *overlap smoothing*) is performed subsequent to decoding the frame, and prior to in-loop deblocking. However, overlap smoothing may be done after the relevant macroblock slices are decoded as this is functionally equivalent to smoothing after decoding the entire frame.

Overlapped transforms are modified block based transforms that exchange information across the block boundary. With a well designed overlapped transform, blocking artifacts may be minimized. For intra blocks, VC-9 simulates an overlapped transform by coupling an 8x8 TRANSFORM-like block transform with overlap smoothing. Edges of an 8x8 block that separate two intra blocks are smoothed – in effect an overlapped transform is implemented at this interface.

Figure 58 shows a portion of a P frame with I blocks. This could be either the Y or U/V 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 58 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.

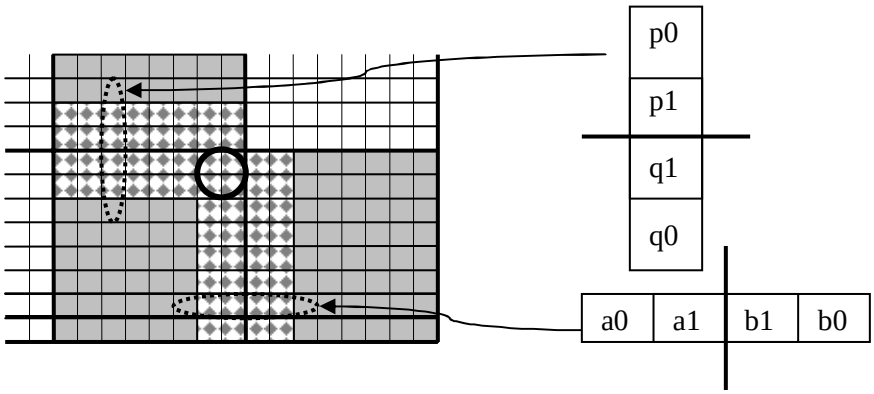


Figure 58: Example showing overlap smoothing

8.5.1 Overlap Smoothing in Main and Simple Profiles

Overlap smoothing in main and simple profiles is applied subject to the following conditions:

1. All 8x8 block boundaries (except those at the periphery of the frame) are smoothed for I frames
2. Only block boundaries separating two intra blocks are smoothed for P frames
3. No overlap smoothing is performed for predicted B frames, i.e. B frames that are not encoded as Intra
4. Subject to the above, overlap smoothing is applied only if the frame level quantization step size PQUANT is 9 or above
5. There is no dependence on DQUANT or differential quantization across macroblocks

Overlap smoothing is carried out on the unclamped 10 bit reconstruction. In other words, the input to the overlap smoothing process is raw, unclamped 10 bit inverse transformed spatial pixels. This is necessary because the forward process associated with overlap smoothing may result in range expansion beyond the permissible 8 bit range for pixel values. The result of overlap smoothing is clamped down to 8 bits, in line with the remainder of the pixels not touched by overlap smoothing.

Vertical edges (pixels a0, a1, b1, b0 in the above example) are filtered first, followed by the horizontal edges (pixels p0, p1, q1, q0). The intermediate result following the first stage of filtering (vertical edge smoothing) is stored in 16 bit. The core filters applied to the four pixels straddling either edge are given 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 take on alternating values of 3 and 4. For both horizontal and vertical edge filters, the rounding values are r0 = 4, r1 = 3 for odd-indexed columns and rows respectively, assuming the numbering within a block to start at 1. For even-indexed columns / rows, r0 = 3 and r1 = 4. Filtering is defined as an in-place 16 bit operation – thus the original pixels are overwritten after smoothing. 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 58, are filtered in both directions. The order of filtering determines their final values, and therefore it is important to maintain the order – vertical edge filtering followed by horizontal edge filtering – for bit exactness.

8.5.2 Overlap Smoothing in Advanced Profile

Overlap smoothing is applied subject to the following conditions:

1. No overlap smoothing is performed for any frame if the sequence level OVERLAP flag is FALSE – the remainder of these rules apply only when OVERLAP is TRUE
2. No overlap smoothing is performed for predicted B frames, i.e. B frames that are not encoded as Intra
3. Only block boundaries separating two intra blocks are 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 B frames encoded as I, 8x8 block boundaries (except those at the periphery of the frame) are smoothed as per the following rules
    b. When picture level quantization step size PQUANT is 9 or higher (regardless of HALFQP), all 8x8 block boundaries (except those at the periphery of the frame) are smoothed
    c. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), no 8x8 block boundaries are smoothed if the conditional overlap flag CONDOVER is 0 binary.
    d. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), all 8x8 block boundaries (except those at the periphery of the frame) are smoothed if the conditional overlap flag CONDOVER is 10 binary
    e. When picture level quantization step size PQUANT is 8 or lower (regardless of HALFQP), some 8x8 block boundaries (except those at the periphery of the frame) are smoothed if the conditional overlap flag CONDOVER is 11 binary as per the following rules
        i. Internal 8x8 block boundaries within the luminance plane of a macroblock are smoothed when the OVERFLAGS pattern for the macroblock is 1
        ii. 8x8 block boundaries between adjacent macroblocks (both luminance and chrominance) are smoothed only when the OVERFLAGS pattern for both adjacent macroblocks are 1
5. There is no dependence on DQUANT or differential quantization across macroblocks

Conditional overlap is applicable only for I 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 both macroblocks' OVERFLAGS are 1. There is no block or block edge level control.

Overlap smoothing is carried out on the unclamped 10 bit reconstruction. This is necessary because the forward process associated with overlap smoothing may result in range expansion beyond the permissible 8 bit range for pixel values. The result of overlap smoothing is clamped down to 8 bits, in line with the remainder of the pixels not touched by overlap smoothing.

Vertical edges (pixels a0, a1, b1, b0 in the above example) are filtered first, followed by the horizontal edges (pixels p0, p1, q1, q0). The intermediate result following the first stage of filtering (vertical edge smoothing) is stored in 16 bit. The core filters applied to the four pixels straddling either edge are given 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 take on alternating values of 3 and 4 to ensure statistically unbiased rounding. The original values are filtered by the matrix with entries that are clearly easy to implement. These values, after adding the rounding factors, are bit shifted by three bits to give the filtered output (y0, y1, y2, y3).

For both horizontal and vertical edge filters, the rounding values are r0 = 4, r1 = 3 for odd-indexed columns and rows respectively, assuming the numbering within a block to start at 1. For even-indexed columns / rows, r0 = 3 and r1 = 4. Filtering is defined as an in-place operation – thus the original pixels are overwritten after smoothing. 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 58, are filtered in both directions. The order of filtering determines their final values, and therefore it is important to maintain the order – vertical edge filtering followed by horizontal edge filtering – for bit exactness. Conceptually, clamping is to be performed subsequent to the two directional filtering stages, on all pixels that are filtered. However, there may be some computational advantage to combining clamping with filtering – this is an implementation issue as long as it is done carefully to generate the correct output.

8.6 In-loop Deblock Filtering

If the sequence layer syntax element LOOPFILTER = 1, then a filtering operation is performed on each reconstructed frame. This filtering operation is performed prior to using the reconstructed frame as a reference for motion predictive coding. Therefore, it is necessary that the decoder perform the filtering operation strictly as defined. When there are multiple slices in a picture, the loopfilter for each slice is performed independently as described 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 is performed at all 8x8 block boundaries. Figure 59 and Figure 60 show the pixels that are filtered along the horizontal and vertical border regions. The figures show the upper left corner of a component (luma, Cb or Cr) plane. The crosses represent pixels and the circled crosses represent the pixels that are filtered.

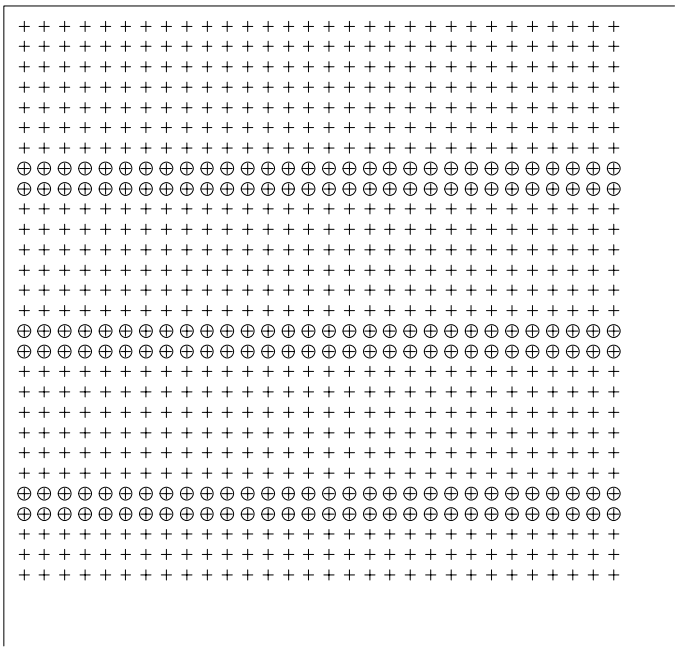


Figure 59: Filtered horizontal block boundary pixels in I picture

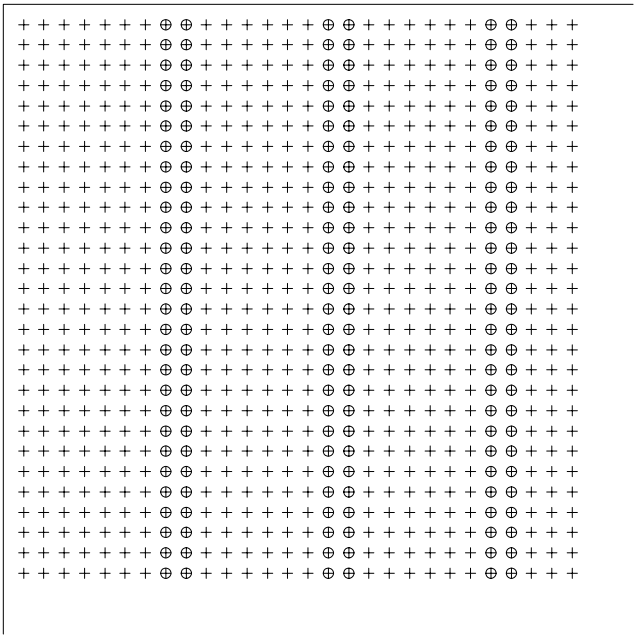


Figure 60: Filtered vertical block boundary pixels in I picture

As the figures show, the top horizontal line and first vertical line are not filtered. Although not depicted, the bottom horizontal line and last vertical line are also not filtered. In more formal terms, the following lines are 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 are 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 always use an 8x8 Transform to transform the samples and the 8x8 block boundaries are always filtered. Inter-coded blocks may use an 8x8, 8x4, 4x8 or 4x4 Inverse Transform to construct the samples that represent the residual error. Depending on the status of the neighboring blocks, the boundary between the current and neighboring blocks may or may not be filtered. The decision of whether to filter a block or subblock border is as follows:

1) The boundaries between coded (at least one non-zero coefficient) subblocks (8x4, 4x8 or 4x4) within an 8x8 block are always filtered.

2) The boundary between a block or subblock and a neighboring block or subblock is not filtered if both have the same motion vector and both have no residual error (no Transform coefficients). Otherwise it is filtered.

Figure 61 shows examples of when filtering between neighboring blocks does and does not occur. In this example it is assumed that the motion vectors for both blocks is 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 neighbors. The same applies for vertical neighbors.

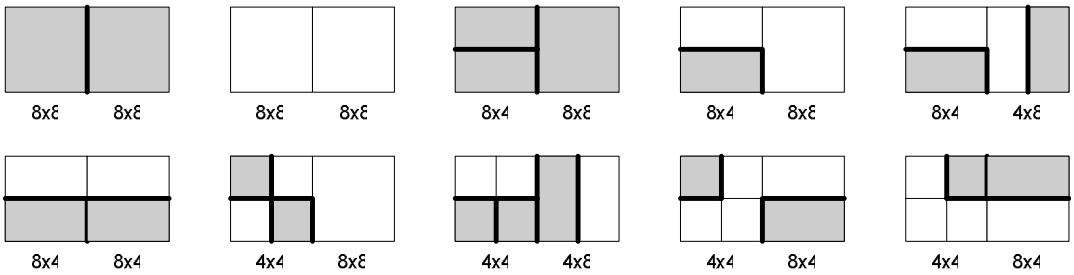


Figure 61: Example filtered block boundaries in P frames

Figure 62 and Figure 63 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 will filtered if the conditions specified above are met.

Figure 62 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 63 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 are not filtered.

The order in which pixels are filtered is important. First, all the 8x8 block horizontal boundary lines in the frame are filtered starting from the top line. Next, all 8x4 block horizontal boundary lines in the frame are filtered starting from the top line. Next, all 8x8 block vertical boundary lines are filtered starting from the leftmost line. Last, all 4x8 block vertical boundary lines are filtered starting with the leftmost line. In all cases, the rules specified above are used to determine whether the boundary pixels are filtered for each block or subblock.

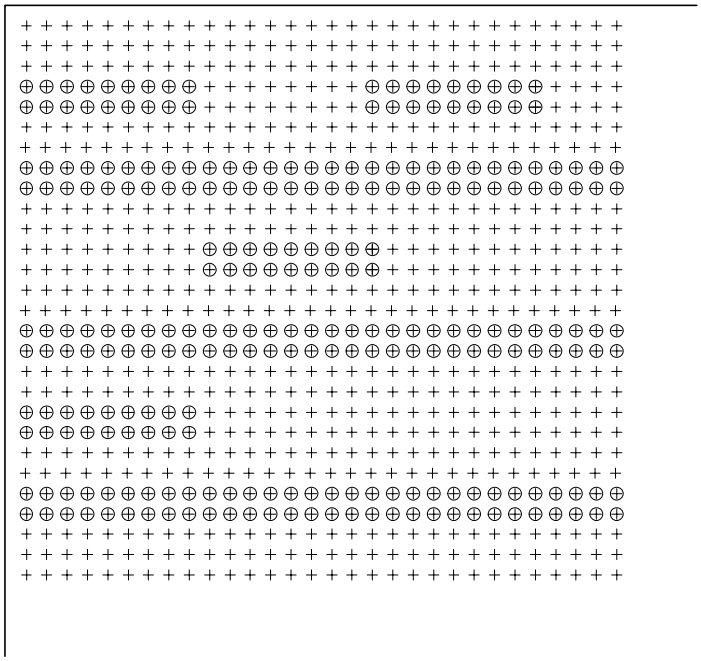


Figure 62: Horizontal block boundary pixels in P picture

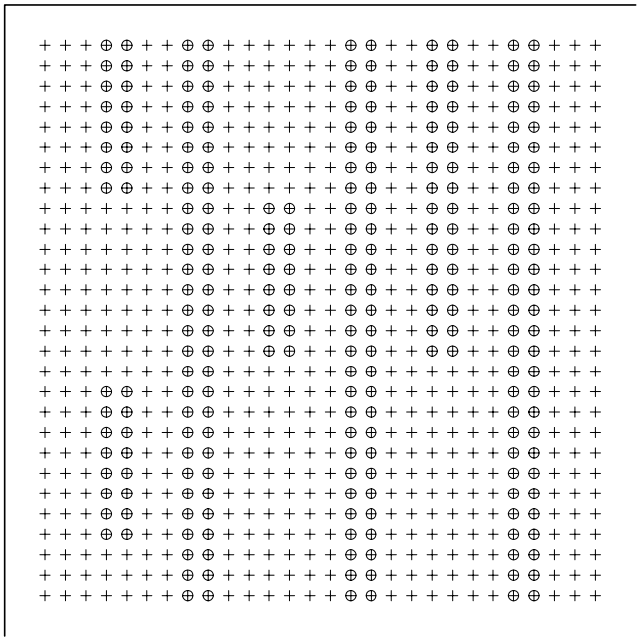


Figure 63: Vertical block boundary pixels in P picture

8.6.3 B Picture In-loop Deblocking

This is exactly the same as I-picture in-loop deblocking, i.e. we only filter the 8x8 block boundaries, and don't consider MV's or 4x8/8x4 as with P pictures.

8.6.4 Filter Operation

This section describes the filtering operation that is performed on the boundary pixels in I and P frames.

For P frames the decision criteria listed in section 8.6.2 determines which vertical and horizontal boundaries are filtered. For I frames, all the 8x8 vertical and horizontal boundaries are filtered. Since the minimum number of consecutive pixels that will be filtered in a row or column is four and the total number of pixels in a row or column will always be a multiple of four, the filtering operation is performed on segments of four pixels.

For example, if the eight pixel pairs that make up the vertical boundary between two blocks is filtered, then the eight pixels are divided into two 4-pixel segments as shown in Figure 64. In each 4-pixel segment, the third pixel pair is filtered first as indicated by the X's. The result of this filter operation determines whether the other three pixels in the segment are also filtered, as described below.

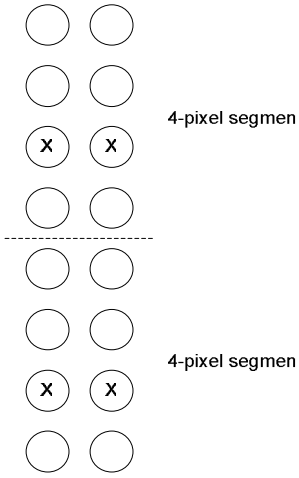


Figure 64: Four-pixel segments used in loop filtering

Figure 65 shows the pixels that are used in the filtering operation performed on the 3rd pixel pair. Pixels P4 and P5 are the pixel pairs that may be changed in the filter operation.

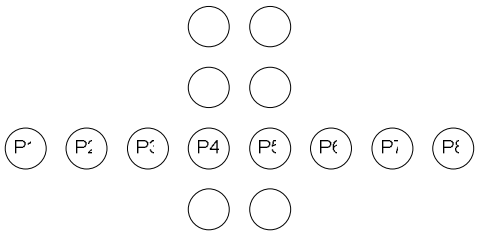


Figure 65: Pixels used in filtering operation

The pseudocode of Figure 66 shows the filtering operation performed on the 3rd pixel pair in each segment. The value filter_other_3_pixels indicates whether the remaining 3 pixel pairs in the segment are also filtered. If filter_other_3_pixels = TRUE, then the other three pixel pairs are filtered. If filter_other_3_pixels = FALSE, then they are not filtered, and the filtering operation proceeds to the next 4-pixel segment. The pseudocode of Figure 67 shows 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
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 66: Pseudo-code illustrating filtering of 3rd pixel pair in segment

```
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 67: 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 is used for filtering the horizontal boundary pixels.

8.7 Bitplane Coding

Certain macroblock-specific information may 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) may 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 for this rule is if the bitplane coding mode (described below) is set to Raw Mode. In this case, the status for each macroblock is coded as one bit per symbol, and transmitted along with other macroblock level syntax elements. Raw mode is the only allowed bit plane mode when multiple slices are used to code the frame. VC-9 uses bitplane coding in three cases to signal information about the macroblocks in a frame. These are: 1) signaling skipped macroblocks, 2) signaling field or frame macroblock mode and 3) signaling 1-MV or 4-MV motion vector mode for each macroblock. This section describes the bitplane coding scheme.

Frame-level bitplane coding is used to encode 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 is used to encode 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 shows the syntax elements that make up the bitplane coding scheme. The follow sections describe how to decode the bitstream and reconstruct the bitplane.

8.7.1 INVERT

The INVERT syntax element shown in the syntax diagram of Figure 28 is a one bit code, which if set indicates that the bitplane has more set bits than zero bits. Depending on INVERT and the mode, the decoder shall invert the interpreted bitplane to recreate the original. Note that the value of this bit shall be ignored when the raw mode is used.

8.7.2 IMODE

The IMODE syntax element shown in the syntax diagram of Figure 28 encodes the mode used code the bitplane. The seven modes are described in section 8.7.3. Table 71 shows the codetable used to encode the IMODE syntax element.

Table 71: IMODE Codetable

CODING MODE	CODEWORD
Raw	0000
Norm-2	10

Diff-2	001
Norm-6	11
Diff-6	0001
Rowskip	010
Colskip	011

8.7.3 DATABITS

The DATABITS syntax element shown in the syntax diagram of Figure 28 is an entropy coded stream of symbols that is based on the coding mode. The seven coding modes are described in the following sections.

8.7.3.1 Raw mode

In this mode, the bitplane is encoded as one bit per symbol scanned in the raster-scan order of macroblocks, and sent as part of the macroblock layer. DATABITS is *rowMB* × *colMB* bits in length.

8.7.3.2 Normal-2 mode

If *rowMB* × *colMB* is odd, the first symbol is encoded raw. Subsequent symbols are encoded pairwise, in natural scan order. The binary VLC table in Table 72 is used to encode symbol pairs.

Table 72: Norm-2/Diff-2 Code Table

SYMBOL 2N	SYMBOL 2N + 1	CODEWORD
0	0	0
1	0	100
0	1	101
1	1	11

8.7.3.3 Diff-2 mode

The Normal-2 method is used to produce the bitplane as described in section 8.7.3.2 and then the **Diff**$^{-1}$ operation is applied to the bitplane as described in section 8.7.3.8.

8.7.3.4 Normal-6 mode

In the *Norm-6* and *Diff-6* modes, the bitplane is encoded in groups of six pixels. These pixels are grouped into either 2x3 or 3x2 tiles. The bitplane is tiled maximally using a set of rules, and the remaining pixels are encoded using a variant of *row-skip* and *column-skip* modes.

2x3 “vertical” tiles are used if and only if *rowMB* is a multiple of 3 and *colMB* is not. Else, 3x2 “horizontal” tiles are used, as shown in Figure 68.

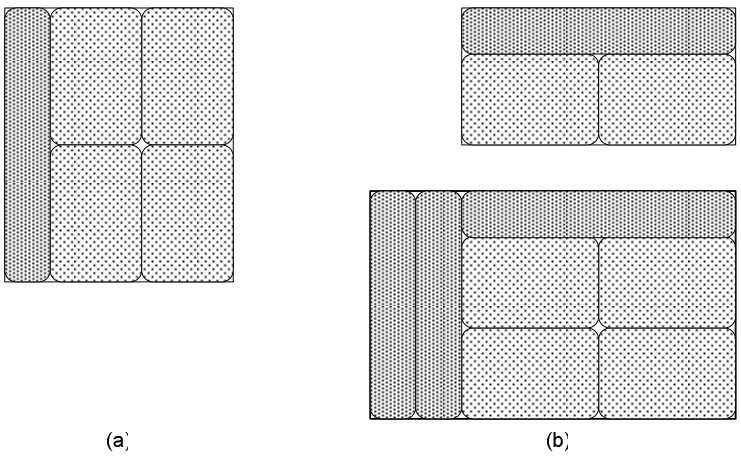


Figure 68: An example of 2x3 "vertical" tiles (a) and 3x2 "horizontal" tiles (b) – the elongated dark rectangles are 1 pixel wide and encoded using row-skip and column-skip coding.

While the picture is tiled as shown in Figure 68, (with linear tiles along the top and left edges of the picture), the coding order of the tiles follows the following pattern. The 6-element tiles are encoded first, followed by the *column-skip* and *row-skip* encoded linear tiles. If the array size is a multiple of 2x3 or of 3x2, the latter linear tiles do not exist and the bitplane is perfectly tiled.

The 6-element rectangular tiles are encoded using an incomplete Huffman code, i.e. a Huffman code which does not use all end nodes for encoding. 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 encode the tile. For $N = 3$, a fixed length escape is followed by a 5 bit fixed length code, and for $N > 3$, a fixed length escape is followed by the code of the complement of the tile.

The rectangular tile contains 6 bits of information. Let k be the code associated with the tile, where $k = b_i\, 2^i$, b_i is the binary value of the i^{th} bit in natural scan order within the tile. Hence $0 \le k \le 64$. Table 73 is used to encode k.

Table 73: 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	3	5
8	5	4		
9	3	8		
10	4	8		

11	2	5	5	5
12	5	8		
13	2	5	6	5
14	2	5	7	5
15	3	5	14	8
16	6	4		
17	6	8		
18	7	8		
19	2	5	9	5
20	8	8		
21	2	5	10	5
22	2	5	11	5
23	3	5	13	8
24	9	8		
25	2	5	12	5
26	2	5	13	5
27	3	5	12	8
28	2	5	14	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	17	5
36	12	8		
37	2	5	18	5
38	2	5	19	5
39	3	5	9	8
40	13	8		
41	2	5	20	5
42	2	5	21	5
43	3	5	8	8
44	2	5	22	5
45	3	5	7	8
46	3	5	6	8
47	3	5	6	4

48	14	8		
49	2	5	24	5
50	2	5	25	5
51	3	5	5	8
52	2	5	26	5
53	3	5	4	8
54	3	5	3	8
55	3	5	5	4
56	2	5	28	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 is used to produce the bitplane as described in section 8.7.3.4 and then the **Diff**$^{-1}$ operation is applied to the bitplane as described in section 8.7.3.8.

8.7.3.6 Row-skip mode

In the row-skip coding mode, all-zero rows are skipped with one bit overhead. The syntax is as shown in Figure 69.

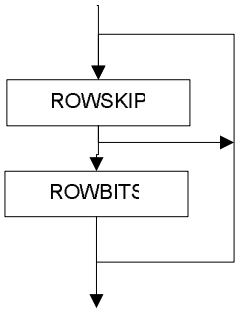


Figure 69: Syntax diagram of row-skip coding

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 are 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 are 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" is first decoded using the corresponding normal modes (Norm-2 or Norm-6 respectively). The differential bits are 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 $b_p(i,j)$ is generated as follows (from bits b(i, j) at positions (i, j)):

$$b_p(i,j) = \begin{cases} A & i = j = 0, \text{ or } b(i,j-1) \neq b(i-1,j) \\ b(0,j-1) & i == 0 \\ b(i-1,j) & \text{otherwise} \end{cases}$$

For the differential coding mode, the bitwise inversion process based on INVERT is not performed. However, the INVERT flag 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 equal to 0 if INVERT equals to 0 and A equals to 1 if INVERT equals to 1. The actual value of the bitplane is obtained by xor'ing the predictor with the decoded differential bit value.

8.8 Sync Markers

Sync markers are known sequences of bits that are inserted at important locations in the bitstream to clearly identify these locations. There are several reasons that require sync markers – the important ones are for error resilience and for parallel decoding of the bitstream. The simple and main profiles of VC-9 allow sync markers to be inserted in the bitstream. The sequence level flag SYNCMARKER determines whether sync markers are enabled in the sequence. If they are enabled, sync markers are sent only for I and P frames. No sync markers are allowed in B frames, including B frames coded as Intra. When SYNCMARKER is enabled, all bitplanes are encoded as raw bitplanes and the relevant data (e.g. 4MV/1MV, skipbit) is sent at the macroblock level. Sync markers are placed only at byte boundaries.

The sync markers in simple/main profiles of VC-9 are not guaranteed to be unique. However, sync markers are 24 bits in length and it is expected that even if they do randomly occur in a bitstream, such occurrences will be rare. Assuming a uniform distribution, it may be expected that one sync marker will randomly be emulated 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 is permitted in the first MB row. When sequence level SYNCMARKER is enabled, a single bit is sent 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 means that no sync marker follows. If this bit is zero, the remainder of the current byte is flushed out. Subsequently, the 24 bit byte aligned data is read from the bitstream. This is the sync marker.

Two sync markers are defined in VC-9. These are the *short* and *long* sync markers. Both the codes are 24 bits in length, but the *payload* or data following the sync marker differs in length. The short sync marker, whose hex representation is 0x0000AA, is followed by a 5 byte payload. The long sync marker (hex 0x0000AB) is followed by a 11 byte payload. Note that the first two bytes of both sync markers are zeros. This design makes the implementation of hardware-based sync marker detection schemes easier.

Currently, there is no requirement on the decoder to do anything with the payload in order to be compliant with the spec. The only interoperability requirement on any decoder is that the decoder correctly handle encoded content that may have embedded sync markers, assuming that no errors are present in the bitstream. The payload may be used to transmit parity, error detection and error recovery information.

Figure 70 represents a 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 0x0000AB is followed by a 11 byte payload.

There are no sync markers in B frames. For B frames, the entropy coded stream follows the order shown in Figure 70 (a) regardless of whether SYNCMARKER is 0 or 1.

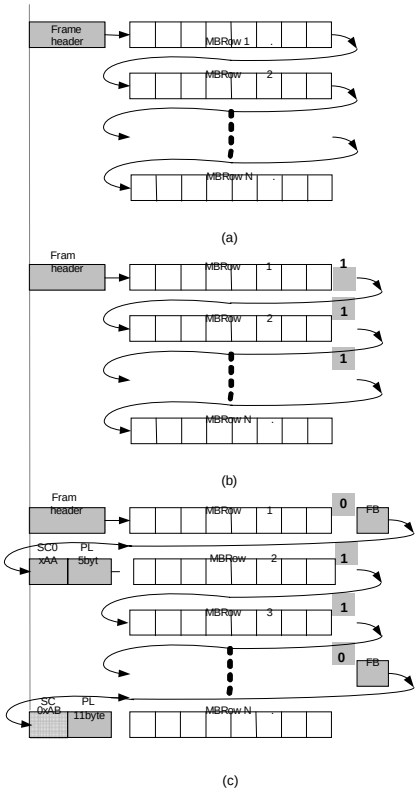


Figure 70: Sync markers in VC-9 – (a) shows sequence of entropy coded data with SYNCMARKER set to zero, (b) SYNCMARKER is 1 but no sync markers are actually sent and (c) SYNCMARKER is 1, a long and a short sync marker are sent, some slices do not have sync markers

8.9 INVERSETRANSFORM Conformance

The decoding process requires strict conformance with the VC-9 Inverse Transform implementation defined in Annex A.

9 Interlace syntax and semantics

9.1 Picture-level Syntax and Semantics

Each compressed video sequence is made up of data structured into three hierarchical layers. This section describes the syntax and semantics of the picture layer, macroblock layer, and block layer, when a picture is coded in interlace mode . Figure 71 through Figure 87 show the bitstream elements that make up each layer.

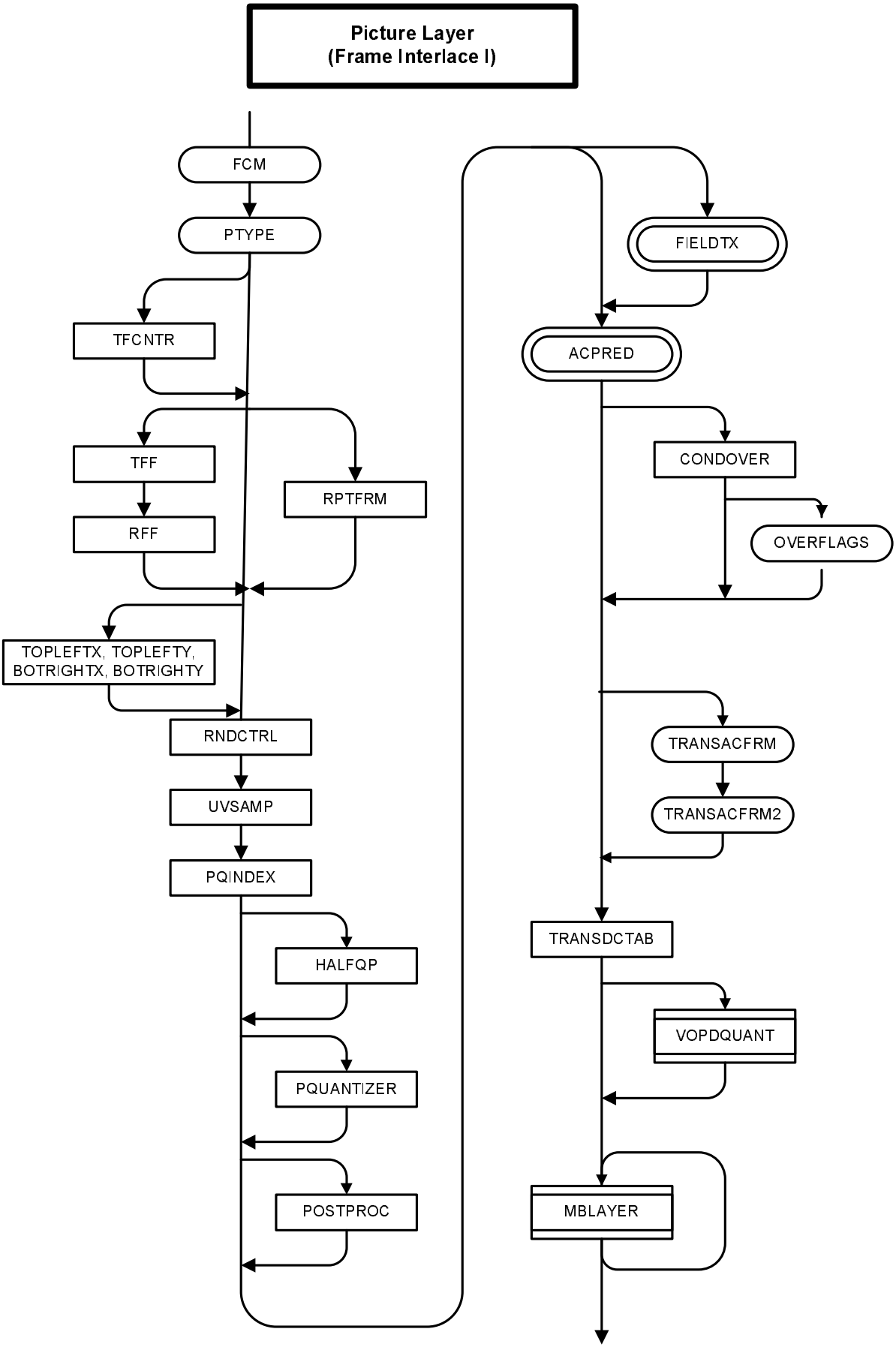

Picture Layer
(Frame Interlace I)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY,
BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
FIELDTX
ACPRED
CONDOVER
OVERFLAGS
TRANSACFRM
TRANSACFRM2
TRANSDCTAB
VOPDQUANT
MBLAYER

Figure 71: Syntax diagram for the picture layer bitstream in Interlace Frame I picture

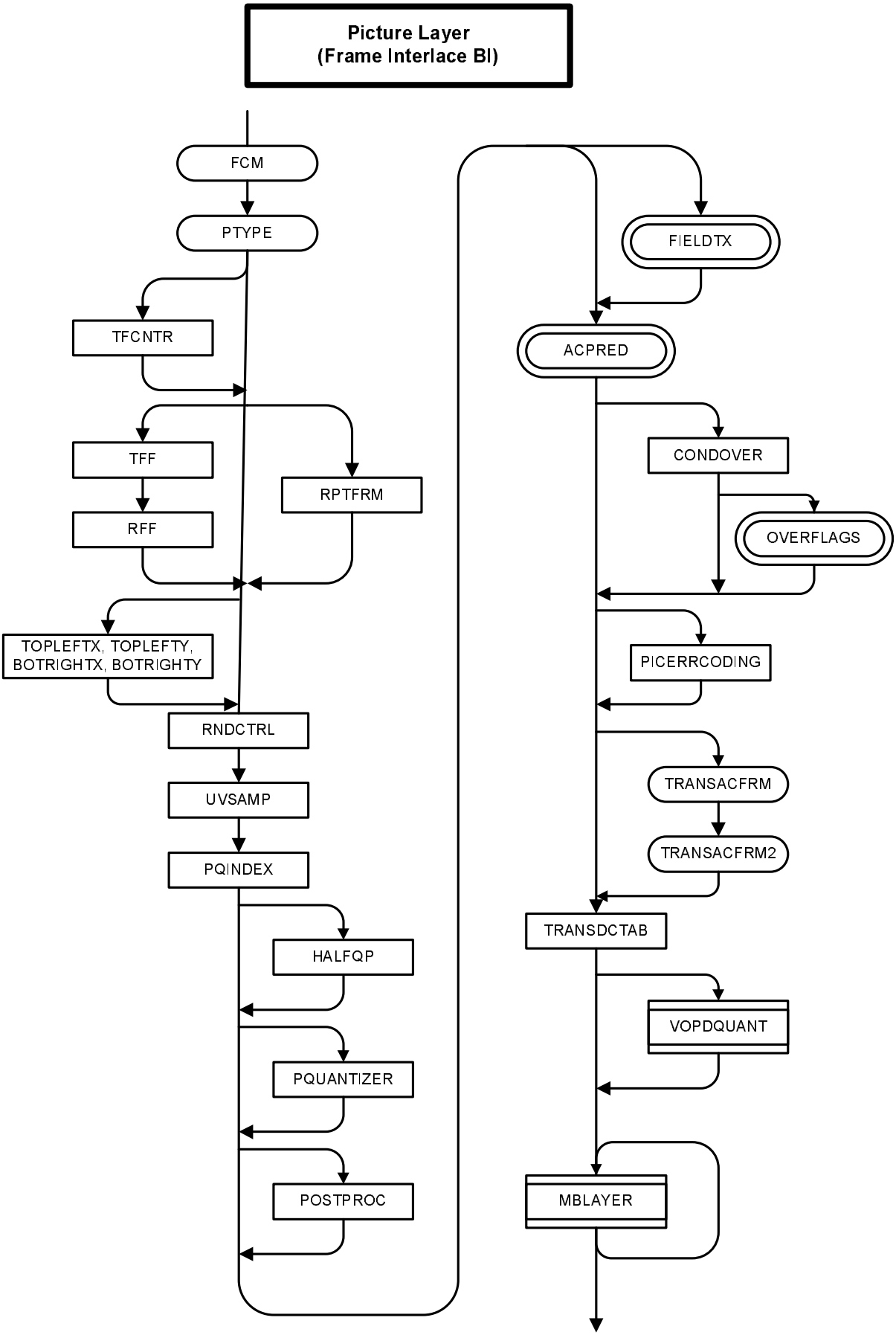

Picture Layer
(Frame Interlace BI)
FCM
PTYPE
TFCNTR
TFF
RFF
RPTFRM
TOPLEFTX, TOPLEFTY,
BOTRIGHTX, BOTRIGHTY
RNDCTRL
UVSAMP
PQINDEX
HALFQP
PQUANTIZER
POSTPROC
FIELDTX
ACPRED
CONDOVER
OVERFLAGS
PICERRCODING
TRANSACFRM
TRANSACFRM2
TRANSDCTAB
VOPDQUANT
MBLAYER

Figure 72: Syntax diagram for the picture layer bitstream in Interlace Frame BI picture

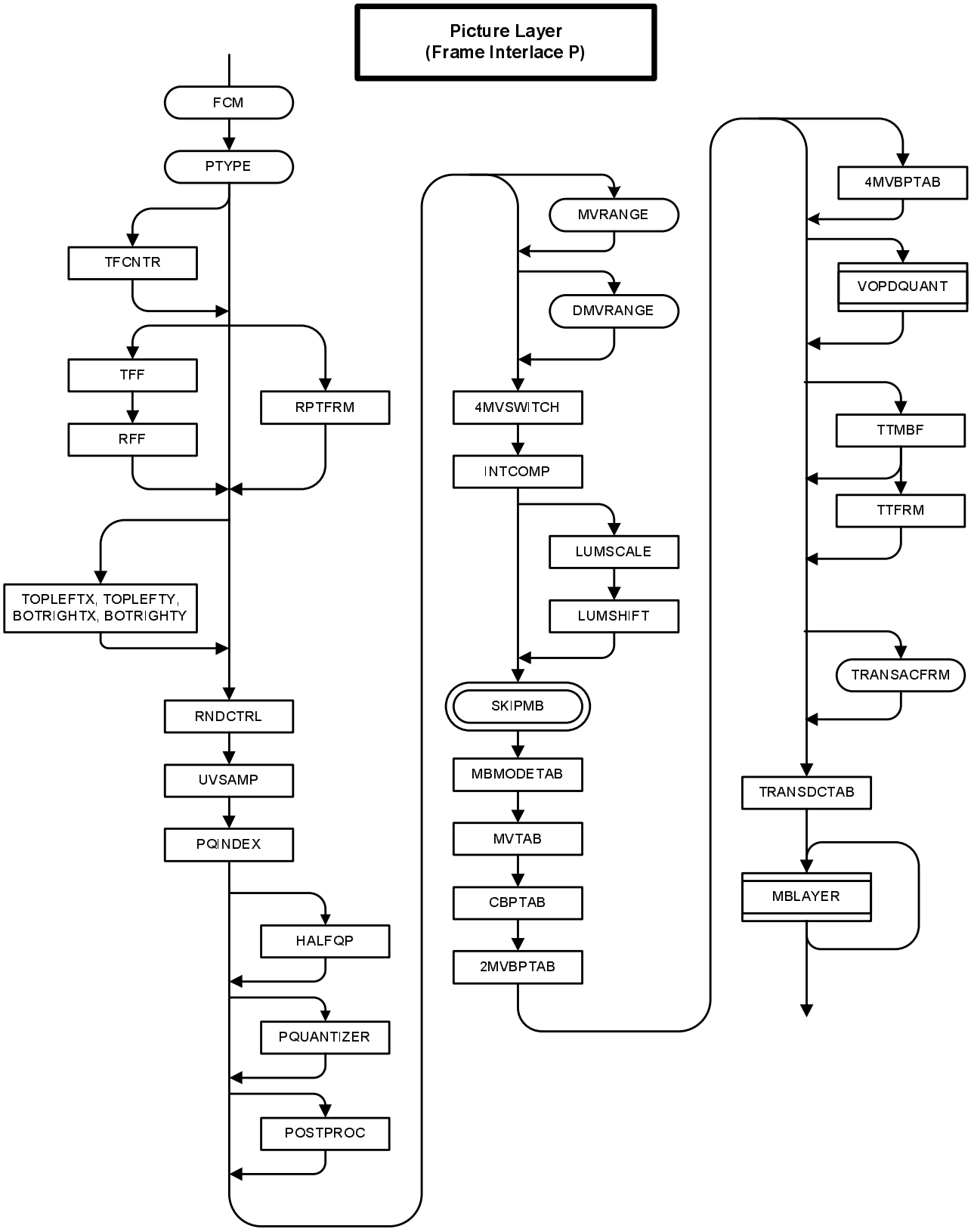


Figure 73: Syntax diagram for the picture layer bitstream in Interlace Frame P picture

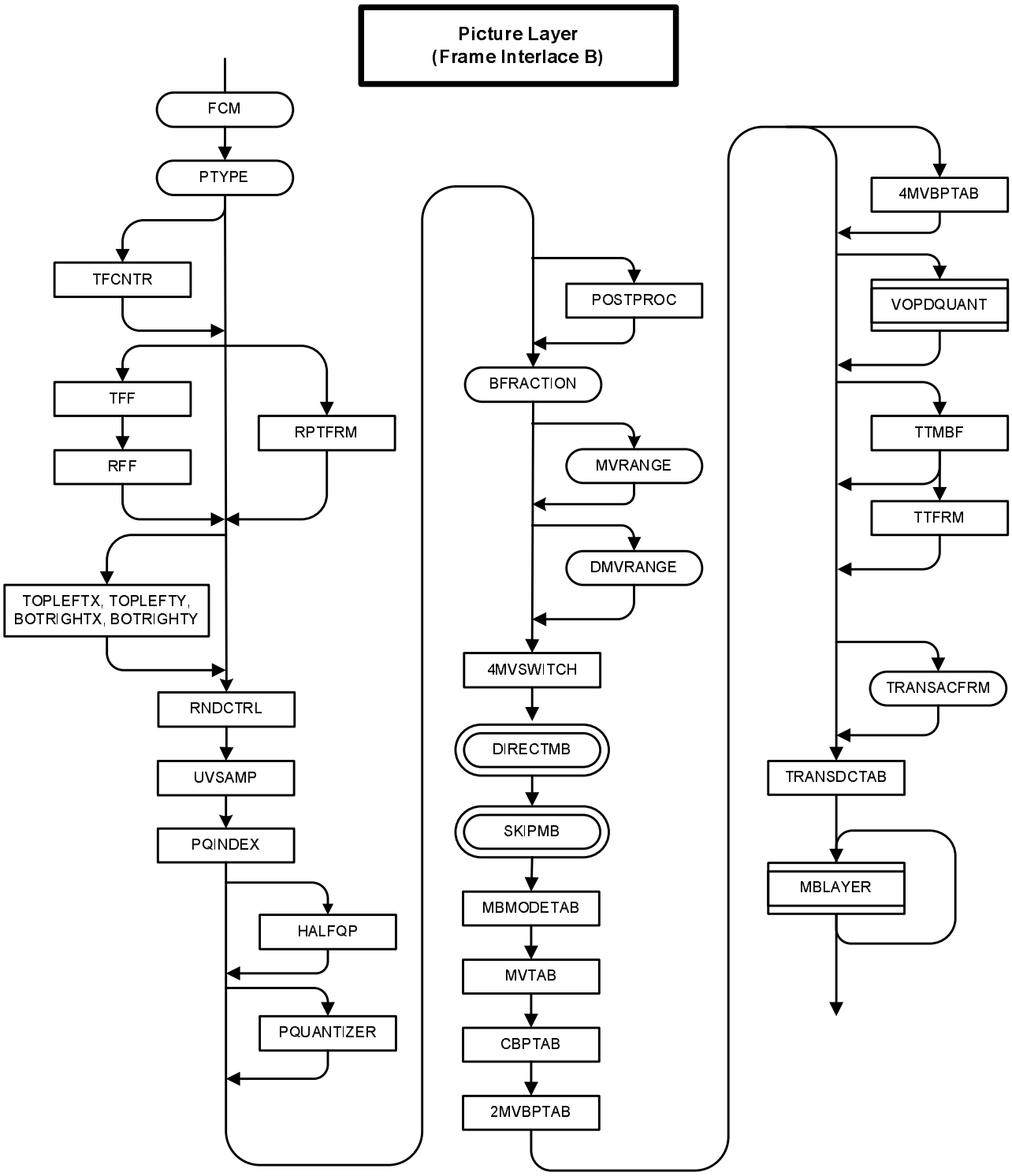


Figure 74: Syntax diagram for the picture layer bitstream in Interlace Frame B picture

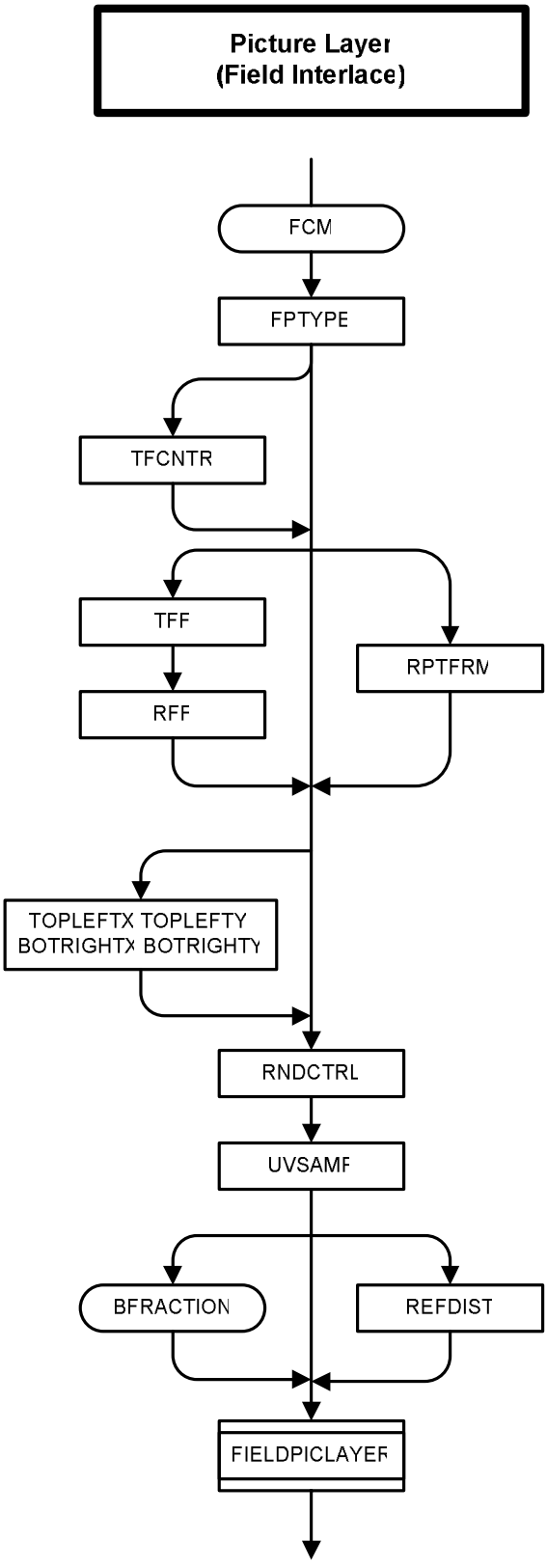


Figure 75: Syntax diagram for the picture layer bitstream in Interlace Field pictures

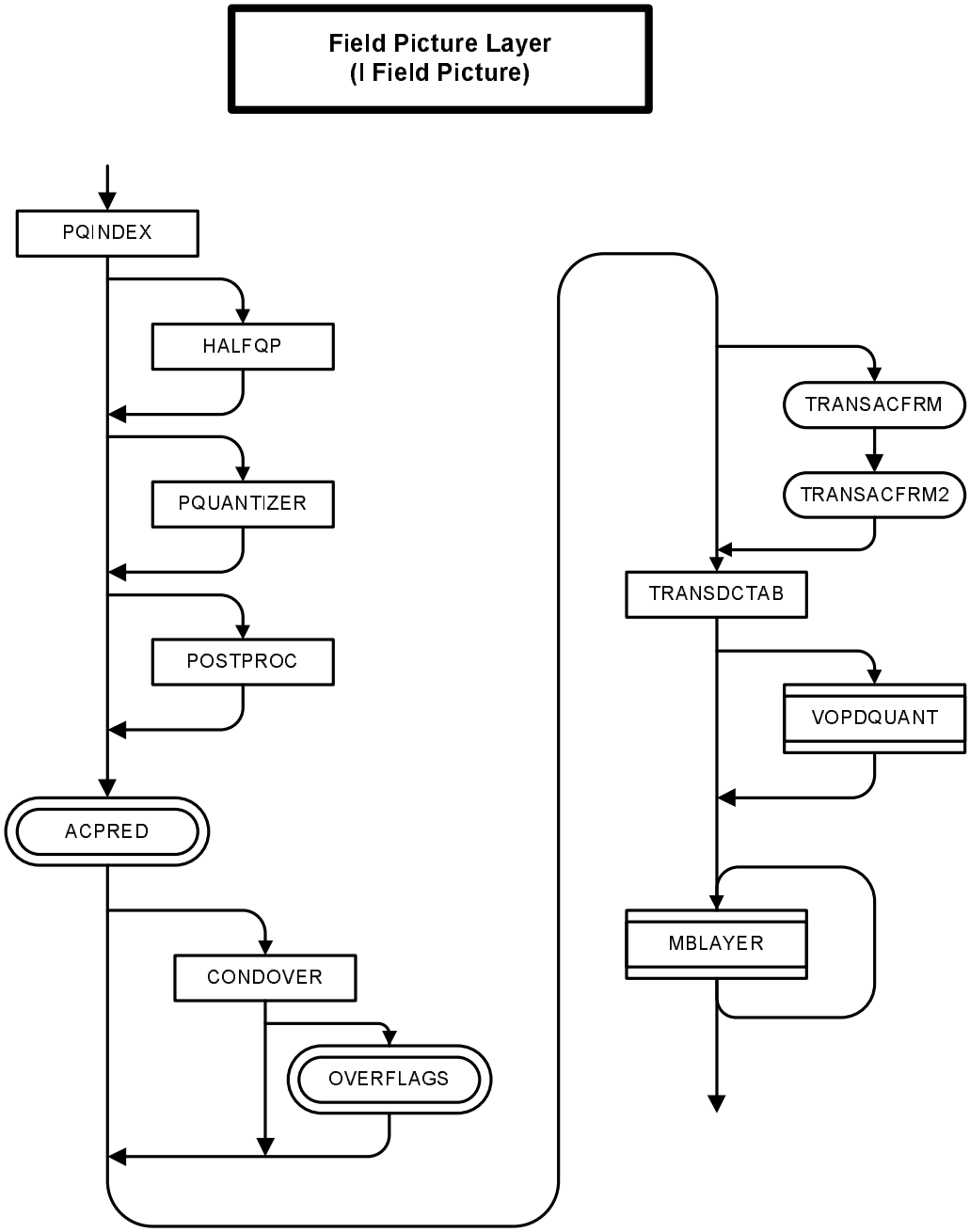


Figure 76: Syntax diagram for the field picture layer bitstream in Interlace I Field pictures

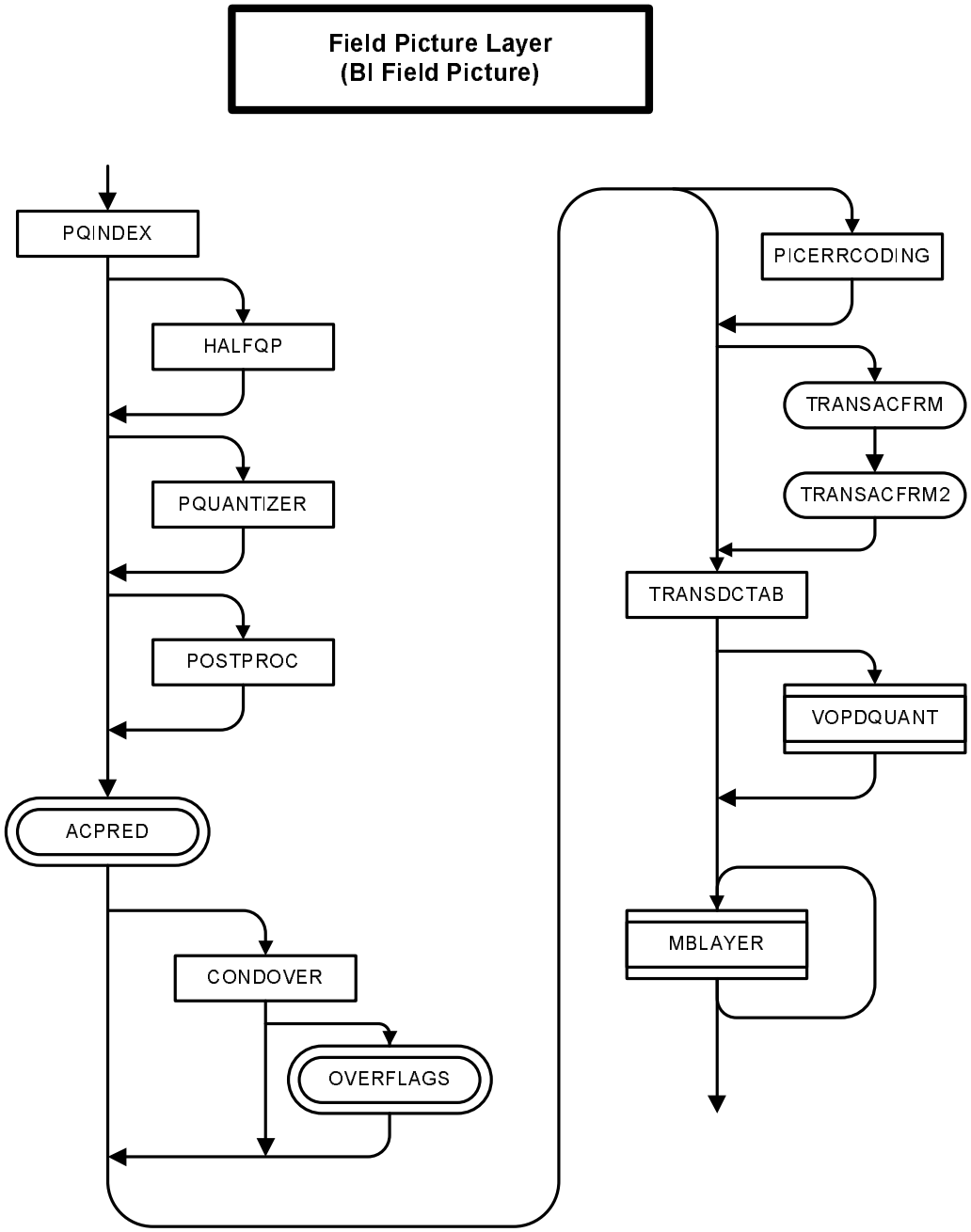


Figure 77: Syntax diagram for the field picture layer bitstream in Interlace BI Field pictures

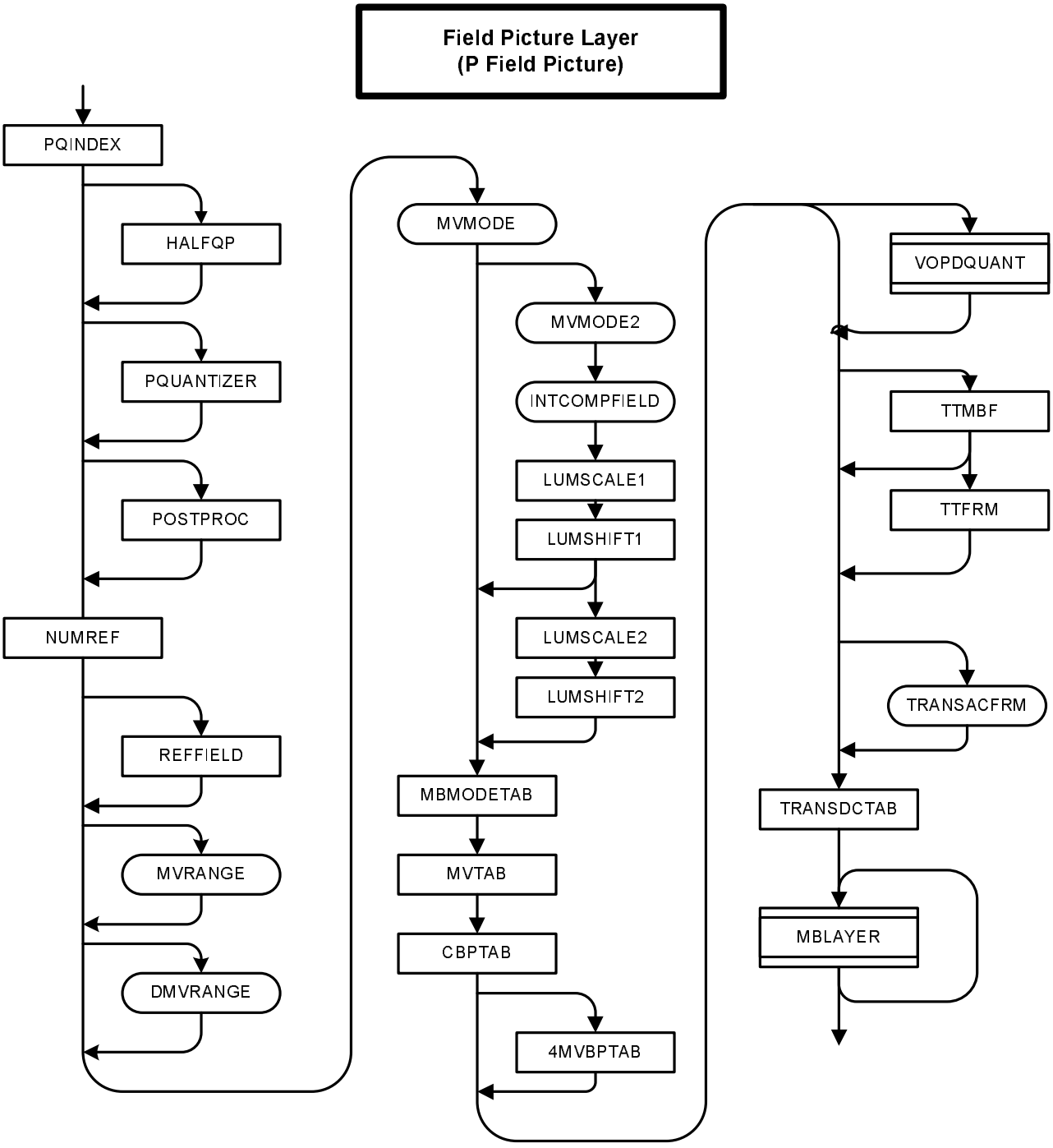


Figure 78: Syntax diagram for the field picture layer bitstream in Interlace P Field pictures

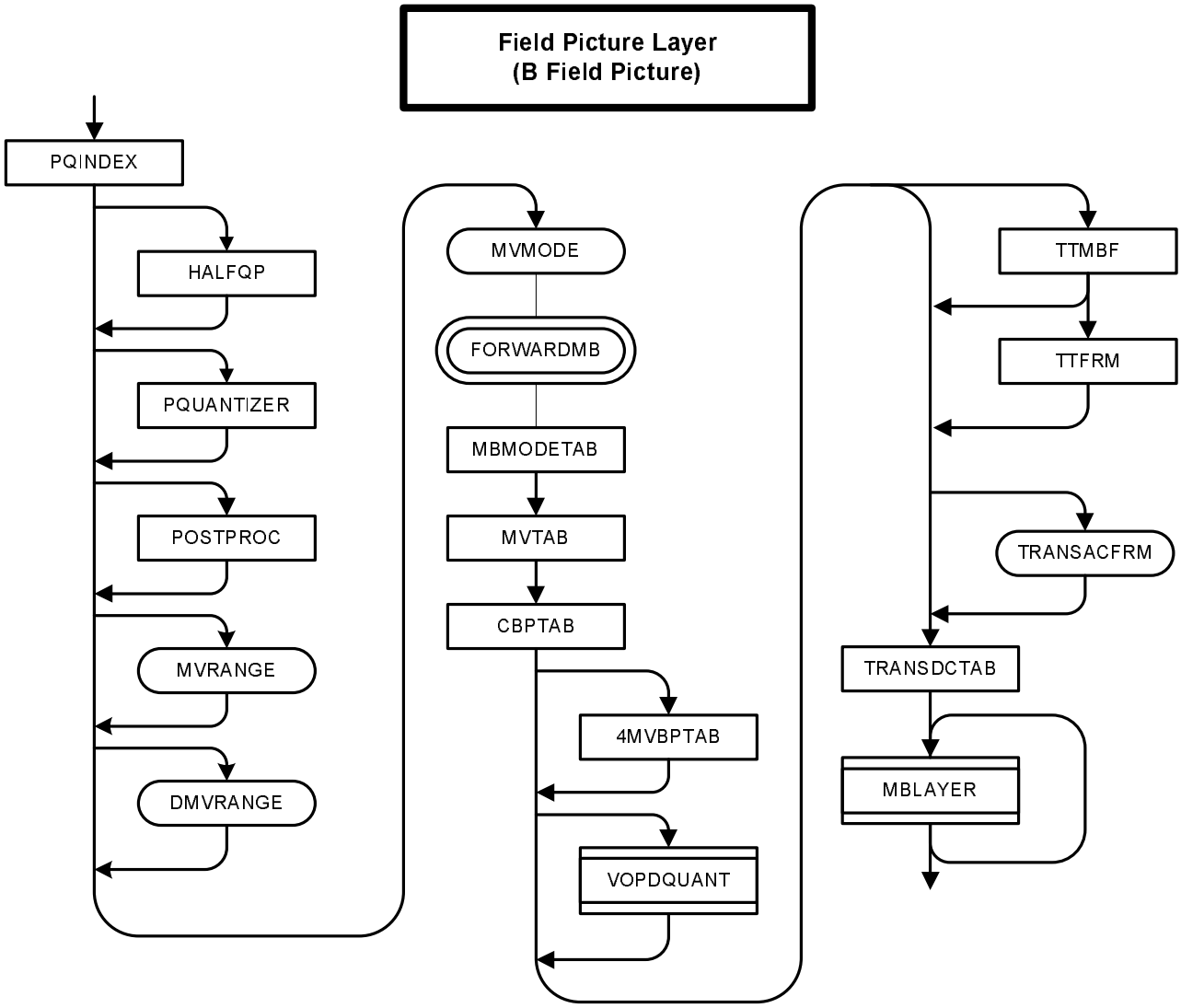


Figure 79: Syntax diagram for the field picture layer bitstream in Interlace B Field pictures

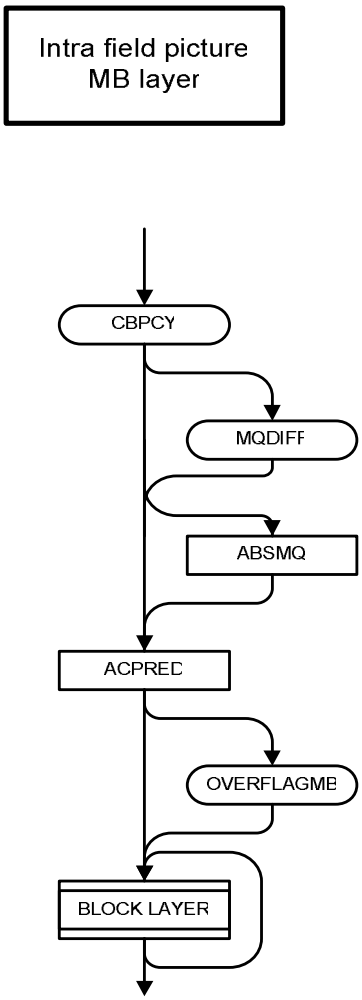


Figure 80: Syntax diagram for macroblock layer bitstream in interlace field I picture

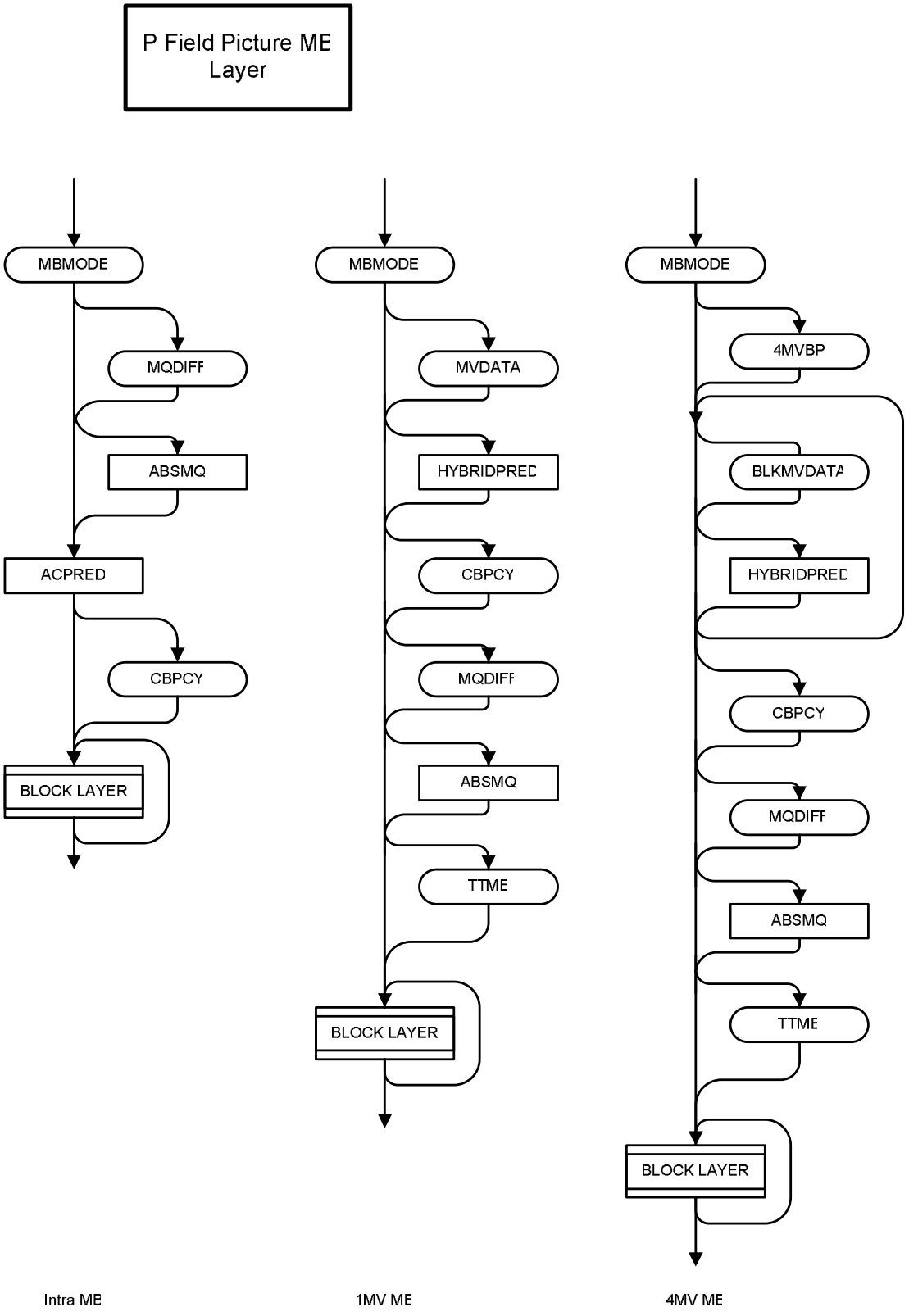


Figure 81: Syntax diagram for macroblock layer bitstream in P field picture

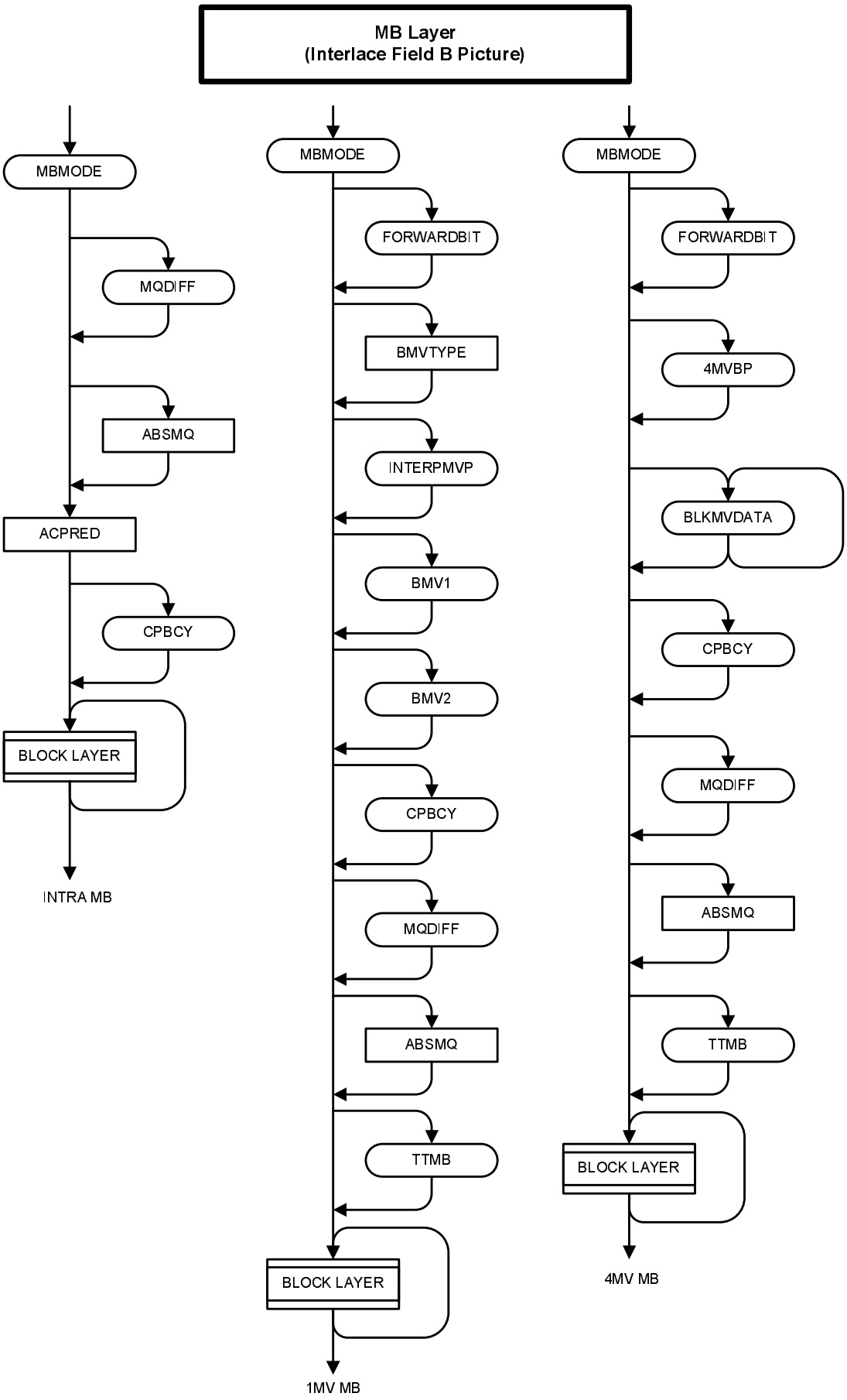

MB Layer
(Interlace Field B Picture)
MBMODE
MQDIFF
ABSMQ
ACPRED
CPBCY
BLOCK LAYER
INTRA MB
MBMODE
FORWARDBIT
BMVTYPE
INTERPMVP
BMV1
BMV2
CPBCY
MQDIFF
ABSMQ
TTMB
BLOCK LAYER
1MV MB
MBMODE
FORWARDBIT
4MVBP
BLKMVDATA
CPBCY
MQDIFF
ABSMQ
TTMB
BLOCK LAYER
4MV MB

Figure 82: Syntax diagram for macroblock layer bitstream in Field B picture

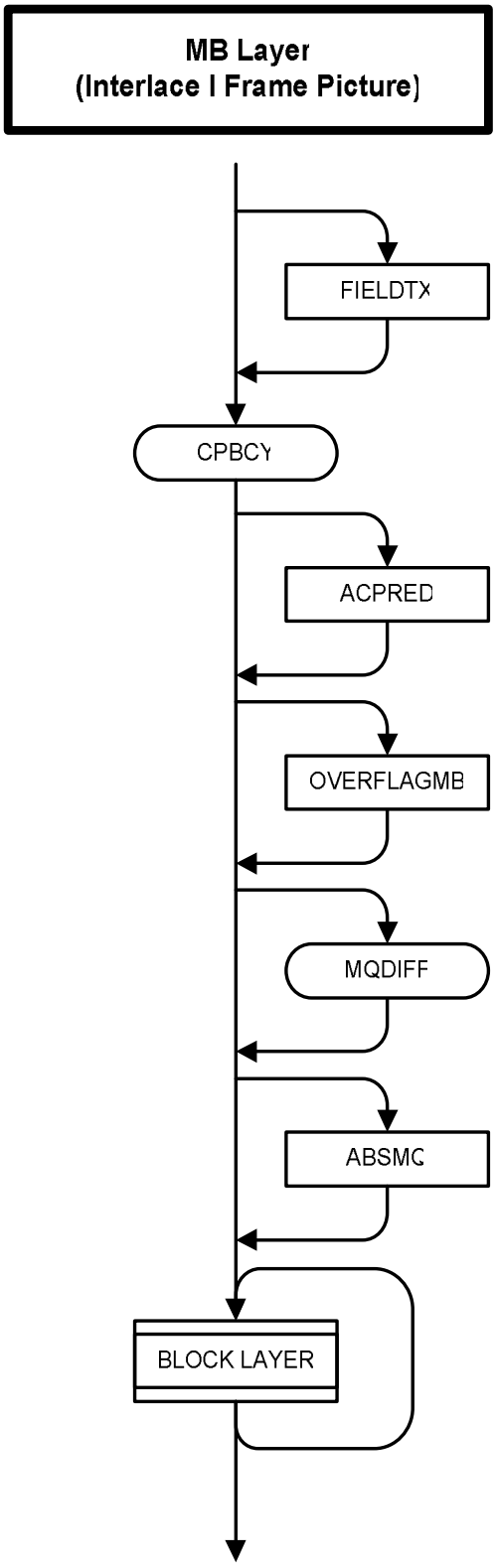


Figure 83: Syntax diagram for macroblock layer bitstream in Interlace Frame I picture

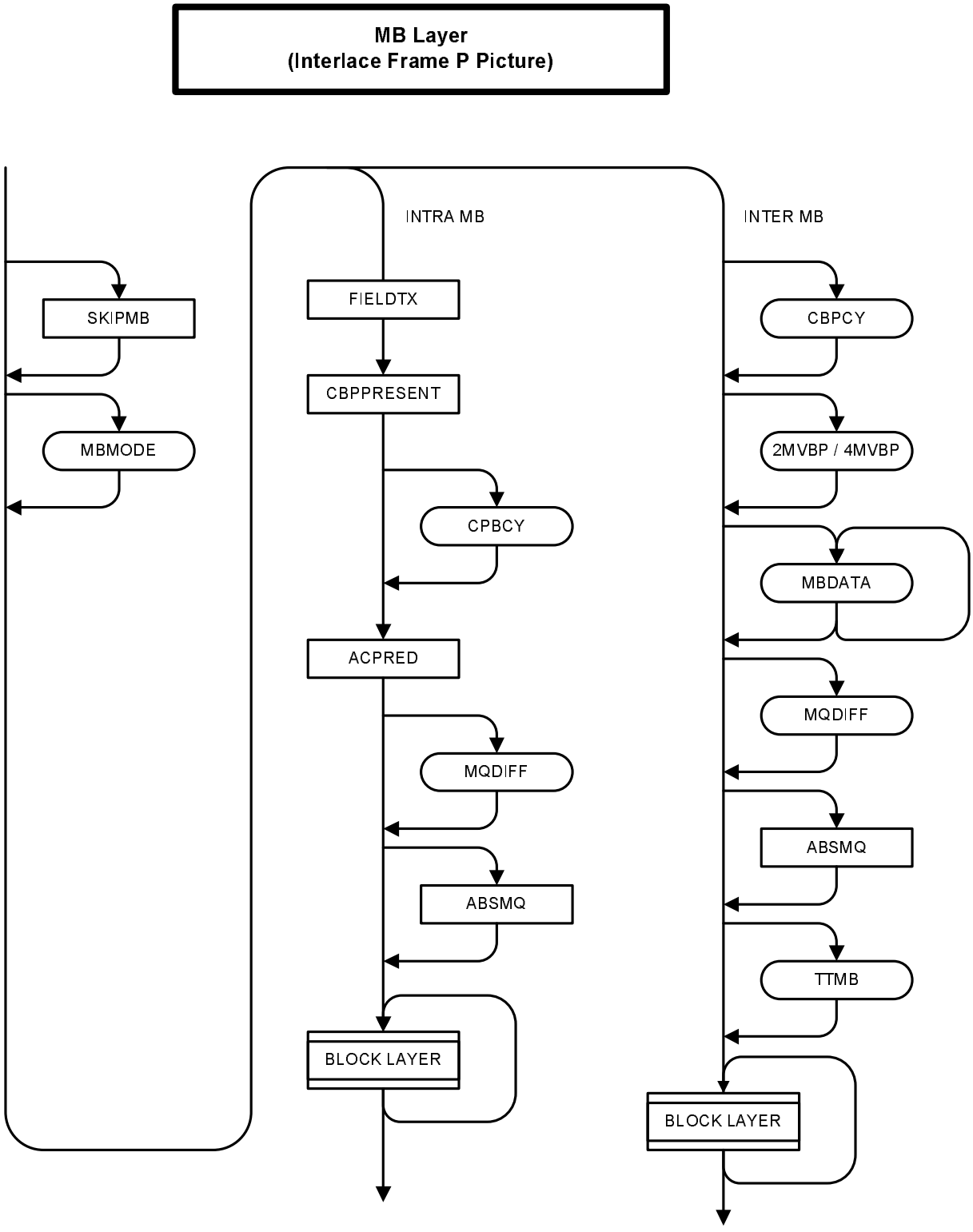


Figure 84: Syntax diagram for macroblock layer bitstream in Interlace Frame P picture

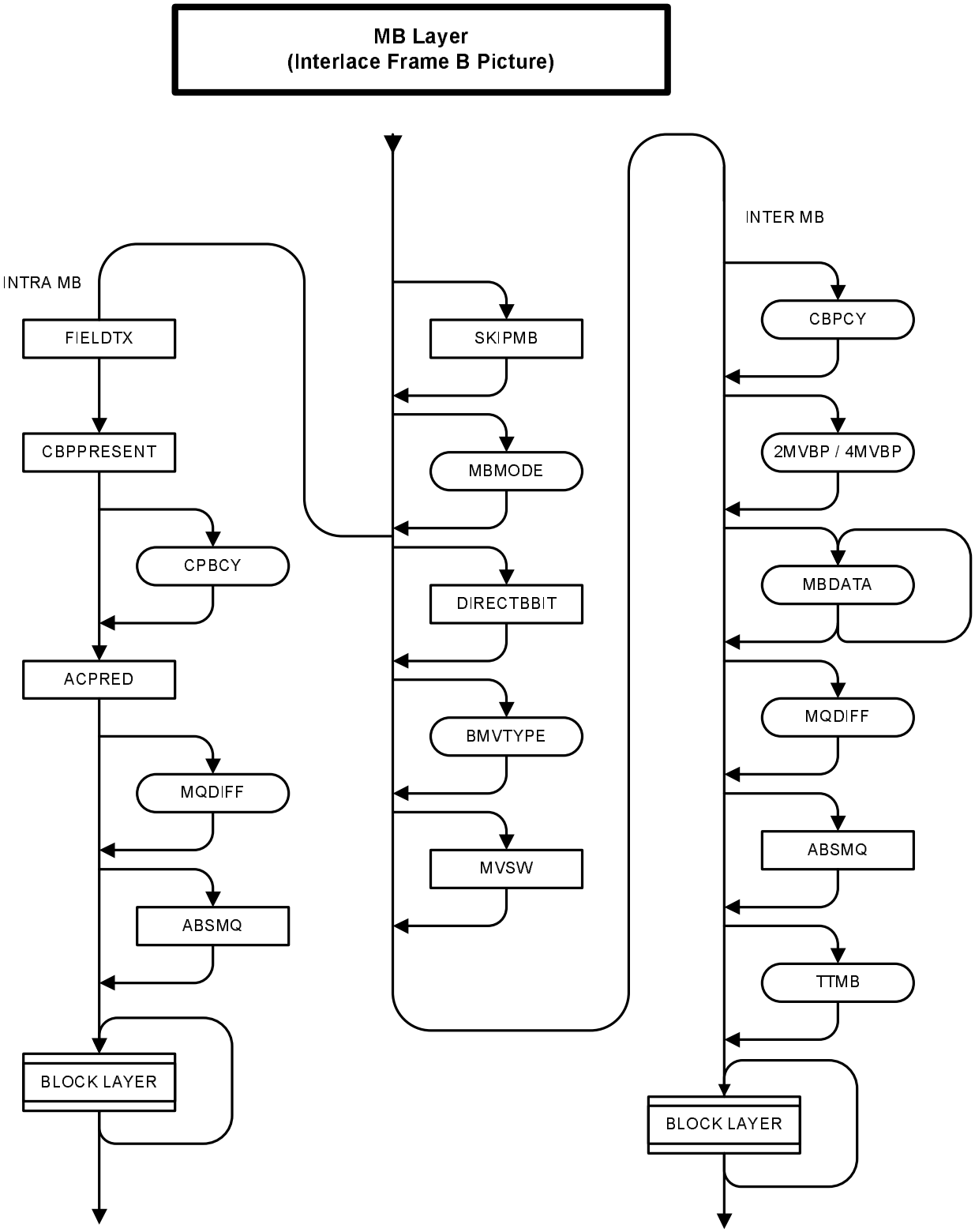


Figure 85: Syntax diagram for macroblock layer bitstream in Interlace Frame B picture

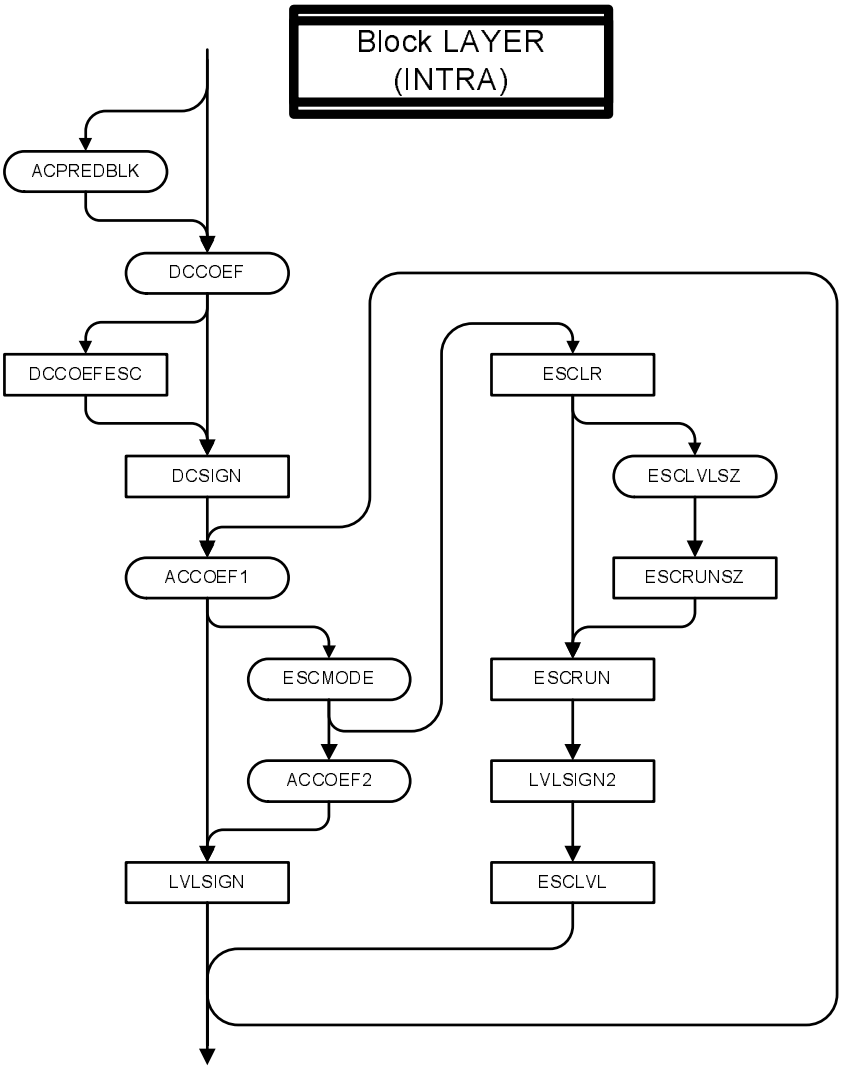


Figure 86: Intra Block Layer in Interlace Frame.

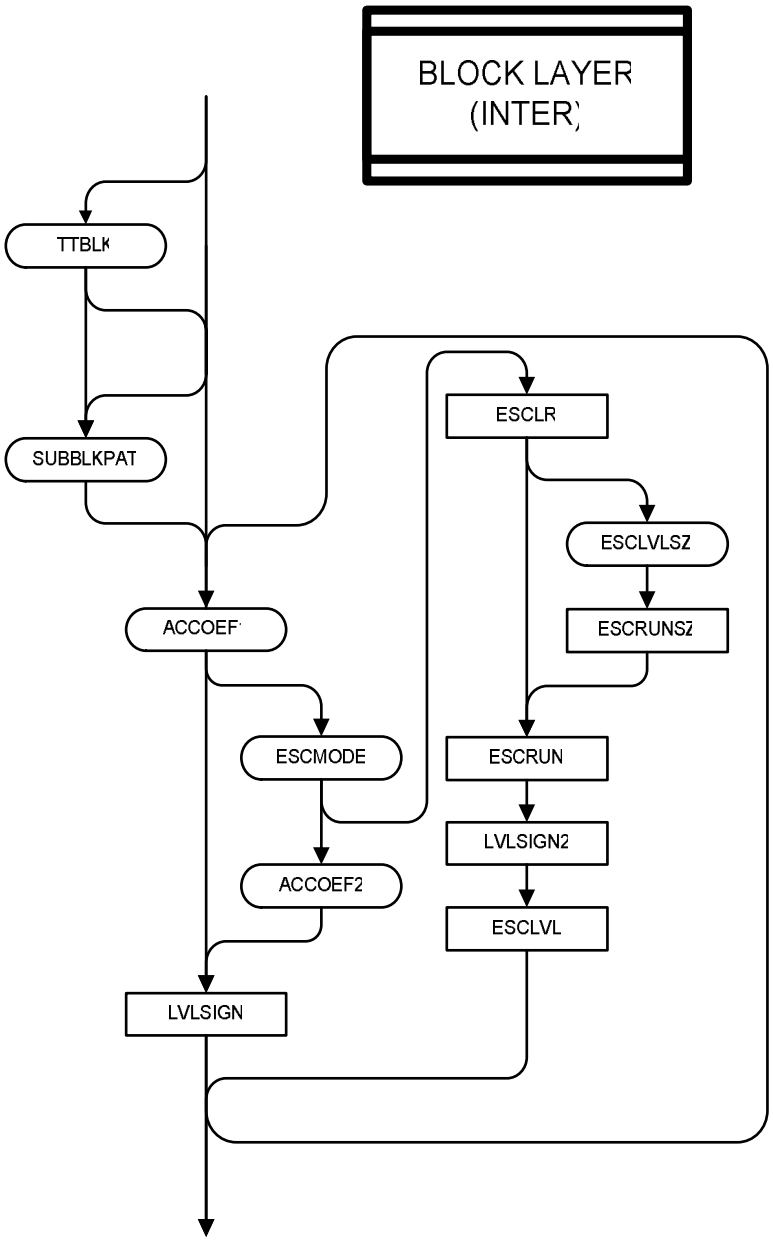


Figure 87: Inter Block Layer in Interlace Frame.

The following tables show the picture-layer syntax elements of a picture that is coded in interlace-mode.

Table 74: Interlaced Frame I picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {	8	

TFCNTR		
}		
if (BROADCAST == 1) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
FIELDTX	Bitplane	
ACPRED	Bitplane	
if (OVERLAP == 1 && ‘other conditions’) {		
CONDOVER	Variable size	

if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 75: Interlaced Frame BI picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {	8	
TFCNTR		
}		
if (BROADCAST == 1) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	

TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
FIELDTX	Bitplane	
ACPRED	Bitplane	
if (OVERLAP == 1 && ‘other conditions’) {		
CONDOVER	Variable size	
if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 76: Interlaced Frame P picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {	8	
TFCNTR		
}		
if (BROADCAST == 1) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		

if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
if (EXTENDED_DMV == 1) {		
DMVRANGE	Variable size	
}		
4MVSWITCH	1	
INTCOMP	1	
if (INTCOMP) {		
LUMSCALE	6	
LUMSHIFT	6	
}		
SKIPMB	Bitplane	
MBMODETAB	2	
MVTAB	2	
CBPTAB	3	
2MVBPTAB	2	
if (4MVSWITCH == 1) {		
4MVBPTAB	2	
}		
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		

MB LAYER( )		
}		
}		

Table 77: Interlaced Frame B picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
FCM	Variable size	
PTYPE	Variable size	
if (TFCNTRFLAG) {	8	
TFCNTR		
}		
if (BROADCAST == 1) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		

PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
BFRACTION	Variable size	
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
if (EXTENDED_DMV == 1) {		
DMVRANGE	Variable size	
}		
4MVSWITCH	1	
DIRECTMB	Bitplane	
SKIPMB	Bitplane	
MBMODETAB	2	
MVTAB	2	
CBPTAB	3	
2MVBPTAB	2	
if (4MVSWITCH == 1) {		
4MVBPTAB	2	
}		
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		
MB LAYER( )		

}		
}		

Table 78: Field Interlace Picture Layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
FCM	Variable size	
FPTYPE	3	
if (TFCNTRFLAG) {	8	
TFCNTR		
}		
if (BROADCAST == 1) {		
if (INTERLACE == 0) {		
RPTFRM	2	
}		
else {		
TFF	1	
RFF	1	
}		
}		
if (PANSCANFLAG) {		
For (i = 0; i < NUMPANSCANWIN; i++) {		
TOPLEFTX	16	
TOPLEFTY	16	
BOTRIGHTX	16	
BOTRIGHTY	16	
}		
}		
RNDCTRL	1	
UVSAMP	1	
if (FPTYPE == I/P, P/I or P/P) {		
REFDIST	Variable size	
}		
if (FPTYPE == B/B, B/BI or BI/B) {		
BFRACTION	Variable size	

}		
FIELDPICLAYER( )		
}		

Table 79: Field Interlace I Field Picture Layer bitstream for Advanced Profile

FIELDPIC LAYER( ) {	Number of bits	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
ACPRED	Bitplane	
if (OVERLAP == 1 && ‘other conditions’) {		
CONDOVER	Variable size	
if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 80: Field Interlace BI Field Picture Layer bitstream for Advanced Profile

FIELDPIC LAYER( ) {	Number of bits	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
ACPRED	Bitplane	
if (OVERLAP == 1 && 'other conditions') {		
CONDOVER	Variable size	
if (CONDOVER == 11b) {		
OVERFLAGS	Bitplane	
}		
}		
TRANSACFRM	Variable size	
TRANSACFRM2	Variable size	
TRANSDCTAB	1	
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
for ('all macroblocks') {		
MB LAYER( )		
}		
}		

Table 81: Field Interlace P Field Picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
PQINDEX	5	
if (PQINDEX <= 8) {		

HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
NUMREF	1	
if (NUMREF == 0) {		
REFFIELD	1	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
if (EXTENDED_DMV == 1) {		
DMVRANGE	Variable size	
}		
MVMODE	Variable size	
if (MVMODE == 'intensity compensation') {		
MVMODE2	Variable size	
INTCOMPFIELD	Variable size	
LUMSCALE1	6	
LUMSHIFT1	6	
if (INTCOMPFIELD = 1b) {		
LUMSCALE2	6	
LUMSHIFT2	6	
}		
}		
MBMODETAB	3	
MVTAB	2 or 3	
CBPTAB	3	
if (MVMODE =='Mixed MV') {		
4MVBPTAB	2	
}		

if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 82: Field Interlace B Field Picture layer bitstream for Advanced Profile

PICTURE LAYER( ) {	Number of bits	
PQINDEX	5	
if (PQINDEX <= 8) {		
HALFQP	1	
}		
if (QUANTIZER == 01b) {		
PQUANTIZER	1	
}		
if (POSTPROCFLAG == 1) {		
POSTPROC	2	
}		
if (EXTENDED_MV == 1) {		
MVRANGE	Variable size	
}		
if (EXTENDED_DMV == 1) {		
DMVRANGE	Variable size	
}		

MVMODE	Variable size	
FORWARDMB	Bitplane	
MBMODETAB	3	
MVTAB	2	
CBPTAB	3	
if (MVMODE ==’Mixed MV’) {		
4MVBPTAB	2	
}		
if (DQUANT != 0) {		
VOPDQUANT ( )	Variable size	
}		
if (VSTRANSFORM == 1) {		
TTMBF	1	
if (TTMBF == 1) {		
TTFRM	2	
}		
}		
TRANSACFRM	Variable size	
TRANSDCTAB	1	
for (‘all macroblocks’) {		
MB LAYER( )		
}		
}		

Table 83: Macroblock layer bitstream in Interlaced Frame I Picture

I PICTURE MB( ) {	Number of bits	
if (“FIELDTX mode == RAW”) {		<
FIELDTX	1	
}		
CBPCY	Variable size	
if (“ACPRED mode == RAW”) {		<
ACPRED	1	
}		
if (“OVERFLAGS mode == RAW” && OVERFLAGS&4) {		<
OVERFLAGMB	1	

}		
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
}		

Table 84: Macroblock layer bitstream in Interlaced Frame P Picture

P PICTURE MB( ) {	Number of bits	
if (“SKIPMB mode = SKIP_RAW”) {		
SKIPMB	1	
}		
MBMODE	Variable size	
if (‘Intra MB’) {		
FIELDTX	1	
if (CBPPRESENT) {		Inferred from MBMODE
CBPCY	Variable size	
}		
ACPRED	1	
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	

} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Intra MB */		
else { /* Inter MB */		
if (!SKIPMBBIT) {		
if (CBPPRESENT) {		
CBPCY	Variable size	
}		
if (“MVTYPEMB mode == MB_FIELD”) {		<
2MVBP	Variable size	
}		
if (“MVTYPEMB mode == MB_4MV”) {		<
4MVBP	Variable size	
}		
for (“all motion vectors”) {		
MVDATA	Variable size	
}		
if (DQUANTFRM && CBPCY) {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		

}		
} /* if (DQPROFILE …*/		
} /* if (DQUANTFRM…*/		
} /* if (!SKIPMBBIT) */		
if (!TTMBF && CBPCY) {		
TTMB	Variable size	
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Inter MB */		
}		

Table 85: Macroblock layer bitstream in Interlaced Frame B Picture

B PICTURE MB( ) {	Number of bits	
if (“SKIPMB mode = SKIP_RAW”) {		<
SKIPMB	1	
}		
MBMODE	Variable size	
if (Intra MB’) {		Inferred from MBMODE
FIELDTX	1	
if (CBPPRESENT) {		// inferred from MBMODE
CBPCY	Variable size	
}		
ACPRED	1	
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	

if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Intra MB */		
else { /* Inter MB */		
if (“DIRECTMB mode = SKIP_RAW”) {		<
DIRECTBBIT	1	
}		
if (!DIRECTBBIT) {		
BMVTYPE	Variable size	
}		
if (“MVTYPEMB mode == MB_FIELD” && BMVTYPE != INTERPOLATE) {		<
MVSW	1	
}		
if (CBPPRESENT) {		
CBPCY	Variable size	
}		
if (MBTYPE != DIRECT) {		
if ((“MVTYPEMB mode == MB_FIELD” && MBTYPE != INTERPOLATE) || (“MVTYPEMB mode == MB_1MV” && MBTYPE == INTERPOLATE)) {		<
2MVBP	Variable size	
}		
} else if (!(“MVTYPEMB mode == MB_1MV” && MBTYPE != INTERP)) {		<
4MVBP	Variable size	
}		
for (“all motion vectors”) {		
MVDATA	Variable size	
}		

if (DQUANTFRM && CBPCY) {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
if (!TTMBF && CBPCY) {		
TTMB	Variable size	
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Inter MB */		
}		

Table 86: Macroblock layer bitstream in Interlaced Field I picture

I PICTURE MB( ) {	Number of bits	
CBPCY	Variable size	
if (‘ACPRED mode == RAW’) {		<
ACPRED	1	
}		
if (CONDOVER == 11b && OVERFLAGS == ‘raw mode’) {		<
OVERFLAGMB	1	
}		
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	

} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
for (‘all coded blocks in MB’) {		
BLOCK( )		
}		
}		

Table 87: Macroblock layer bitstream in Interlaced Field P Picture

P PICTURE MB( ) {	Number of bits	
MBMODE	Variable size	
if (‘Intra MB’) {		
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
}		
}		
ACPRED	1	
if (‘CBP is Present’) {		inferred from MBMODE
CBPCY	Variable size	

}		
for (“all coded blocks in MB”) {		Inferred from CBPCY
BLOCK( )		
}		
} /* Intra MB */		
else { /* Inter MB */		
if (‘1MV MB’) {		Inferred from MBMODE
if (‘MV Data is Present’) {		Inferred from MBMODE
MVDATA	Variable size	
}		
if (‘Hybrid MV is Present’) {		
HYBRIDPRED	1	
}		
}		
else { // 4MV Macroblock		
if (‘4MVBP is Present’) {		Inferred from MBMODE
4MVBP	Variable Size	
}		
for (‘all Y blocks’) {		
if (‘BLKMVDATA is present’) {		
BLKMVDATA	Variable Size	
}		
if (‘HYBRIDPRED is present’) {		
HYBRIDPRED		
}		
} // all Y blocks		
}		
if (‘CBP is Present’) {		Inferred from MBMODE
CBPCY	Variable size	
}		

if (DQUANTFRM && CBPCY) {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
} /* if (DQPROFILE …*/		
} /* if (DQUANTFRM…*/		
if (!TTMBF && CBPCY) {		
TTMB	Variable size	
}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Inter MB */		
}		

Table 88: Macroblock layer bitstream in Interlaced Field B Picture

P PICTURE MB( ) {	Number of bits	
MBMODE	Variable size	
if (‘Intra MB’) {		inferred from MBMODE
if (DQUANTFRM**)** {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		

}		
}		
}		
ACPRED	1	
if (‘CBP is Present’) {		inferred from MBMODE
CBPCY	Variable size	
}		
for (“all coded blocks in MB”) {		Inferred from CBPCY
BLOCK( )		
}		
} /* Intra MB */		
else { /* Inter MB */		
If (‘FORWARDMB Mode’ = = ‘RAW’) {		
FORWARDBIT	1	
}		
if (‘1MV MB’) {		Inferred from MBMODE
If (‘MB Type’ != ‘FORWARD’) {		Inferred from FORWARDBIT or corresponding bitplane
BMVTYPE	Variable Size	0 = backward, 10 = Direct, 11 = Interpolate
if (‘BMV Type’ == ‘INTERPOLATE’) {		Inferred from FORWARDBIT and BMVTYPE
INTERPMVP	1	
}		
}		
if (‘BMV Type’ != ‘DIRECT’ && ‘Block MV Data is Present’) {		Presence of Block MV Data is Inferred from MBMODE
BMV1	Variable size	
}		
If (‘BMV Type’ == ‘INTERPOLATE’ && INTERPMVP) {		

BMV2	Variable size	
}		
}		
else { // 4MV Macroblock		
BMVTYPE is assumed to be backward		
if (‘4MVBP is Present’) {		Inferred from MBMODE
4MVBP	Variable Size	
}		
for (‘all Y blocks’) {		
if (‘BLKMVDATA is present’) {		Inferred from MBMODE
BLKMVDATA	Variable Size	
}		
} // all Y blocks		
}		
if (‘CBP is Present’) {		Inferred from MBMODE
CBPCY	Variable size	
}		
if (DQUANTFRM && CBPCY) {		
if (DQPROFILE == ‘all macroblocks‘) {		
if (DQBILEVEL){		
MQDIFF	1	
} else {		
MQDIFF	3	
if (MQDIFF == 7) {		
ABSMQ	5	
}		
}		
} /* if (DQPROFILE …*/		
} /* if (DQUANTFRM…*/		
if (!TTMBF && CBPCY) {		
TTMB	Variable size	

}		
for (“all coded blocks in MB”) {		
BLOCK( )		
}		
} /* Inter MB */		
}		

9.1.1 Picture layer

Data for each picture consists of a picture header followed by data for the macroblock layer. The bitstream elements that make up the interlace frame headers for I, P and B picture types are shown in Figure 71, Figure 73 and Figure 74 respectively. The bitstream elements that make up the frame header for field pictures is shown in Figure 75. The bitstream elements that make up the field picture headers for I, P and B pictures are shown in Figure 76, Figure 78 and Figure 79 respectively. The following sections describe the bitstream elements in the picture and field picture headers.

9.1.1.1 Frame Coding Mode (FCM) (Variable size)

FCM in interlace frame and field picture headers is the same as described in section 7.1.1.2.

9.1.1.2 Field Picture Type (FPTYPE) (3 bits)

FPTYPE is a 3 bit syntax element present in the picture header for interlace field pictures. FPTYPE is decoded according to Table 89.

Table 89: Field Picture Type FLC

FPTYPE FLC	First Field Picture Type	Second Field Picture Type
000	I	I
001	I	P
010	P	I
011	P	P
100	B	B
101	B	BI
110	BI	B
111	BI	BI

9.1.1.3 Picture Type (PTYPE) (Variable size)

PTYPE in interlace frame pictures is the same as described in section 7.1.1.9 for progressive pictures.

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

TFCNTR in interlace frame and field picture headers is the same as described in section 7.1.1.1.

9.1.1.5 Top Field First (TFF) (1 bit)

TFF in interlace frame and field picture headers is the same as described in section 7.1.1.3.

9.1.1.6 Repeat First Field (RFF) (1 bit)

RFF in interlace frame and field picture headers is the same as described in section 7.1.1.4.

9.1.1.7 Repeat Frame Count (RPTFRM) (2 bits)

RPTFRM in interlace frame and field picture headers is the same as described in section 7.1.1.5.

9.1.1.8 B Picture Fraction (BFRACTION)(Variable size)

BFRACTION in interlace frame and field picture headers is the same as described in section 7.1.1.10.

9.1.1.9 Rounding Control Bit (RNDCTRL)(1 bit)

RNDCTRL is a 1 bit syntax element that is present in progressive advanced profile picture headers (I, P, B). The flag is used to indicate the type of rounding used for the current frame. If RNDCTRL = 1, the parameter R which controls rounding is set to 1. Otherwise, R is set to zero. See Section 8.3.7 for more details on the effect of R on rounding.

9.1.1.10 UV Sampling Format (UVSAMP)(1 bit)

UVSAMP is a 1 bit syntax element that is only present in advanced profile picture headers (I, P, B), when the sequence level field INTERLACE is 1. The flag is used to indicate the type of chroma subsampling used for the current frame. If UVSAMP = 1, then progressive subsampling of the chroma is used, otherwise, interlace subsampling of the chroma is used. This syntax element does not affect decoding of the bitstream.

9.1.1.11 P Reference Distance (REFDIST) (Variable size)

REFDIST is a variable sized syntax element present in interlace field picture headers, if the picture type is not one of the following types: B/B, B/BI, BI/B, BI/BI. This element indicates the number of frames between the current frame and the reference frame. Table 90 shows the VLC codewords used to encode the REFDIST values.

Table 90: REFDIST VLC Table

Reference Frame Distance	VLC Codeword (Binary)	VLC Size
0	00	2
1	01	2
2	10	2
N	11[(N-3) 1s]0	N

The last row in Table 90 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 is the reference frame distance. The last bit in the codeword is 0. For example:

N = 3, VLC Codeword = 110, VLC Size = 3

N = 4, VLC Codeword = 1110, VLC Size = 4

N = 5, VLC Codeword = 11110, VLC Size = 5

9.1.1.12 Picture Quantizer Index (PQINDEX) (5 bits)

PQINDEX in interlace pictures is the same as described in section 7.1.1.15.

9.1.1.13 Half QP Step (HALFQP) (1 bit)

HALFQP in interlace pictures is the same as described in section 7.1.1.16.

9.1.1.14 Picture Quantizer Type (PQUANTIZER) (1 bit)

PQUANTIZER in interlace pictures is the same as described in section 7.1.1.17.

9.1.1.15 Post Processing (POSTPROC)(2 bits)

POSTPROC is a 2 bits syntax element that occurs in all pictures for advanced profile when the sequence level flag POSTPROCFLAG is set to 1. It is the same as described in section 7.1.1.36 for progressive content.

9.1.1.16 AC Prediction (ACPRED)(Variable size)

The ACPRED syntax element is only present in interlace field I and interlace frame I pictures and it is the same as described in section 7.1.1.33.

9.1.1.17 Conditional Overlap Flag (CONDOVER) (Variable size)

CONDOVER is present only in frame / field I pictures and it is the same as described in section 7.1.1.34.

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

OVERFLAGS is present only in frame / field I pictures and it is the same as described in section 7.1.1.35.

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

TRANSACFRM in interlace pictures is the same as described in section 7.1.1.37.

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

TRANSACFRM2 in interlace pictures is the same as described in section 7.1.1.38.

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

TRANSDCTAB in interlace pictures is the same as described in section 7.1.1.39.

9.1.1.22 Macroblock Quantization (VOPDQUANT) (Variable size)

VOPDQUANT in interlace pictures is the same as described in section 7.1.1.29.

9.1.1.23 Number of Reference Pictures (NUMREF) (1 bit)

NUMREF is a 1 bit syntax element present only in interlace P field pictures headers.

9.1.1.24 Reference Field Picture Indicator (REFFIELD) (1 bit)

REFFIELD is a 1 bit syntax element present in interlace P field picture headers if NUMREF = 0.

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

MVRANGE is a variable-sized syntax element present only in field / frame P and B picture headers. It is the same as described in section 7.1.1.18.

9.1.1.26 Extended Differential MV Range Flag (DMVRANGE) (Variable size)

DMVRANGE is a variable sized syntax element present in field / frame P and B pictures if the sequence level syntax element EXTENDED_DMV = 1. The following table is used to encode the DMVRANGE element. See section 10.3.4.5.1 for a description of how the DMVRANGE value is used.

Table 91: DMVRANGE VLC Table

Extended Horizontal Differential MV Range	Extended Vertical Differential MV Range	VLC Codeword (Binary)	VLC Size
No	No	0	1

Yes	No	10	2
No	Yes	110	3
Yes	Yes	111	3

9.1.1.27 Skipped Macroblock Decoding (SKIPMB)(Variable size)

The SKIPMB syntax element is only present in interlace frame P and interlace frame B pictures. The interlace frame P picture layer contains the SKIPMB syntax element which is a bitplane coded syntax element that indicates 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 in raster scan order from upper left to lower right. Refer to section 7.2 for a description of the bitplane coding. A value of 0 indicates that the macroblock is not skipped. A value of 1 indicates that the macroblock is coded as skipped. A skipped status for a macroblock in interlace frame P picture means that the decoder shall treat this macroblock as 1 MV with the motion vector differential being zero and the coded block pattern being zero. In addition, no other information is expected to follow for this macroblock.

9.1.1.28 4 Motion Vector Switch (4MVSWITCH) (Variable size or 1 bit)

The 4MVSWITCH syntax element is a 1-bit present in interlace frame P and B picture headers. If 4MVSWITCH is set to zero, the macroblocks in the picture have only one motion vector or two motion vectors, depending on whether the macroblock has been frame-coded or field-coded respectively. If 4MVSWITCH is set to 1, there may be 1, 2 or 4 motion vectors per macroblock. See section 10.7.2 for more details on the use of 4MVSWITCH in decoding.

9.1.1.29 Motion Vector Mode (MVMODE) (Variable size or 1 bit)

The MVMODE syntax element is a variable size syntax element that is present in interlace field P and B picture headers. It is the same as the corresponding MVMODE syntax element described for progressive pictures in section 7.1.1.22.

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

The MVMODE2 syntax element is present in interlace field P and interlace field B picture headers, and only if MVMODE signals intensity compensation.

9.1.1.31 Intensity Compensation (INTCOMP)(1 bit)

INTCOMP is a 1 bit syntax element that is only present in interlace frame P headers. INTCOMP is used to indicate whether intensity compensation mode is used in the current frame.

9.1.1.32 Intensity Compensation Field (INTCOMPFIELD)(Variable size)

INTCOMPFIELD is a variable sized syntax element present in interlace field P field picture headers. INTCOMPFIELD is used to indicate which reference field undergoes intensity compensation.

Table 92: INTCOMPFIELD VLC Table

INTCOMPFIELD VLC	Intensity Compensation Applied to:
1	Both fields
00	First field
01	Second field

9.1.1.33 Luminance Scale (LUMSCALE)(6 bits)

The LUMSCALE syntax element is present in P interlace frame pictures if the frame header syntax element INTCOMP = 1. Refer to section 8.3.8 for a description of intensity compensation.

9.1.1.34 Luminance Shift (LUMSHIFT)(6 bits)

The LUMSHIFT syntax element is present in P interlace frame pictures if the frame header syntax element INTCOMP = 1. Refer to section 8.3.8 for a description of intensity compensation.

9.1.1.35 Field Picture Luminance Scale 1 (LUMSCALE1)(6 bits)

The LUMSCALE1 syntax element is present in the P interlace field picture header if the field picture header syntax element MVMODE signals intensity compensation. If the INTCOMPFIELD element is '1' or '00' then LUMSCALE1 is applied to the first field. Otherwise it is applied to the second field.

9.1.1.36 Field Picture Luminance Shift 1 (LUMSHIFT1)(6 bits)

The LUMSHIFT1 syntax element is present in the P interlace field picture header if the field picture header syntax element MVMODE signals intensity compensation. If the INTCOMPFIELD element is '1' or '00' then LUMSHIFT1 is applied to the first field. Otherwise it is applied to the second field.

9.1.1.37 Field Picture Luminance Scale 2 (LUMSCALE2)(6 bits)

The LUMSCALE2 syntax element is present in the P interlace field picture header if the field picture header syntax element MVMODE signals intensity compensation and INTCOMPFIELD = '01'. LUMSCALE2 is applied to the second field.

9.1.1.38 Field Picture Luminance Shift 2 (LUMSHIFT2)(6 bits)

The LUMSHIFT2 syntax element is present in the P interlace field picture header if the field picture header syntax element MVMODE signals intensity compensation and INTCOMPFIELD = '01'. LUMSHIFT2 is applied to the second field.

9.1.1.39 B Frame Direct Mode Macroblock Bit Syntax Element (DIRECTMB)(Variable size)

The DIRECTMB syntax element is only present in interlace frame B pictures. It is the same as described in 7.1.1.21.

9.1.1.40 B Field Forward Mode Macroblock Bit Syntax Element (FORWARDMB)(Variable size)

The FORWARDMB syntax element is only 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. The 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. Refer to section 8.7 for a description of the bitplane coding method.

9.1.1.41 Macroblock Mode Table (MBMODETAB) (2 or 3 bits)

The MBMODETAB syntax element is a fixed length field that is present in interlace frame P, frame B, field P and field B pictures.

For field P and field B pictures, MBMODETAB is a 3 bit value that indicates which one of the eight Huffman tables is used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer. There are two sets of eight Huffman tables and the set that is being used depends on whether 4MV is used or not as indicated by the MVMODE flag.

Table 93: **MBMODETAB code-table for interlace field P, B pictures**

FLC	Macroblock Mode Huffman Table
000	Huffman Table 0
001	Huffman Table 1

010	Huffman Table 2
011	Huffman Table 3
100	Huffman Table 4
101	Huffman Table 5
110	Huffman Table 6
111	Huffman Table 7

For frame P and frame B pictures, MBMODETAB is a 2 bit value that indicates which one of the four Huffman tables in used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer. There are two sets of four Huffman tables and the set that is being used depends on whether 4MV is used or not as indicated by the 4MVSWITCH flag.

Table 94: **MBMODETAB code-table for interlace frame P, B pictures**

FLC	Macroblock Mode Huffman Table
00	Huffman Table 0
01	Huffman Table 1
10	Huffman Table 2
11	Huffman Table 3

9.1.1.42 Motion Vector Table (MVTAB) (2 or 3 bits)

The MVTAB 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 MVTAB is 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. For B interlace field pictures, MVTAB is a 3 bit syntax element that indicates which of eight interlace Huffman tables are used to decode the motion vector data. For P interlace field pictures in which NUMREF = 0, MVTAB is a 2 bit syntax element that indicates which of four progressive Huffman tables are used to decode the motion vector data. For P interlace field pictures in which NUMREF = 1, MVTAB is a 3 bit syntax element that indicates which of eight interlace Huffman tables are used to decode the motion vector data.. Refer to section 10.3.4 for a description of the motion vector decoding process.

Table 95: **MVTAB code-table**

FLC	Motion Vector Huffman Table
00	Huffman Table 0
10	Huffman Table 1
01	Huffman Table 2
11	Huffman Table 3
100	Huffman Table 4
101	Huffman Table 5
110	Huffman Table 6
111	Huffman Table 7

The motion vector Huffman tables are listed in section 11.10.

9.1.1.43 Coded Block Pattern Table (CBPTAB) (3 bits)

The CBPTAB syntax element is a 3 bit value pesent in interlace field P, B and interlace frame P, B pictures. This syntax element signals which of four Huffman tables is used to decode the CBPCY syntax element in intra-coded or inter-coded macroblocks.

Table 96: **CBPTAB code-table**

FLC	CBP Huffman Table
000	Huffman Table 0
001	Huffman Table 1
010	Huffman Table 2
011	Huffman Table 3
100	Huffman Table 4
101	Huffman Table 5
110	Huffman Table 6
111	Huffman Table 7

9.1.1.44 2MV Block Pattern Table (2MVBPTAB) (2 bits)

The 2MVBPTAB syntax element is a 2 bit value present only in interlace frame P and interlace frame B pictures. This syntax element signals which one of four Huffman tables is used to decode the 2MV block pattern (2MVBP) syntax element in 2 MV field macroblocks.

Table 97: **2MVBP code-table**

FLC	CBP Huffman Table
00	Huffman Table 0
10	Huffman Table 1
01	Huffman Table 2
11	Huffman Table 3

9.1.1.45 4MV Block Pattern Table (4MVBPTAB) (2 bits)

The 4MVBPTAB syntax element is a 2 bit value present only in interlace frame P, B and interlace field P, B pictures. For interlace field P and B pictures, it is only 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 and B pictures, it is present if 4MVSWITCH syntax element is set to 1. The 4MVBPTAB syntax element signals which of four Huffman tables is used to decode the 4MV block pattern (4MVBP) syntax element in 4MV macroblocks.

Table 98: **4MVBP code-table**

FLC	CBP Huffman Table

00	Huffman Table 0
10	Huffman Table 1
01	Huffman Table 2
11	Huffman Table 3

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

This syntax element is present only in interlace field P, B pictures and interlace frame P, B pictures. It is the same as decribed in section 7.1.1.31.

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

This syntax element is present only in interlace field P, B pictures and interlace frame P, B pictures. It is the same as decribed in section 7.1.1.32.

9.1.1.48 Pan scan window coordinates (TOPLEFTX, TOPLEFTY, BOTRIGHTX, BOTRIGHTY)(4 X 16 bits)

TOPLEFTX, TOPLEFTY, BOTRIGHTX, BOTRIGHTY are four coordinates that specify each pan-scan window. Each occupies 16 bits, and is sent only in advanced profile when PANSCANFLAG = 1.

9.1.2 Slice Layer

Slice-layer may be present in interlaced coding of pictures in Advanced Profile. Refer to Section 7.1.2 for a description of the Slice layer. For interlaced frame and field pictures, the syntax elements of the slice layer are identical to those of progressive pictures. For interlace field pictures, two points should be emphasized. If the PIC_HEADER_FLAG = 1 in the slice layer of an interlace field picture, the picture header information that is repeated consists of both the frame picture header, and the field picture header of that field. While coding SLICE_ADDR syntax element, the row address is not 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 as the row address of the last macroblock row in the first field incremented by one.

9.1.3 Macroblock Layer

Data for each macroblock consists of a macroblock header followed by the block layer. Figure 80 to Figure 85 show 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 that the macroblock layer syntax elements occur in are indicated in the square brackets.

9.1.3.1 Macroblock Mode (MBMODE)(Variable size)[P,B]

MBMODE is a variable-length syntax element present in interlace field P, B and interlace frame P, B macroblocks. It is described in section 10.3.4.4 for interlace field P, B pictures and section 10.7.2.3 for interlace frame P, B pictures.

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

This syntax element is present only in I pictures, only when CONDOVER has the binary value 11, and when the raw mode is chosen to encode the OVERFLAGS plane. In this case, one bit is sent in the macroblock header to indicate whether or not to perform overlap filtering to edge pixels within the block and neighboring blocks. See Section 4.1.1.7 for a description.

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

SKIPMBBIT is a 1-bit syntax element present in P and B interlace frame macroblocks if the frame level syntax element SKIPMB (see Section 7.1.1.20) indicates that the raw mode is used. If SKIPMBBIT = 1, then the macroblock is skipped.

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

CBPCY is a variable-length syntax element present in both I picture and P picture macroblock layers. Section 8.1.1.5 describes the CBPCY syntax element in I picture macroblocks and section 10.3.4.6 describes the CBPCY syntax element in P picture macroblocks.

9.1.3.5 Field Transform Flag (FIELDTX)(1 bit)[I, P,B]

FIELDTX is a 1-bit syntax present in all I interlace frame macroblocks, and in those P and B interlace macroblocks which are intra-coded. In an I-interlace frame, this syntax element may be bitplane coded at the picture level. This syntax element indicates whether a macroblock is frame or field coded (basically, the internal organization of the macroblock). FIELDTX = 1 indicates that the macroblock is field coded. Otherwise, the macroblock is frame coded. See section 10.5.1 for more details on the use of FIELDTX.

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

The ACPRED syntax element is present in all I interlace frame and field picture macroblocks and Intra macroblocks in field and frame P and B pictures. This is a 1-bit syntax element that specifies whether the blocks were coded using AC prediction. ACPRED = 0 indicates that AC prediction is not used. ACPRED = 1 indicates that AC prediction is used. See section 8.1.1.6 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 pictures.

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

MQDIFF is present in interlace field I, P, B pictures and interlace frame I, P, B pictures and it is the same as described in section 7.1.3.6.

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

ABSMQ is present in interlace field I, P, B pictures and interlace frame I, P, B pictures and it is the same as described in section 7.1.3.7.

9.1.3.9 Motion Vector Data (MVDATA)(Variable size)[P,B]

MVDATA is a variable sized syntax element present in interlace field P, B and interlace frame P, B picture macroblocks. This syntax element encodes the motion vector(s) for the macroblock. See section 10.3.4.5 for a description of the motion vector decode process.

9.1.3.10 Block-level Motion Vector Data (BLKMVDATA)(Variable size)[inter]

BLKMVDATA is a syntax element that contains motion information for the block. It is a variable sized syntax element and is only present in certain situations. See section 8.3.5.1 for a description of when the BLKMVDATA syntax element is present and how it is used.

9.1.3.11 Hybrid Motion Vector Prediction (HYBRIDPRED)(1 bit)[P,B]

HYBRIDPRED is a 1-bit syntax element per motion vector, present in interlace field P picture macroblocks. Section 10.3.4.5.3.5 describes how HYBRIDPRED is used in the decoding process.

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

TTMB is present in interlace field P, B pictures and interlace frame P, B pictures and it is the same as described in section 7.1.3.11.

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

DIRECTBBIT is a 1-bit syntax element present only in interlace frame B picture macroblocks if the frame level syntax element DIRECTMB (see section 7.1.1.21) indicates that the raw mode is used. If DIRECTBBIT = 1, then the macroblock is coded using direct mode.

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

BMV1 is a variable sized syntax element present in interlace frame and field B picture macroblocks. This syntax element encodes the first motion vector for the macroblock. See section 10.3.4.5.1 for a description of the motion vector decode process.

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

BMV2 is a variable sized syntax element present in interlace frame and field B picture macroblocks if the Interpolation mode is used. This syntax element encodes the second motion vector for the macroblock. See section 10.3.4.5.1 for a description of the motion vector decode process.

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

BMVTYPE is a variable sized syntax element present in interlace frame and field B picture macroblocks that indicates whether the macroblock uses forward, backward or interpolated prediction. As Table 46 shows, the value of BFRACTION (in the picture header, see section 7.1.1.10) along with BMVTYPE determine whether forward or backward prediction are indicated.

9.1.3.17 Forward B Field Coding Mode (FORWARDBIT)(1 bit)[B]

FORWARDBIT is a 1-bit syntax element 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 is coded using forward mode.

9.1.3.18 Interpolated MV Present (INTERPMVP)(1 bit)[B]

INTERPMVP is a 1-bit syntax element present in B field macroblocks if the field level syntax element BMVTYPE indicates that the macroblock type is interpolated. If INTERPMVP = 1, then the the interpolated MV, i.e. BMV2 is present, else it is not (i.e. zero).

9.1.3.19 B Frame MV Switch (MVSW)(1 bit)[B]

MVSW is a 1-bit syntax element present in B frame macroblocks if the MB is in field mode and if the BMVTYPE is forward or backward. If MVSW = 1, then the MV type and prediction type changes from forward to backward (or backward to forward) in going from the top to the bottom field.

9.1.4 Block Layer Syntax Elements

The block layer syntax elements in interlace pictures are identical to the corresponding syntax elements in progressive pictures which is described in 7.1.4.

10 Interlace Decoding Process

10.1 Interlace Field I Picture Decoding

The following sections describe the process for decoding field I pictures.

10.1.1 Macroblock Layer Decode

The macroblocks are coded in raster scan order form left to right. Figure 20 shows the elements that make up the I picture macroblock layer. Figure 3 shows how the frame is composed of macroblocks.

10.1.1.1 Coded Block Pattern

The coded block pattern is the same as advanced profile I pictures as described in section 8.1.1.5.

10.1.1.2 AC Prediction Flag

The ACPRED syntax element in the macroblock header is a one-bit syntax element that specifies whether AC prediction is used to decode the AC coefficients for all the blocks in the macroblock. Section 8.1.1.13 describes the AC prediction process. If ACPRED is 1, then AC prediction is used, otherwise it is not used.

10.1.2 Block Layer Decode

The 4 blocks that make up the Y component of the macroblock are coded first followed by the Cb and Cr blocks as shown in Figure 3. This section describes the process used to reconstruct the blocks.

Figure 4 shows the forward intra-coding steps used to encode the 8x8 pixel blocks. Figure 31 shows the inverse process used to reconstruct the 8x8 blocks.

As Figure 31 shows, the DC and AC Transform coefficients are coded using separate techniques. The DC coefficient is coded 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 in I pictures

10.1.2.1 DC Differential Bitstream Decode

The DC differential decoding process is the same as described in section 8.1.1.7.

10.1.2.2 DC Predictor

The DC Predictor is formed as described in section 8.1.1.8.

10.1.2.3 DC Inverse-quantization

The DC inverse-quantization process is described in section 8.1.1.9.

10.1.2.4 AC Coefficient Bitstream Decode

The AC Coefficient decoding process is described in section 8.1.1.10.

10.1.2.5 Zig-zag Scan of AC Coefficients

The zig-zag scanning of AC coefficient is the same as described in section 8.1.1.12.

10.1.2.6 AC Prediction

The AC predition process is the same as described in section 8.1.1.13.

10.1.2.7 Inverse AC Coefficient Quantization

The inverse AC coefficient quantization process is the same as described in section 8.1.1.14.

10.1.2.8 Coefficient Scaling

For DC and AC prediction, the coefficients in the predicted blocks are scaled if the macroblocks quantizers are different than that of the current block as described in section 8.1.1.15.

10.1.2.9 Inverse Transform

After reconstruction of the Transform coefficients, the resulting 8×8 blocks are processed by a separable two-dimensional inverse transform of size 8 by 8. The inverse transform output has a dynamic range of 10 bits. See section 8.8 regarding transform conformance.

Subsequent to the inverse transform, the process of overlap smoothing is carried out if signaled. This is covered in Section 10.9. Finally, the constant value of 128 is added to the reconstructed and possibly overlap smoothed intra block. This result is clamped to the range [0 255] and forms the reconstruction prior to loop filtering.

10.2 Interlace BI Field Decoding

When B frames are used (in main and advanced profiles only), we code a special type of frame that is in some ways a hybrid of I and B frames. The syntax of BI frames is identical to that of I, but they are usually coded at higher QP's and can never be used as an anchor or reference frame to predict other frames.

10.3 Interlace Field P Picture Decoding

Figure 51 shows the steps required to decode and reconstruct blocks in interlace field P pictures. The following sections describe the process for decoding interlace field P pictures.

10.3.1 Out-of-bounds Reference Pixels

The previously one or two decoded field(s) is used as the reference for motion-compensated predictive coding of the current field P picture. 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 field. In these cases, the out-of-bounds pixel values are the replicated values of the edge pixel. Figure 41 illustrates pixel replication for the upper-left corner of the frame. For motion vectors that reference out-of-field pixels, part or all of the reference block of pixels is made up of padded pixel values. The padding is conceptually considered to be infinite for the purpose of motion compensation. Note that in advanced profile, "frame edge", "frame corner" and "outside the boundary" refer to the true frame dimensions, not the dimensions right or top/bottom justified to the edge of the macroblock. In other words, the right and bottom pixels that are repeated to infinity for a 200 x 300 image begin at column 304 and row 208 for the simple and main profiles. However, for the advanced profile, these begin respectively at column 300 and row 200.

10.3.2 Reference Pictures

A P Field Picture may reference either one or two previously decoded fields. The NUMREF syntax element in the picture layer is a one bit syntax element that indicates whether the current field may reference one or two previous reference field pictures. If NUMREF = 0, then the current P field picture may only reference one field. In this case, the REFFIELD syntax element follows in the picture layer bitstream. The REFFIELD syntax element is a one bit syntax element that indicates which previously decoded field is used as a reference. If REFFIELD = 0, then the temporally closest (in display order) I or P field is used as a reference. If REFFIELD = 1, then the second most temporally recent I or P field picture is used as reference.

If NUMREF = 1, then the current P field picture uses the two temporally closest (in display order) I or P field pictures as reference.

Figure 89 and Figure 88 show examples of reference field pictures for NUMREF = 0 and NUMREF = 1.

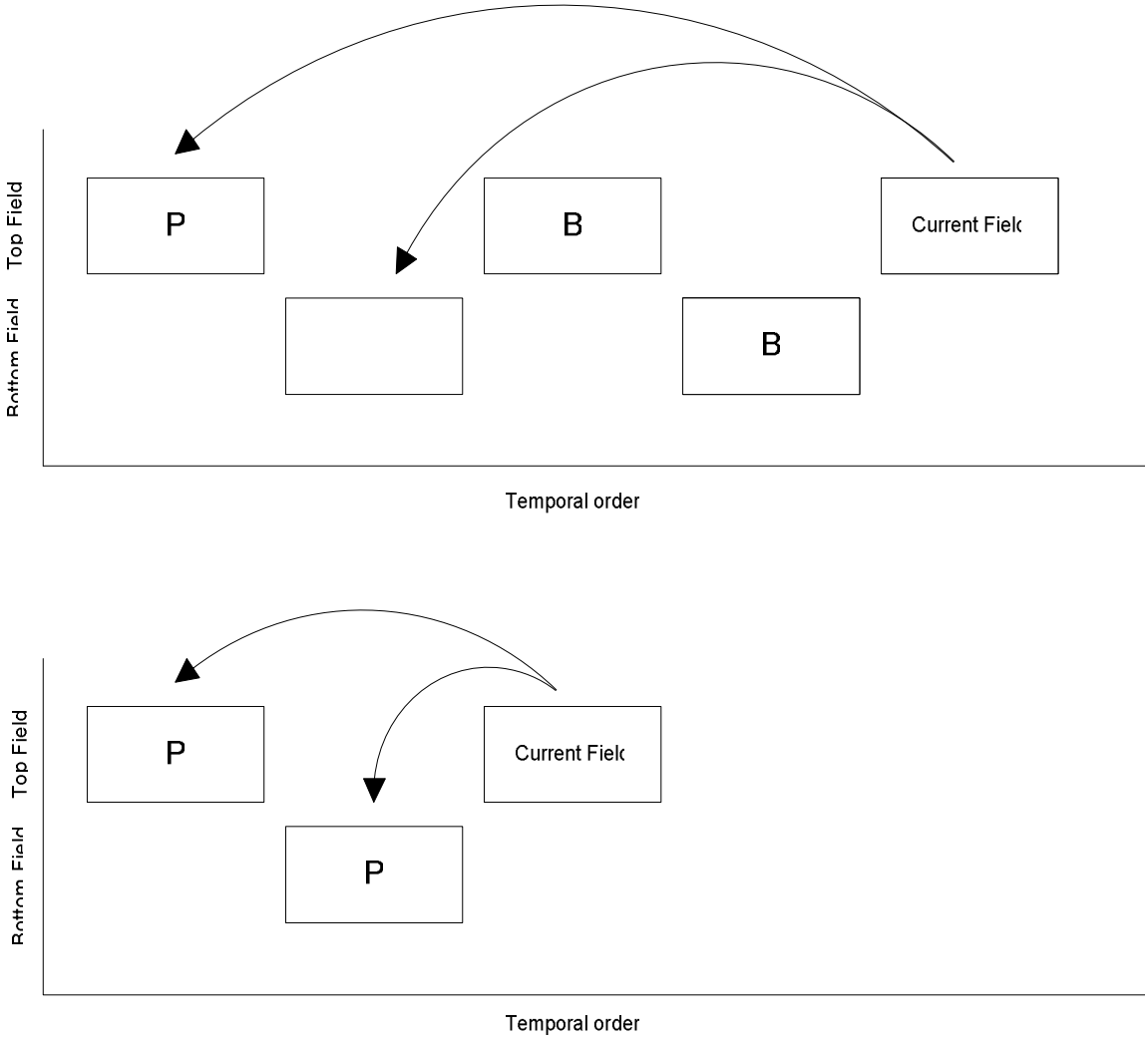


Figure 88: Example of two reference field pictures (NUMREF = 1)

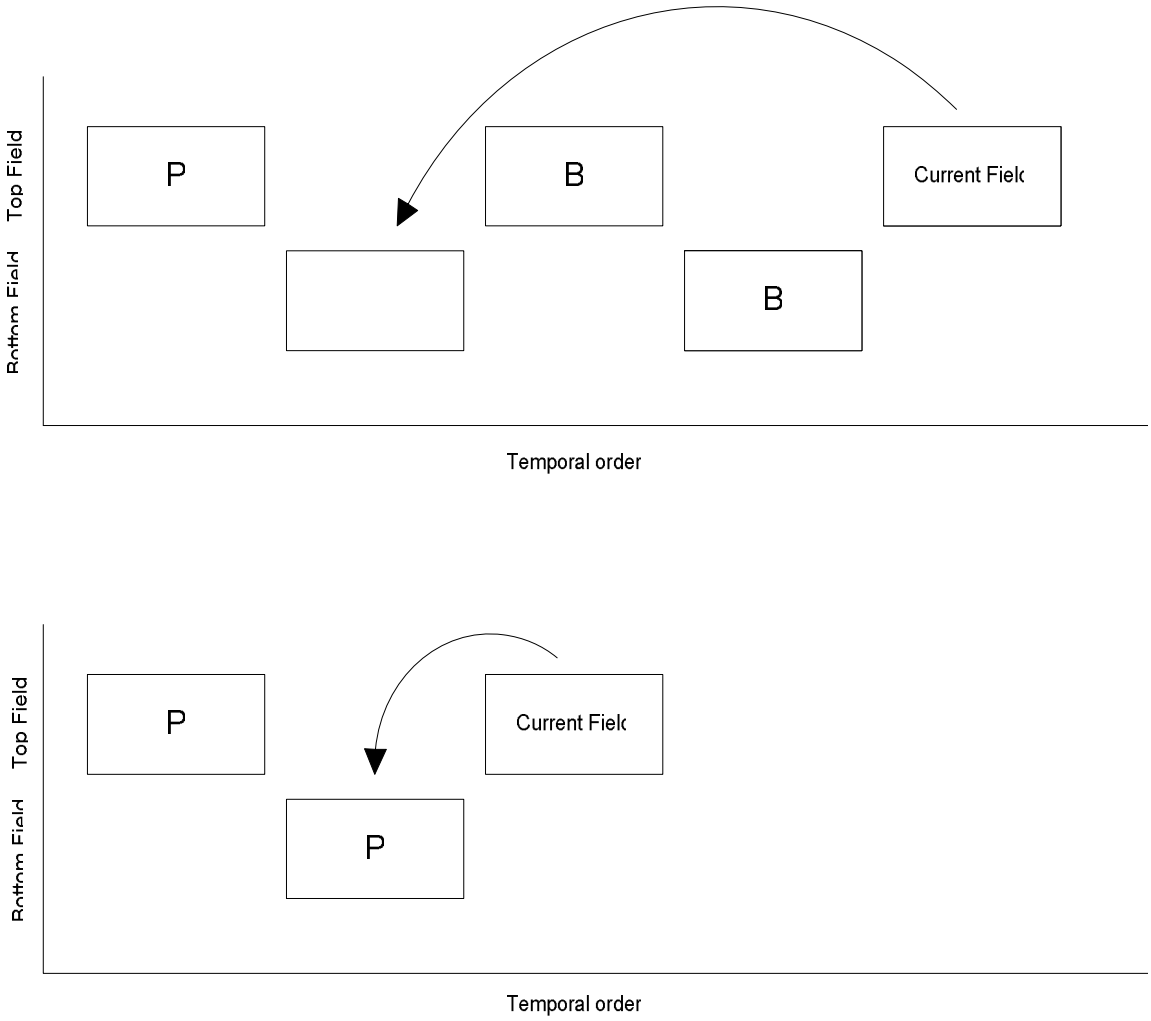


Figure 89: Example of one reference field picture (NUMREF = 0) using temporally most recent reference (REFFIELD = 0)

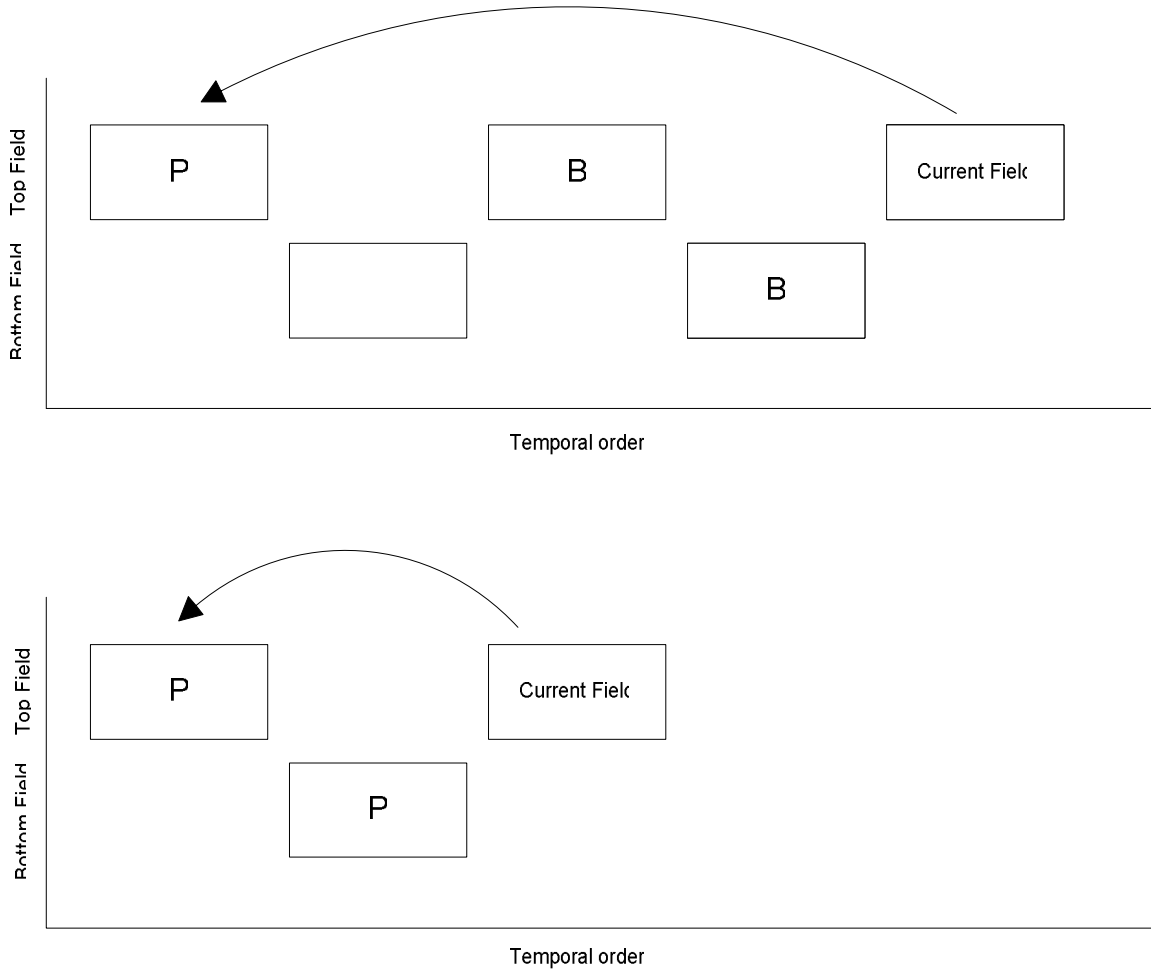


Figure 90: Example of one reference field picture (NUMREF = 0) using temporally second-most recent reference (REFFIELD = 1)

10.3.3 P Picture Types

P pictures may be one of two types: 1-MV or Mixed-MV. The following sections describe each P picture type.

10.3.3.1 1-MV P Picture

In 1-MV P pictures, a single motion vector is 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.

10.3.3.2 Mixed-MV P Picture

In Mixed-MV P pictures, each macroblock may be encoded as a 1-MV or a 4-MV macroblock. In 4-MV macroblocks, each of the 4 luminance blocks has a motion vector associated with it. The 1-MV or 4-MV mode for each macroblock is indicated by the MBMODE syntax element at every macroblock. The Mixed-MV mode is signaled by the MVMODE and MVMODE2 picture layer syntax elements as described in section 8.3.4.3.

10.3.4 Macroblock Layer Decode

Macroblocks in P pictures may be one of 3 possible types: 1MV, 4MV, 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.

10.3.4.1 1MV Macroblocks

1MV macroblocks may occur in 1-MV and Mixed-MV P pictures. A 1MV 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 1MV macroblocks the MBMODE syntax element in the macroblock layer indicates three things:

1) That the macroblock type is 1MV
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 is present in the macroblock layer in the corresponding position. The CBPCY 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 is assumed to equal 0 and no block data is 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 is present in the macroblock layer in the corresponding position. The MVDATA 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 MVDATA syntax element is not present, then the motion vector differential is assumed to be zero and therefore the motion vector is equal to the motion vector predictor.

10.3.4.2 4MV Macroblocks

4MV macroblocks may only occur in Mixed-MV P pictures. A 4MV macroblock is one where each of the 4 luminance 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 chroma blocks is derived from the 4 luminance motion vectors. This procedure is 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 1MV macroblocks the MBMODE syntax element in the macroblock layer indicates three things:

1) That the macroblock type is 4MV
2) Whether the CBPCY syntax element is present
3) Whether the 4MVBP 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 is assumed to equal 0 and no block data is present for any of the 6 blocks in the macroblock.

The 4MVBP syntax element indicates which of the 4 luminance 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 indicates whether the motion vector for the corresponding luminance block is present.

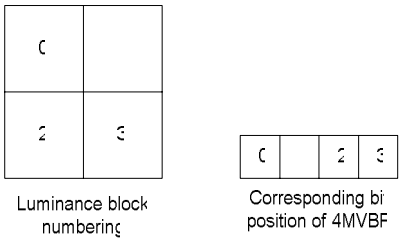


Figure 91: Association of bits in 4MVBP to luminance blocks

For each of the 4 bit positions in the 4MVBP, a value of 0 indicates that no motion vector differential (BLKMVDATA) is present for that block and the motion vector differential is assumed to be 0. A value of 1 indicates

that a motion vector differential (BLKMVDATA) is present for that block in the corresponding position. For example, if 4MVBP decodes to a value of 1100 (binary), then the bitstream contains MVDATA for blocks 0 and 1 and no MVDATA is present for blocks 2 and 3.

If the MBMODE syntax element indicates that the 4MVBP syntax element is not present, then it is assumed that motion vector differential data (BLKMVDATA) is present for all 4 luminance blocks.

10.3.4.3 Intra Macroblocks

Intra macroblocks may 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 difference between the current block pixel and a constant value of 128 is encoded in the block.

For Intra macroblocks, the MBMODE syntax element in the macroblock layer indicates two things:

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 is 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 is assumed to equal 0 and no AC coefficient data is present for any of the 6 blocks in the macroblock.

10.3.4.4 Macroblock Mode

The MBMODE syntax element indicates the macroblock type (1MV, 4MV or Intra) and also the presence of the CBP flag and MV data, as described above. Depending on whether the MVMODE/MVMODE2 syntax element indicates mixed-MV or all-1MV the MBMODE signals the information as follows:

10.3.4.4.1 Macroblock Mode in All-1MV Pictures

Table 99 shows how the MBMODE signals information about the macroblock in all-1MV pictures.

Table 99: Macroblock Mode in All-1MV Pictures

Index	Macroblock Type	CBP Present	MV Present
0	Intra	No	NA
1	Intra	Yes	NA
2	1MV	No	No
3	1MV	No	Yes
4	1MV	Yes	No
5	1MV	Yes	Yes

10.3.4.4.2 Macroblock Mode in Mixed-1MV Pictures

Table 100 shows how the MBMODE signals information about the macroblock in mixed-MV pictures.

Table 100: Macroblock Mode in Mixed-1MV Pictures

Index	Macroblock Type	CBP Present	MV Present
0	Intra	No	NA

1	Intra	Yes	NA
2	1MV	No	No
3	1MV	No	Yes
4	1MV	Yes	No
5	1MV	Yes	Yes
6	4MV	No	NA
7	4MV	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. The Huffman Mixed-MV mode tables are shown in Table 134 through Table 141. The 1-MV mode tables are shown in Table 142 through Table 149.

10.3.4.5 Motion Vector Decoding Process

The following sections describe the motion vector decoding process for P field picture macroblocks.

10.3.4.5.1 Field Picture Coordinate System

In the following sections which describe the motion vector decoding process the motion vector units are expressed in field picture units. For example, if the vertical component a motion vector indicates that the displacement is +6 (in quarter pel units) then this indicates a displacement of 1 ½ field picture lines.

Figure 92 shows the relationship between the vertical component of the motion vector and the spatial location for all combinations of current and reference field polarities. The figure shows one vertical column of pixels in the current and reference fields. Each circles represents integer pixel positions and the x's represent quarter pixel positions. The figure shows that if the current field is a top field and the reference field is a bottom field, a vertical motion vector component value of 0 represents a position in the reference field that is a half pixel offset below the location in the current field. Similarly, if the current field is a bottom field and the reference field is a top field then a vertical motion vector component of 0 represents a position in the reference field that is a half pixel offset above the location in the current field.

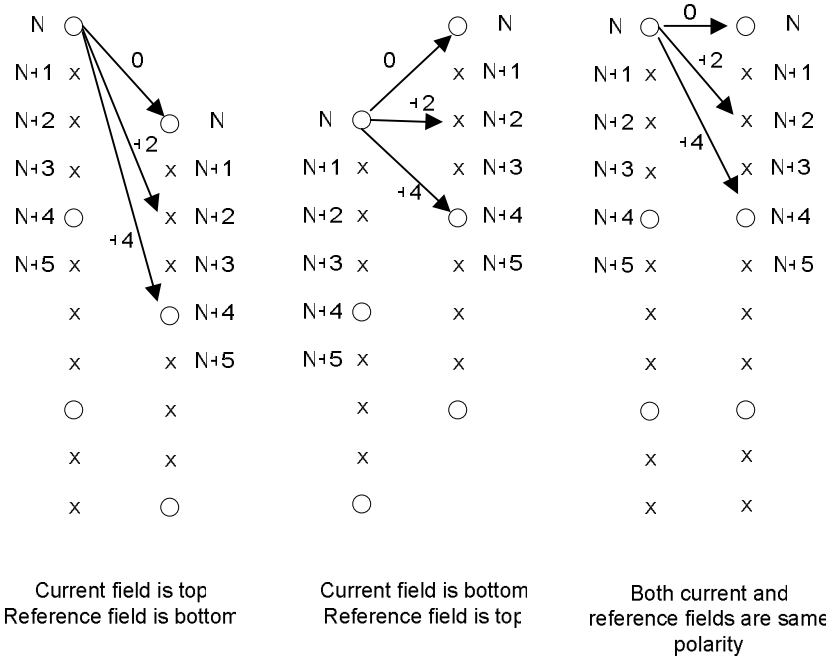


Figure 92: Vertical relationship between motion vectors and current and reference fields

10.3.4.5.2 Decoding Motion Vector Differential

The MVDATA or BLKMVDATA syntax elements encode motion information for the blocks in the macroblock. 1MV macroblocks have a single MVDATA syntax element, and 4MV macroblocks may have between zero and four BLKMVDATA. 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.

10.3.4.5.2.1 Motion Vector Differentials in One-Reference Field Pictures

In field pictures that have only one reference field, each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly encodes two things: 1) the horizontal motion vector differential component and 2) the vertical motion vector differential component.

The MVDATA or BLKMVDATA syntax element is a variable length Huffman codeword followed by a fixed length codeword. The value of the Huffman codeword determines the size of the fixed length codeword. The MVTAB syntax element in the picture layer specifies the Huffman table used to decode the variable sized codeword.

The following pseudocode illustrates how the motion vector differential is decoded.

The values **dmv_x** and **dmv_y** are computed in the following pseudocode. 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 7.1.1.16) according to Table 66.

Table 101: 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

extend_x: extened range for horizontal motion vector differential

extend_y: extended range for vertical motion vector differential

extend_x and **extend_y** are derived from the DMVRANGE picture field 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.

The value **halfpel_flag** used in the following pseudocode 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 (or MVMODE2, when MVMODE indicates intensity compensation) specifies the mode as 1MV or Mixed MV, then **halfpel_flag** = 0 and quarter-pel precision is used. If MVMODE (or MVMODE2, when MVMODE indicates intensity compensation) specifies the mode as 1MV Half-pel or 1MV Half-pel Bilinear, then **halfpel_flag** = 1 and half-pel precision is used.

The **offset_table** is an array used in the following pseudocode and is 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}

```
if (extend_x == 1 || extend_y == 1)
      offset_table = offset_table2
else
      offset_table = offset_table1
index = vlc_decode()     // Use the Huffman table indicated by MVTAB in the picture layer
if (index == 0) {
      val = get_bits (1 + extend_x)
       sign = 0 - (val & 1)
       dmv_x = sign ^ ((val >> 1) + 1)
       dmv_x = dmv_x – sign
      dmv_y = 0
}
if (index == 71)
{
      dmv_x = get_bits(k_x – halfpel_flag)
      dmv_y = get_bits(k_y – halfpel_flag)
}
else
{
          index1 = (index + 1) % 9
          val = get_bits (index1 + extend_x)
          sign = 0 - (val & 1)
          dmv_x = sign ^ ((val >> 1) + offset_table[index1])
          dmv_x = dmv_x - sign

          index1 = (index + 1) / 9
          val = get_bits (index1 + extend_y)
          sign = 0 - (val & 1)
          dmv_y = sign ^ ((val >> 1) + offset_table[index1])
          dmv_y = dmv_y – sign
}
```

10.3.4.5.2.2 Motion Vector Differentials in Two-Reference Field Pictures

Two-reference 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 field pictures that have two reference fields, each MVDATA or BLKMVDATA syntax element in the macroblock layer jointly encodes three things: 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 Huffman codeword followed by a fixed length codeword. The value of the Huffman codeword determines the size of the fixed length codeword. The MVTAB syntax element in the picture layer specifies the Huffman table used to decode the variable sized codeword.

The following pseudocode illustrates how the motion vector differential, and dominant/non-dominant predictor information are decoded.

The values **predictor_flag, dmv_x** and **dmv_y** are computed in the following pseudocode. 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: extened 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 7.1.1.18) according to Table 66.

extend_x and **extend_y** are derived from the DMVRANGE picture field 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.

The value **halfpel_flag** used in the following pseudocode 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 (or MVMODE2, if MVMODE indicates intensity compensation) specifies the mode as 1MV or Mixed MV, then **halfpel_flag** = 0 and quarter-pel precision is used. If MVMODE (or MVMODE2, if MVMODE indicates intensity compensation) specifies the mode as 1MV Half-pel or 1MV Half-pel Bilinear, then **halfpel_flag** = 1 and half-pel precision is used.

The tables **size_table** and **offset_table** are arrays used in the following pseudocode and are 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}

```
if (extend_x == 1 || extend_y == 1)
    offset_table = offset_table2
else
    offset_table = offset_table1
index = vlc_decode()    // Use the Huffman table indicated by MVTAB in the picture layer
if (index == 0) {
    val = get_bits (1 + extend_x)
    sign = 0 - (val & 1)
    dmv_x = sign ^ ((val >> 1) + 1)
    dmv_x = dmv_x – sign
    dmv_y = 0
    predictor_flag = 0
}
else if (index == 125)
{
    dmv_x = get_bits(k_x – halfpel_flag)
    dmv_y = get_bits(k_y – halfpel_flag)
    predictor_flag = dmv_y & 1
    dmv_y = dmv_y >> 1
}
else
{
```

```
        index1 = (index + 1) % 9
        val = get_bits (index1 + extend_x)
        sign = 0 - (val & 1)
        dmv_x = sign ^ ((val >> 1) + offset_table[index1])
        dmv_x = dmv_x - sign

        index1 = (index + 1) / 9
        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
        predictor_flag = index1 & 1
}
```

10.3.4.5.3 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 1MV P pictures and Mixed-MV P pictures.

10.3.4.5.3.1 Motion Vector Predictors In 1MV P Pictures

Figure 43 shows the three motion vectors used to compute the predictor for the current macroblock. As the figure shows, the predictor is 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 is taken from the top-left macroblock instead of the top-right.

For the special case where the frame is one macroblock wide, the predictor is always Predictor A (the top predictor).

10.3.4.5.3.2 Motion Vector Predictors In Mixed-MV P Pictures

Figure 44 and Figure 45 show the 3 candidate motion vectors for 1MV and 4MV 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 is always Predictor A (the top predictor).

Figure 44 shows the candidate motion vectors for 1MV macroblocks. The neighboring macroblocks may be 1MV or 4 MV macroblocks. The figure shows the candidate motion vectors assuming the neighbors are 4MV (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 1MV macroblocks, then the motion vector predictors shown in Figure 44 are 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 is from block 3 of the top-left macroblock instead of from block 2 in the top-right macroblock as is the case otherwise.

Figure 45 shows the predictors for each of the 4 luminance blocks in a 4MV 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 is 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 is 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 are set equal to 0.

10.3.4.5.3.3 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 is considered to 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 is considered to be the dominant predictor. Intra-coded macroblocks are not 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 are set to zero and the dominant predictor is taken to be from the opposite field.

10.3.4.5.3.4 Calculating the Motion Vector Predictor

If the NUMREF syntax element in the picture header = 0, then the current field picture may refer to only one previously coded picture. If NUMREF = 1, then the current field picture may refer to the two most recent field pictures. In the former case, a single predictor is calculated for each motion vector. In the latter case, two motion vector predictors are calculated. The following pseudocode describes how the motion vector predictors are calculated for each case. The variables *fieldpred_x* and *fieldpred_y* in the pseudocode represent the horizontal and vertical components of the motion vector predictor.

10.3.4.5.3.4.1 Motion Vector Predictors in One-Reference Field Pictures

```
if (predictorA is not out of bounds) {
    if (predictorC is not out of bounds) {
        if (predictorA is intra) {
            predictorA_x = 0
            predictorA_y = 0
        }
        if (predictorB is intra) {
            predictorB_x = 0
            predictorB_y = 0
        }
        if (predictorC is intra) {
            predictorC_x = 0
            predictorC_y = 0
        }
        fieldpred_x =
            median (predictorA_x, predictorB_x, predictorC_x)
        fieldpred_y =
            median (predictorA_y, 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 and PredictorB
                predictorC_x = 0
                predictorC_y = 0
                if (predictorA is intra) {
                    predictorA_x = 0
                    predictorA_y = 0
                }
                if (predictorB is intra) {
                    predictorB_x = 0
                    predictorB_y = 0
                }
                if (predictorC is intra) {
                    predictorC_x = 0
                    predictorC_y = 0
                }
                fieldpred_x =
                    median (predictorA_x, predictorB_x, predictorC_x)
                fieldpred_y =
                    median (predictorA_y, predictorB_y, predictorC_y)
            }
        }
    }
    else {
        // Predictor A is out of bounds
        if (predictorC is out of bounds) {
            fieldpred_x = 0
            fieldpred_y = 0
        }
        else {
            // Use predictorC
            fieldpred_x = predictorC_x
            fieldpred_y = predictorC_y
        }
    }
```

10.3.4.5.3.4.2 Motion Vector Predictors in Two-Reference Field Pictures

In 2-reference pictures (NUMREF = 1) the current field may reference the two most recent fields. In this case two motion vector predictors are computed for each macroblock. One predictor is from the reference field of the same polarity and the other is from the reference field with the opposite polarity.

Given the 3 motion vector predictor candidates, the following pseudocode illustrates the process for calculating the motion vector predictors. The variables *samefieldpred_x* and *samefieldpred_y* in the pseudocode 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.

```
samecount = 0;
oppositecount = 0;
if (predictorA is not out of bounds) {
    if (predictorC is not out of bounds) {
        if (predictorA is intra) {
            predictorA_x = 0
            predictorA_y = 0
        }
        if (predictorB is intra) {
            predictorB_x = 0
            predictorB_y = 0
        }
        if (predictorC is intra) {
            predictorC_x = 0
            predictorC_y = 0
        }
        if (predictorA is from same field) {
            samecount = samecount + 1
            samefieldpredA_x = predictorA_x
            samefieldpredA_y = predictorA_y
            oppositefieldpredA_x = scaleforopposite_x(predictorA_x)
            oppositefieldpredA_y = scaleforopposite_y(predictorA_y)
        }
        else {
            oppositecount = oppositecount + 1
            oppositefieldpredA_x = predictorA_x
            oppositefieldpredA_y = predictorA_y
            samefieldpredA_x = scaleforsame_x(predictorA_x)
            samefieldpredA_y = scaleforsame_y(predictorA_y)
        }
        if (predictorB is from same field) {
            samecount = samecount + 1
            samefieldpredB_x = predictorB_x
            samefieldpredB_y = predictorB_y
            oppositefieldpredB_x = scaleforopposite_x(predictorB_x)
            oppositefieldpredB_y = scaleforopposite_y(predictorB_y)
        }
        else {
            oppositecount   = oppositecount + 1
            oppositefieldpredB_x = predictorB_x
            oppositefieldpredB _y = predictorB_y
            samefieldpredB_x = scaleforsame_x(predictorB_x)
            samefieldpredB_y = scaleforsame_y(predictorB_y)
        }
        if (predictorC is from same field) {
            samecount = samecount + 1
```

```
            samefieldpredC_x = predictorC_x
            samefieldpredC_y = predictorC_y
            oppositefieldpredC_x = scaleforopposite_x(predictorC_x)
            oppositefieldpredC_y = scaleforopposite_y(predictorC_y)
        }
        else {
            oppositecount = oppositecount + 1
            oppositefieldpredC_x = predictorC_x
            oppositefieldpredC _y = predictorC_y
            samefieldpredC _x = scaleforsame_x(predictorC_x)
            samefieldpredC _y = scaleforsame_y(predictorC_y)
        }
        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 = oppsositefield
    }
    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) {
            predictorA_x = 0
            predictorA_y = 0
        }
        if (predictorB is intra) {
            predictorB_x = 0
            predictorB_y = 0
        }
        if (predictorC is intra) {
            predictorC_x = 0
            predictorC_y = 0
        }
        if (predictorA is from same field) {
            samecount = samecount + 1
            samefieldpredA_x = predictorA_x
            samefieldpredA_y = predictorA_y
            oppositefieldpredA_x = scaleforopposite_x(predictorA_x)
            oppositefieldpredA_y = scaleforopposite_y(predictorA_y)
        }
        else {
            oppositecount = oppositecount + 1
            oppositefieldpredA_x = predictorA_x
            oppositefieldpredA _y = predictorA_y
            samefieldpredA_x = scaleforsame_x(predictorA_x)
            samefieldpredA_y = scaleforsame_y(predictorA_y)
        }
        if (predictorB is from same field) {
            samecount   = samecount + 1
            samefieldpredB_x = predictorB_x
            samefieldpredB_y = predictorB_y
            oppositefieldpredB_x = scaleforopposite_x(predictorB_x)
            oppositefieldpredB_y = scaleforopposite_y(predictorB_y)
        }
        else {
            oppositecount = oppositecount + 1
            oppositefieldpredB_x = predictorB_x
```

```
                oppositefieldpredB_y = predictorB_y
                samefieldpredB_x = scaleforsame_x(predictorB_x)
                samefieldpredB_y = scaleforsame_y(predictorB_y)
            }
            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 = oppsositefield
        }
    }
}
else {
    // Predictor A is out of bounds
    if (predictorC is out of bounds) {
        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
        }
    }
}
```

The scaling operation used to derive the other field's predictor is defined as follows:

```
scaleforopposite_x (n)  {
    int scaledvalue
    scaledvalue = (n * SCALEOPP) >> 8
    return scaledvalue
}
scaleforopposite_y (n)  {
    int scaledvalue
    if (current field is top)
        scaledvalue = ((n * SCALEOPP) >> 8) – 2
    else   //current field is bottom
        scaledvalue = ((n * SCALEOPP) >> 8) + 2
    return scaledvalue
}
scaleforsame_x (n)  {
    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
    }
    return scaledvalue
}
scaleforsame_y (n)  {
    if (current field is top) {
        if (abs (n) < SCALEZONE1_Y)
            scaledvalue = ((n + 2) * SCALESAME1) >> 8
        else {
            if (n < 0)
                scaledvalue = (((n + 2) * SCALESAME2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = (((n + 2) * SCALESAME2) >> 8) + ZONE1OFFSET_Y
        }
    }
    else {      //current field is bottom
        if (abs (n) < SCALEZONE1_Y)
            scaledvalue = ((n - 2) * SCALESAME1) >> 8
        else {
            if (n < 0)
                scaledvalue = (((n - 2) * SCALESAME2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = (((n - 2) * SCALESAME2) >> 8) + ZONE1OFFSET_Y
        }
    }
```

```
    return scaledvalue
}
```

The values of SCALEOPP, SCALESAME1, SCALESAME2, SCALEZONE1_X, SCALEZONE1_Y, ZONE1OFFSET_X and ZONE1OFFSET_Y are shown in Table 102 for the case where the current field is the first field and Table 103 for the case where the current field is the second field. The reference frame distance is encoded in the REFDIST field in the picture header. The reference frame distance is REFDIST + 1.

The value of N is dependant on the motion vector range. Extended motion vector range is signaled at the sequence level by the sequence header syntax element EXTENDED_MV= 1. If EXTENDED_MV = 1 then the MVRANGE syntax element is present in the picture header and signals the motion vector range. If EXTENDED_MV = 0 then the default motion vector range is used. Table 104 shows the relationship between N and the MVRANGE.

Table 102: P Field Picture MV Predictor Scaling Values when Current Field is First

	Reference Frame Distance			
	1	2	3	4 or greater
SCALEOPP	128	192	213	224
SCALESAME1	512	341	307	293
SCALESAME2	219	236	242	245
SCALEZONE1_X	32 * N	48 * N	53 * N	56 * N
SCALEZONE1_Y	8 * N	12 * N	13 * N	14 * N
ZONE1OFFSET_X	37 * N	20 * N	14 * N	11 * N
ZONE1OFFSET_Y	10 * N	5 * N	4 * N	3 * N

Table 103: P Field Picture MV Predictor Scaling Values when Current Field is Second

	Reference Frame Distance			
	1	2	3	4 or greater
SCALEOPP	128	64	43	32
SCALESAME1	512	1024	1536	2048
SCALESAME2	219	204	200	198
SCALEZONE1_X	32 * N	16 * N	11 * N	8 * N
SCALEZONE1_Y	8 * N	4 * N	3 * N	2 * N
ZONE1OFFSET_X	37 * N	52 * N	56 * N	11 * N
ZONE1OFFSET_Y	10 * N	5 * N	4 * N	3 * N

Table 104: Derivation of N

MVRANGE	N
0 or default	1
10	2
110	8

10.3.4.5.3.5 Hybrid Motion Vector Prediction

If the P picture is 1MV or Mixed-MV, then 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 so, then a bit is present that indicates whether to use predictor A or predictor C as the motion vector predictor. The following pseudocode illustrates hybrid motion vector prediction decoding.

The variables are defined as follows in the pseudocode:

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 hybdrid motion vector prediction
predictor_post_y: The vertical motion vector predictor after checking for hybdrid 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) == 0) { // 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 (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) == 0) {
                // 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
            }
        }
    }
}
```

Note that in the above pseudocode predictor_pre, predictor_post, predictorA, predictorB and predictorC all represent fields of the polarity indicated by the value of predictor_flag as described in section 10.3.4.5.2.2. For example, if the predictor_flag indicates that the opposite field predictor is used then:

predictor_pre_x = oppositefieldpred_x

predictor_pre_x = 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 predictor_flag indicates that the same field predictor is used then:

predictor_pre_x = samefieldpred_x

predictor_pre_x = 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 10.3.4.5.3.4.2.

10.3.4.5.4 Reconstructing Motion Vectors

The following sections describe how to reconstruct the luminance and chroma motion vectors for 1MV and 4MV macroblocks.

10.3.4.5.4.1 Luminance Motion Vector Reconstruction

In all cases (1MV and 4MV macroblocks) the luminance motion vector is reconstructed by adding the differential to the predictor as follows:

$$mv\_x = (dmv\_x + predictor\_x) \text{ smod } range\_x$$

$$mv\_y = (dmv\_y + predictor\_y) \text{ smod } range\_y$$

The modulus operation "smod" is a signed modulus, defined as follows:

$$A \text{ smod } b = ((A + b) \% (2*b)) - b$$

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 66.

If the picture uses two reference pictures (NUMREF = 1), then the *predictor_flag* derived after decoding the motion vector differential is combined with the value of *dominantpredictor* derived from motion vector prediction to determine which field is used as reference. The following pseudocode describes 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
}
```

1MV Macroblock Notes

In 1MV macroblocks there will be a single motion vector for the 4 blocks that make up the luminance component of the macroblock.

If the MBMODE syntax element indicates that no MV data is present in the macroblock layer, then dmv_x = 0 and dmv_y = 0 (mv_x = predictor_x and mv_y = predictor_y).

4MV Macroblock Notes

Each of the Inter-coded luminance blocks in a macroblock will have its own motion vector. Therefore there will be between 0 and 4 luminance motion vectors in each 4MV macroblock.

If the 4MVBP syntax element indicates that no motion vector information is present for a block, then dmv_x = 0 and dmv_y for that block (mv_x = predictor_x and mv_y = predictor_y).

10.3.4.5.4.2 Chroma Motion Vector Reconstruction

The chroma motion vectors are derived from the luminance motion vectors. Also, for 4MV macroblocks, the decision on whether to code the chroma blocks as Inter or Intra is made based on the status of the luminance blocks or fields. The following sections describe how to reconstruct the chroma motion vectors for 1MV and 4MV macroblocks. The chroma vectors are reconstructed in two steps.

As a first step, the nominal chroma motion vector is obtained by combining and scaling the luminance 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, a sequence level 1-bit FASTUVMC syntax element is used to determine if further rounding of chroma motion vectors is necessary. The purpose of this mode is speed optimization of the decoder. If FASTUVMC = 0, no rounding is performed in the second stage. If FASTUVMC = 1, the chroma motion vectors that are at quarter pel offsets will be rounded to the nearest half and full pel positions as described in Section 8.3.5.4.5.

Only bilinear filtering will be used for all chroma interpolation

The motivation for this rounding is the significant difference between the complexities of interpolating pixel offsets that are at a) integer pel; b) half pel; c) at least one coordinate (of x and y) at a quarter pel; and d) both coordinates at quarter pel positions. The ratio of a:b:c:d is roughly 1:4:4.7:6.6. By applying this mode we may favor a) and b), thus cutting down on decoding time. Since this is being done only for chroma interpolation, the coding and quality loss (especially visible quality) are both negligible.

In the sections below cmv_x and cmv_y denote the chroma motion vector components and lmv_x and lmv_y denote the luminance motion vector components.

10.3.4.5.4.2.1 First-stage Chroma Motion Vector Reconstruction - 1MV Chroma Motion Vector Case:

In a 1MV macroblock, the chroma motion vectors are derived from the luminance 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

10.3.4.5.4.2.2 First-stage Chroma Motion Vector Reconstruction - 4MV Chroma Motion Vector Case:

The following pseudocode illustrates how the chroma motion vectors are derived from the motion information in the 4 luminance blocks in 4MV macroblocks. In this section, ix and iy are temporary variables.

Chroma Motion Vector Derivation in 1-Reference P Pictures

```
// 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

Chroma Motion Vector Derivation in 2-Reference P Pictures

```
if (all 4 luminance block motion vectors are from same field)
{
    // 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 luminance block motion vectors are from same field)
{
    // 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)
}
else if (2 of the luminance block motion vectors are from same field)
{
    // Use the 2 motion vectors from the field which 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
}

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

See section 4.9 for the definition of median3 and median4.

10.3.4.5.4.2.3 Second Stage Chroma Rounding

This follows the operations described in Section 8.3.5.4.5.

10.3.4.6 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 59 shows 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 indicates that the corresponding block does not contain any non-zero AC coefficients. A value of 1 indicates that at least one non-zero AC coefficient is present. The DC coefficient is still present for each block in all cases. For inter-coded macroblocks, a value of 0 in a particular bit position indicates that the corresponding block does not contain any non-zero coefficients. A value of 1 indicates that at least one non-zero coefficient is present. For cases where the bit is 0, no data is encoded for that block.

10.3.5 Block Layer Decode

10.3.5.1 Intra Coded Block Decode

The decoding process for Intra coded blocks is the same as described in section 8.3.6.1. The only exception is that unlike progressive P frames, field picture P frames do not contain Intra blocks within 4MV-coded macroblocks, so the section describing the decoding process for those blocks may be ignored.

10.3.5.2 Inter Coded Block Decode

Figure 51 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 may also be transformed with two 4x8 Transforms, four 4x4 Transforms, or one 8x8 Transform. The steps required to reconstruct an inter-coded block include: 1) transform type selection, 2) sub-block pattern decode, 3) coefficient decode, 4) inverse Transform, 5) obtain predicted block and 6) motion compensation (add predicted and error blocks). The following sections describe these steps.

10.3.5.2.1 Transform Type Selection

If variable-sized transform coding is enabled (signaled by the sequence-level syntax element VSTRANSFORM = 1 as described in section 6.1.14), then the 8x8 error block may be transformed using one 8x8 Transform, or as shown in Figure 52, 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 4 4x4 Transforms. The transform type is signaled at the picture, macroblock or block level. As shown in Tables Table 43,

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
11	8x8	Block	NA
101110	8x4	Block	Bottom
1011111	8x4	Block	Top
00	8x4	Block	Both
10110	4x8	Block	Right
10101	4x8	Block	Left
01	4x8	Block	Both
100	4x4	Block	NA
10100	8x8	Macroblock	NA
1011110001	8x4	Macroblock	Bottom
101111001	8x4	Macroblock	Top
101111011	8x4	Macroblock	Both
101111000000	4x8	Macroblock	Right
101111000001	4x8	Macroblock	Left
10111100001	4x8	Macroblock	Both
101111010	4x4	Macroblock	NA

Table 44: Medium Rate (5 <= PQUANT < 13) TTMB VLC Table

and

TTMB VLC	Transform Type	Signal Level	Subblock Pattern
110	8x8	Block	NA
0110	8x4	Block	Bottom
0011	8x4	Block	Top
0111	8x4	Block	Both
1111	4x8	Block	Right
1110	4x8	Block	Left
000	4x8	Block	Both

010	4x4	Block	NA
10	8x8	Macroblock	NA
0010100	8x4	Macroblock	Bottom
0010001	8x4	Macroblock	Top
001011	8x4	Macroblock	Both
001001	4x8	Macroblock	Right
00100001	4x8	Macroblock	Left
0010101	4x8	Macroblock	Both
00100000	4x4	Macroblock	NA

Table 45. If TTMB indicates that the signal level is Block, then the transform type is signaled at the block level. If the transform type is specified at the block level, then the TTBLK syntax element is present within the bitstream as shown in Figure 25. This syntax element indicates the transform type used for the block. Table 47, Table 48, and Table 49 show the code tables used to encode the transform types if block mode signaling is used.

If variable-sized transform coding is not enabled, then the 8x8 Transform is used for all blocks.

10.3.5.2.2 Subblock Pattern Decode

If the transform type is 8x4, 4x8 or 4x4, then information about which of the subblocks have non-zero coefficients shall be signaled to the decoder. For 8x4 and 4x8 transform types, the subblock pattern is decoded as part of the TTMB or TTBLK syntax element. If the transform type is 4x4, then the SUBBLKPAT syntax element is present in the bitstream as shown in Figure 25. Section 7.1.4.2 describes the SUBBLKPAT syntax element.

If the subblock pattern indicates that no non-zero coefficients are present for the subblock, then no other information for that subblock is present in the bitstream. For the 8x4 transform type, the data for the top subblock (if present) is coded first followed by the bottom subblock. For the 4x8 transform type, the data for the left subblock (if present) is coded first followed by the right subblock. For the 4x4 transform type, the data for the upper left subblock is coded first followed, in order, by the upper right, lower left and lower right subblocks.

10.3.5.2.3 Coefficient Bitstream Decode

The process of transform coefficient decoding is described in Section 8.3.6.2.3.

10.3.5.2.4 Inverse Quantization

The non-zero quantized coefficients reconstructed as described in the sections above are inverse quantized in one of two ways depending on the value of PQUANT.

If the uniform quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep)$$

If the nonuniform quantizer is used, the following formula describes the inverse quantization process:

$$dequant\_coeff = quant\_coeff * (2 * quant\_scale + halfstep) + \text{sign}(quant\_coeff) * quant\_scale$$

where:

quant_coeff is the quantized coefficient

dequant_coeff is the inverse quantized coefficient

quant_scale = The quantizer scale for the block (either PQUANT or MQUANT)

halfstep = The half step encoded in the picture layer as described in section 7.1.1.16.

PQUANT is encoded in the picture layer as described in section 7.1.1.15.

MQUANT is encoded as described in section 8.3.5.2.

10.3.5.2.5 Inverse Transform

After reconstruction of the TRANSFORM coefficients, the resulting 8x8, 8x4, 4x8 or 4x4 blocks are processed by the appropriate two-dimensional inverse transforms (INVERSETRANSFORM). The 8x8 blocks are transformed using the 8x8 INVERSETRANSFORM, the 8x4 blocks are transformed using the 8x4 INVERSETRANSFORM, the 4x8 blocks are transformed using the 4x8 INVERSETRANSFORM and the 4x4 blocks are transformed using the 4x4 INVERSETRANSFORM. The inverse transforms output has a dynamic range of 10 bits. For P blocks, the inverse transform may be clamped to the range [-255 255] without loss of bit-exactness.

See section 8.8 regarding INVERSETRANSFORM implementation and conformance.

10.3.5.2.6 Motion Compensation

The motion compensation process is the same as described in section 8.3.6.2.

10.3.6 Rounding Control

The rounding control process is the same as described in section 8.3.7.

10.3.7 Intensity Compensation

The intensity compensation process is the same as described in section 8.3.8.

10.4 Interlace Field B Picture Decoding

Figure 75 shows the steps required to decode B field pictures. The following sections describe this process. B field syntax is very similar to P field syntax, so we will not repeat all the common syntax elements here. We will instead focus only on the differences between P and B syntax. When using B frames, both forward and backward frames are needed for motion compensation. In field mode coding, the first (B-) field shall be used as reference for the second B field being coded. For B fields we always treat the number of reference fields (see NUMREF) as 2. B fields always use 4 reference fields (top and bottom forward, top and bottom backward) to predict the current MB.

The following are the salient features of B-field coding.

1) The first B field references the first and second fields from the previous and next anchor frames (Figure 93).

2) The second B field references the first B field from the current frame (i.e. the top B field in Figure 93) as the “opposite polarity” and the second field of the previous anchor frame (“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 93. If TFF = 0 then the bottom field is the first field and the top field is the second field.

3) We send the “forward/not forward” (0/1) decision (per MB) as a frame level bit (this is different from progressive B frames, where the frame level bit plane codes “direct/not direct”).

4) MV prediction follows similar logic as field P pictures, but we retain forward and backward contexts separately. We also fill in the “holes” e.g. when backward, we fill in the forward buffer’s MV with what would be the predicted MV.

5) We use 4MV for forward and backward modes (not direct and interpolated).

6) We also follow the MB mode (comprising 1/4MV/Intra, skipped MB, CBP present, 4MV block pattern) joint coding, new MV tables and MV architecture as defined for P field pictures.

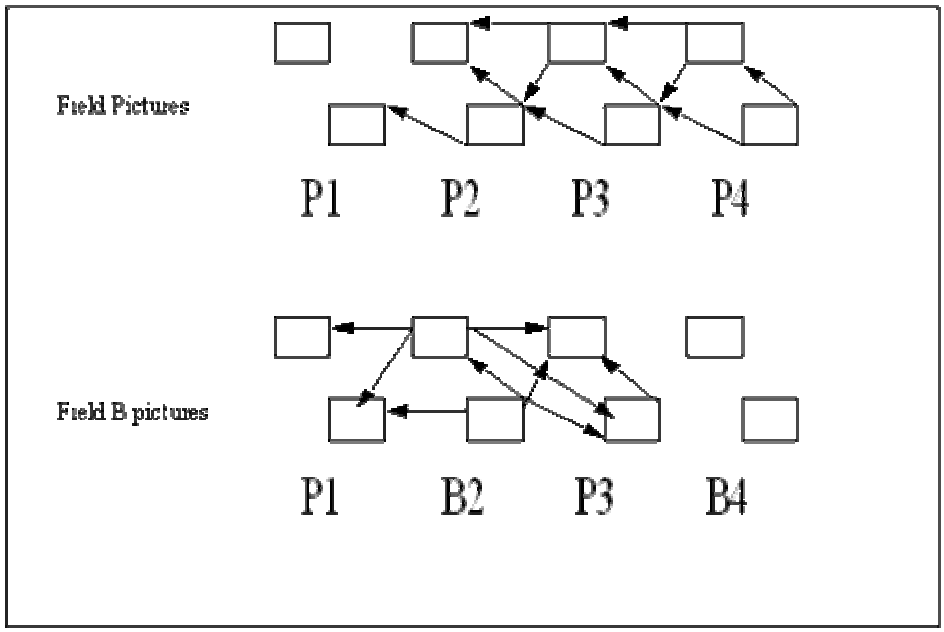


Figure 93: B field references

10.4.1 B Macroblock Layer Decode

At the MB level, the B field syntax is also similar to P field MB (e.g see Figure 81 and Figure 82). We will once again focus on describing the differences and avoid repeating the elements that remain the same.

10.4.1.1 Forward Bit

If the picture-level syntax element FORWARDMB is coded in raw mode, then the forward bit shall be used to signal forward/non-forward at the MB level.

10.4.1.2 MB Mode

In case the MB mode is not forward, we send additional bits in BMVTYPE to signal if the B-MB is backward, direct or interpolated. This is a simple VLC, where backward = 0, direct = 10, interpolated = 11.

10.4.1.3 Non-zero interpolated MV

If the MB is interpolated, then we send an extra bit to signal non-zero interpolated MV (INTERPMVP).

10.4.1.4 1 MV mode Motion Vectors (BMV1, BMV2)

In 1 MV mode, the first motion vector, BMV1 is signaled exactly the same way as MVDATA in P-fields 1 MV mode. If we are in interpolated mode and INTERPMVP = 1, then BMV1 is followed by BMV2, which corresponds to the backward reference motion vector.

10.4.1.5 4 MV mode Motion Vectors

Note that only forward and backward modes use 4MV, direct and interpolated always work with 1MV.

10.4.1.6 B Frame Modes

Macro blocks in B fields are identified as belonging to one of four modes, viz. *backward*, *forward*, *direct* and *interpolated*. The forward mode is akin to conventional P picture prediction. In the forward mode, the macro block is interpolated from its temporally previous anchor fields. Likewise, backward mode macro blocks are entirely interpolated from their temporally subsequent anchor frame.

Direct mode and interpolated mode macro blocks use both the anchors for prediction. Since there are two reference images for these modes, there are two motion vectors for each macro block. The direct mode implicitly derives these

motion vectors by appropriately scaling and bounding the motion vectors of the collocated macro block in the temporally subsequent 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 use round-up averaging for combining the pixel values of the two interpolated references into one set of macro block pixels:

Average pixel value = (Forward interpolated value + Backward interpolated value + 1) >> 1

The interpolation processes (e.g. bicubic, bilinear, quarter or half pel) are signaled and implemented exactly the same way as with P fields.

10.4.2 B Block Layer Decode

Block layer decoding is identical to P field pictures.

10.4.3 MV Prediction in B fields

Motion compensation is performed in both the forward and backward directions for B fields. The following describe how to perform motion vector prediction in B fields in the forward and backward directions.

10.4.3.1 Forward MV Prediction in B fields

Forward MV prediction for both B fields is identical to P field MV prediction as described in section 10.3.4.5.3. The only difference is in the method of deriving the reference frame distance. The forward reference frame distance is computed from the BFRACTION syntax element in the B field picture header and from the REFDIST syntax element in the backward reference frame header. The forward reference frame distance is computed as:

Forward Reference Frame Distance (FRFD) =

NINT ((BFRACTION numerator / BFRACTION denominator) * Reference Frame Distance) – 1

if (FRFD < 0) then FRFD = 0

NINT is the nearest integer operator as described in section 4.2.

The BFRACTION numerator and BFRACTION denominator are decoded from the BFRACTION syntax element as described in section 7.1.1.10.

The Reference Frame Distance is computed from the value decoded from the REFDIST syntax element in the backward reference frame. Reference Frame Distance = REFDIST + 1. Section 9.1.1.11 describes how to decode the REFDIST syntax element.

10.4.3.2 Backward MV Prediction in B fields

Backward MV prediction for the second B field in the frame is identical to P field MV prediction as described in section 10.3.4.5.3.

Backward MV prediction for the first B field in the frame is identical to P field MV prediction with exception that the MV scaling is different. For this case, scaleforopposite_x, scaleforopposite_y, scaleforsame_x and scaleforsame_y are defined as follows:

```
scaleforopposite_x (n)  {
    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
    }
```

```
    return scaledvalue
}
scaleforopposite_y (n)  {
    if (current field is top) {
        if (abs (n) < SCALEZONE1_Y)
            scaledvalue = ((n + 2) * SCALEOPP1) >> 8
        else {
            if (n < 0)
                scaledvalue = (((n + 2) * SCALEOPP2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = (((n + 2) * SCALEOPP2) >> 8) + ZONE1OFFSET_Y
        }
    }
    else {      //current field is bottom
        if (abs (n) < SCALEZONE1_Y)
            scaledvalue = ((n - 2) * SCALEOPP1) >> 8
        else {
            if (n < 0)
                scaledvalue = (((n - 2) * SCALEOPP2) >> 8) – ZONE1OFFSET_Y
            else
                scaledvalue = (((n - 2) * SCALEOPP2) >> 8) + ZONE1OFFSET_Y
        }
    }

    return scaledvalue
}

scaleforsame_x (n)  {
    int scaledvalue
    scaledvalue = (n * SCALESAME) >> 8
    return scaledvalue
}

scaleforsame_y (n)  {
    int scaledvalue
    if (current field is top)
        scaledvalue = ((n * SCALESAME) >> 8) – 2
    else   //current field is bottom
        scaledvalue = ((n * SCALESAME) >> 8) + 2
    return scaledvalue
}
```

The values of SCALESAME, SCALEOPP1, SCALEOPP2, SCALEZONE1_X, SCALEZONE1_Y, ZONE1OFFSET_X and ZONE1OFFSET_Y are shown in.

The backward reference frame distance is computed from the BFRACTION syntax element in the B field picture header and from the REFDIST syntax element in the backward reference frame header. The backward reference frame distance is computed as:

Backward Reference Frame Distance (BRFD) = Reference Frame Distance – FRFD – 1

The Reference Frame Distance is computed from the value decoded from the REFDIST syntax element in the backward reference frame. Reference Frame Distance = REFDIST + 1. Section 9.1.1.11 describes how to decode the REFDIST syntax element.

The forward reference frame distance (FRFD) is computed as described in section 10.4.3.1.

The value of N is dependant on the motion vector range. Extended motion vector range is signaled at the sequence level by the sequence header syntax element EXTENDED_MV= 1. If EXTENDED_MV = 1 then the MVRANGE syntax element is present in the picture header and signals the motion vector range. If EXTENDED_MV = 0 then the default motion vector range is used. Table 104 shows the relationship between N and the MVRANGE

Table 105: B Field Picture Backward MV Predictor Scaling Values for when Current Field is First

	Reference Frame Distance			
	1	2	3	4 or greater
SCALESAME	171	205	219	228
SCALEOPP1	384	320	299	288
SCALEOPP2	230	239	244	246
SCALEZONE1_X	32 * N	48 * N	53 * N	56 * N
SCALEZONE1_Y	8 * N	12 * N	13 * N	14 * N
ZONE1OFFSET_X	37 * N	20 * N	14 * N	11 * N
ZONE1OFFSET_Y	10 * N	5 * N	4 * N	3 * N

10.4.3.3 Motion Vector Prediction Process in B Fields

Only forward MVs are used as predictors for other forward MVs, and only backward MVs are used as predictors for backward MVs. For macroblocks that use either direct or interpolated prediction modes, we store both the forward and backward MV components. These components are computed implicitly in the case of direct and explicitly from received motion vector information in the case of interpolated mode.

If the forward or backward prediction mode is used then a motion vector for the other (missing) direction shall be derived for use as a predictor. To derive the motion vector, the motion vector prediction process as described above is peformed for the macroblock in the missing direction. The dominant polarity motion vector predictor is then used as the missing motion vector. In this way, both a complete set of forward and backward motion vectors is present for use in MV prediction.

The scheme for B field MV prediction is as follows –

1) If the MB is forward predicted, then median predict its MV from the neighborhood of the “forward MV buffer”. Store the forward MV (computed by Compute_MV__Predictors) in the forward buffer, and the dominant predictor in the backward buffer.
2) If the MB is backward predicted, then median predict its MV from the neighborhood of the “backward MV buffer”. Store the backward MV (computed by Compute_MV__Predictors) in the backward buffer, and the dominant predictor in the forward buffer.
3) If the MB is interpolated, then use the forward MV buffer to predict the forward component, the backward buffer to predict the backward component, and store the forward and backward MVs (both computed by Compute_MV__Predictors), once these have been calculated, in the forward and backward MV buffers, respectively.
4) If the MB is direct predicted, we compute the direct mode MVs as described in 8.4.3.2

All direct mode MV's (motion vectors) are treated as (0, 0) when the co-located macroblock (of the next anchor frame) is INTRA. This is also true when the previously decoded frame (i.e. the temporally "next" anchor frame) is an I-frame.

There shall be no computation of direct mode MV's for macroblocks (MB's) that are not using the direct mode prediction (e.g. forward or backward), unlike for progressive B frame. We predict them based on the forward or backward MV buffer from the current (B) field/frame, plus the rules of predicting motion vectors as defined with P fields/frames. Here's a simple example to make this clearer. Say we are decoding MB:(12,13) and it is forward predicted. Then its forward MV residual is explicitly sent in the bitstream. We decode this and insert it in the buffer of forward MV's (after adding to the predicted MV). We then take the backward MV buffer and use MV prediction logic to fill in the MV at position (12, 13) - e.g. if the prediction rule is a simple median of 3 neighbors, then we may take the median of backward MV's from (11, 13), (12, 12) and (13, 12) to fill in the backward MV for (12, 13).

In field interlaced B field coding, we use the collocated MB's MV from the (temporally next) P field with the same parity. If the P frame's collocated MV was 1 MV then that MV is simply buffered for the next B to be coded, else if 4 MV then we first consider the polarities of the 4 MV's to favor the dominant polarity. Thus, if the number of MV's (out of 4) from the same field outnumber those from the opposite field, we use median4, median3, arithmetic mean of 2 or the values of the same field MV's if we have 4, 3, 2, or 1 same field MV's respectively. Otherwise if the MV's from the opposite field outnumber those from the same field, we use similar operations to get a representative MV to use from the opposite field MV's. If more than two of the original set of 4 MV's, (irrespective of polarity) are intra, then we simply treat the collocated representative MV as intra, i.e. 0,0.

This selection process is elucidated by the following pseudo-code –

```
MotionVector SelectDirectModeMVFromColocatedMB ()
{
    MotionVector SelectedMV;

    if the corresponding MB used 1 MV
    then use that MV
    else  // 4 MV's to pick from
    {
        Count the number of Intra MV's // from among the 4 MVs of the
                                       // collocated macroblock

        if (Intra MV's > 2)
         then SelectedMV = INTRA
        else { // Use the motion vectors from the most dominant field

          Count the number of same field and opposite field MV's

            if (OppFieldCount > SameFieldCount)
            then use only the opposite field MV's in next step
            // i.e. opposite is the chosen polarity
          else use only the same field MV's in the next step
          // i.e. same is the chosen polarity

          Count the number of MV's of the chosen polarity
```

```
            if (Chosen MVs = 3) {
              SelectedMV = Median of 3 of the chosen MV's
            }
            else if (Chosen MVs = 2) {
                SelectedMV = Average of the chosen MV's
            }
            else if (Chosen MVs = 1) {
                SelectedMV = The chosen MV
            }
            else { // all 4 are of the chosen polarity
                 SelectedMV = Median of 4 of the chosen MV's
            }
        }
    }

    return (SelectedMV);
}
```

NB: The selection process outlined above is a pre-cursor to the actual scaling to produce the forward and backward pointing direct mode MV's, for which the steps are identical to those described above (in 8.4.3.2).

10.5 Interlace Frame I Picture Decoding

The following sections describe interlace frame I picture type.

10.5.1 Macroblock Layer Decode

The macroblocks are coded in raster scan order from left to right. Figure 83 shows the elements that make up the intra MB layer. Each macroblock may be either frame or field coded as indicated by FIELDTX which indicates the internal organization of a macroblock. FIELDTX = 1 indicates that the macroblock is field coded. For frame coded macroblocks, the luminance blocks is interlaced with each field occurring alternatively. For field coded macroblocks, the top two luminance blocks contain only the lines from top field while the bottom two luminance blocks contain only lines from the bottom field. The Cb/Cr blocks remain interlaced 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. Figure 94 shows the process used to reconstruct the 8x8 blocks.

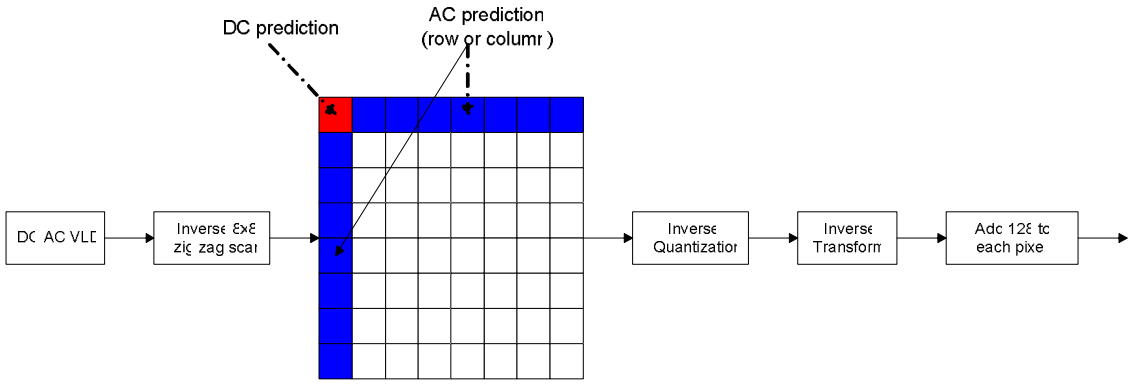


Figure 94: Intra Block Decode

The DC coefficients are coded differentially using neighboring block's DC coefficients. The quantized DC value for the current block is obtained by adding the DC predictor to the DC differential. The process of DC inverse quantization and DC differential decoding is the same as advanced profile I picture.

The ACPRED flag for each macroblock indicates whether some of AC coefficients are coded differentially. If the AC coefficients are differentially coded, then the AC coefficients for the current block is 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 is the same as advanced profile I picture.

After the inverse transform, we add 128 to each pixel in the block and clip it to be between 0 and 255 to form the decoded blocks. In addition, we permutate the decoded luminance blocks if the current macroblock is field coded.

10.5.2.1 DC Predictor

The DC predictor is obtained from one of the previously decoded adjacent blocks. Figure 33 shows the current block and the candidate predictors from the adjacent blocks. The values A, B and C represent the quantized DC values prior to the addition of 128 for the top-left, top and left adjacent blocks respectively. The FIELDTX flag does not have an effect of the DC/AC prediction process. For example, the adjacent blocks for block 0 of the 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*).

If all three blocks A, B, and C are present, then a prediction direction is formed based on the values of A, B and C and either the B or C predictor is chosen. The prediction direction is calculated the same way as the for I frame as shown in Figure 34.

If an adjacent block is missing, then the following rules apply:

- If block C is missing and block B is not, then use block B as the predictor.
- If block B is missing and block C is not, then use block C as the predictor.
- If both block B and block C are missing, then no predictor is used.
- If block A is missing and B, C are present, then we choose block B if the DC predictor for block C is smaller than the DC predictor for block B, otherwise, we choose block C.

In addition, the DC predictor is scaled in the same way as advanced profile I picture if MQUANT mode is on.

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 might be differentially encoded. The decision for the 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, then the top row of the current block is differentially coded.
- If DC is predicted from the left, then the left column of the current block is differentially coded.
- If DC is not predicted, then the AC coefficients are not differentially coded.

The AC coefficients in the predicted row or column are added to the corresponding decoded AC coefficients in the current block to produce the fully reconstructed quantized Transform coefficient block. In addition, the AC predictor is scaled in the same way as advanced profile I picture if MQUANT mode is on.

10.6 Interlace BI Frame Decoding

When B frames are used (in main and advanced profiles only), we code a special type of frame that is in some ways a hybrid of I and B frames. The syntax of BI frames is identical to that of I, but they are usually coded at higher QP's and can never be used as an anchor or reference frame to predict other frames.

10.7 Interlace Frame P Picture Decoding

The following sections describe interlace frame P picture type.

10.7.1 Out-of-bounds Reference Pixels

The previously interlaced frame is 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 may include pixel locations that are outside the boundary of the reference frame In these cases, the out-of-bounds pixel values are the replicated values of the edge pixel for the left and right boundary while the top and bottom boundaries are formed by repeating the top two and bottom two field lines thus preserving the interlace structure into the repeatpad region. The padding is conceptually considered to be infinite for the purpose of motion compensation. Note that in advanced profile, "frame edge", "frame corner" and "outside the boundary" refer to the true frame dimensions, not the dimensions right or top/bottom justified to the edge of the macroblock. In other words, the right and bottom pixels that are repeated to infinity for a 200 x 300 image begin at column 304 and row 208 for the simple and main profiles. However, for the advanced profile, these begin respectively at column 300 and row 200.

10.7.2 Macroblock Layer Decode

In interlace frame P picture, each macroblock may be motion compensated in frame mode using 1 or 4 motion vector(s) or in field mode using 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 does not contain any intra blocks. In additon, the residual after motion compensation may be coded in frame transform mode or field transform mode same as the interlace frame I picture. More specifically, the luminance component of the residual are re-arranged according to fields if it is coded in field transform mode and it remains unchanged in frame transform mode while the chroma component remains the same. A macroblock may also be coded as intra, in this case, the decoding process is the same as I macroblocks decoding in interlace frame I picture.

The motion compensation may be restricted to not include 4 (both field/frame) motion vectors and this is signaled through 4MVSWITCH. The type of motion compensation / residual coding is jointly indicated for each macroblock through MBMODE and SKIPMB. MBMODE employs different set of tables according to 4MVSWITCH.

Macroblocks in interlace frame P pictures are classified into 5 types: 1 MV, 2 Field MV, 4 Frame MV, 4 Field MV, and Intra. The first four types of macroblock are inter-coded while the last type indicates that the macroblock is intra-coded. The macroblock type is signaled by MBMODE syntax element in the macroblock layer along with the skip bit. MBMODE jointly encode macroblock types along with various pieces of information regarding the macroblock for different types of macroblock.

10.7.2.1 Inter Macroblock Types

The following sections describe four types of motion compensation:

10.7.2.1.1 1 MV Macroblock

In 1 MV macroblock, the displacement of the four luminance blocks is represented by a single motion vector. A corresponding chroma motion vector is derived to represent the displacements of each of the two 8x8 chroma blocks.

10.7.2.1.2 2 Field MV Macroblock

In 2 Field MV macroblock, the displacement of each field of the luminance blocks described by a different motion vector (see Figure 95). The top field motion vector describes the displacement of the even lines of the luminance blocks while the bottom field motion vector describes the displacement of the odd lines of the luminance blocks. Using the top field motion vector, we derive a corresponding top field chroma motion vector that describes the displacement of the even lines of the chroma blocks. Similarly, a bottom field chroma motion vector is derived from the bottom field motion vector that describes the displacements of the odd lines of the chroma blocks.

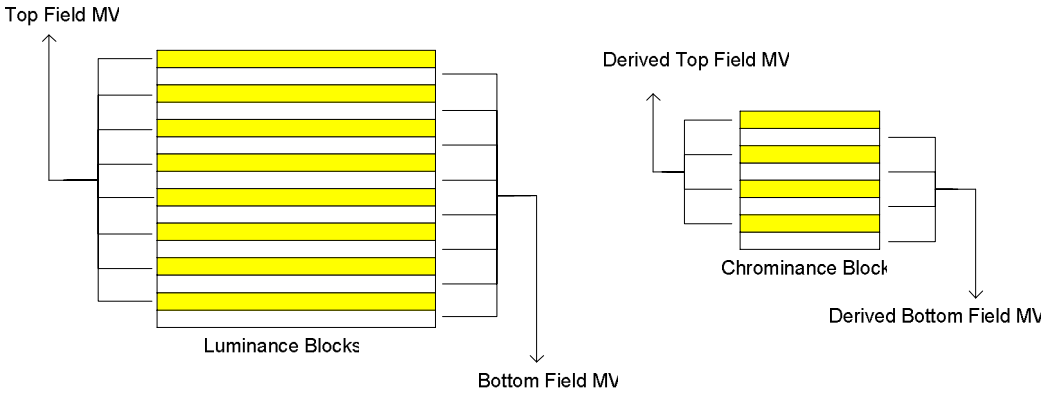


Figure 95: Two Field MV Macroblock

10.7.2.1.3 4 Frame MV Macroblock

In 4 Frame MV macroblock, each one of the four luminance block's displacement is described by a different motion vector (see Figure 96). Similarly, each chroma block is motion compensated using four derived chroma motion vector that describes the displacement of the four 4x4 subblocks. Each 4x4 subblock's chroma motion vector is derived (as described in section 10.7.2.6) from the spatially corresponding luminance block's motion vector. See details.

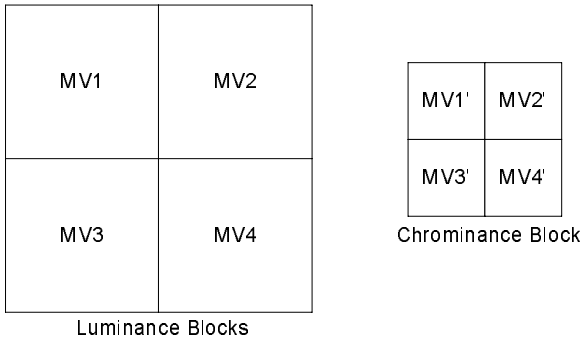


Figure 96: 4 Frame MV Macroblock

10.7.2.1.4 4 Field MV Macroblock

In 4 Field MV macroblock, the displacement of each field in the luminance blocks is described by two different motion vectors (see Figure 97). The even lines of the luminance blocks are subdivided vertically to form two eight by eight

regions. The displacement of the left region is described by the top left field block motion vector and the displacement of the right region is described by the top right field block motion vector. Similarly, the odd lines in the luminance blocks are subdivided vertically to form two eight by eight regions. The displacement of the left region is described by the bottom left field block motion vector and the displacement of the right region is described by the bottom right field block motion vector. Similarly, each chroma block is partitioned into four regions in the same way as the luminance blocks and each region is motion compensated using a derived field chroma motion vector.

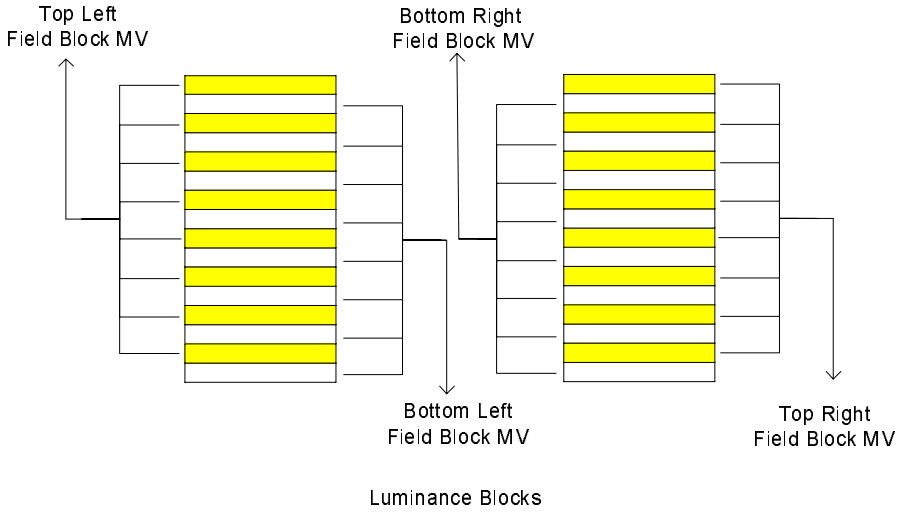


Figure 97: 4 Field MV Macroblock – Luminance Block

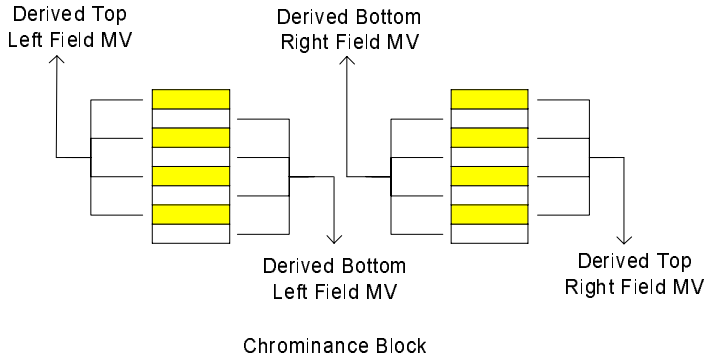


Figure 98: 4 Field MV Macroblock – Chrominance Block

10.7.2.2 Skipped Macroblock Signaling

The SKIPMB field indicates the skip condition for a macroblock. If the SKIPMB field is 1, then the current macroblock is said to be skipped and there are 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).

On the other hand, if the SKIPMB field is not 1, then the MBMODE field will have be decoded to indicate the type of macroblock and various other key pieces of information regarding the current macroblock.

10.7.2.3 Macroblock Mode Signaling

There are fifteen possible events that are indicated by MBMODE which jointly specifies the type of macroblock (inter-1mv, 4mv, 2 field mv, 4 field mv, or intra), types of transform for inter-coded macroblock (i.e. field or frame or no coded blocks), and in addition, whether there is differential motion vector for the 1MV macroblock.

Let <MVP> denote a binary event that signals whether there is nonzero 1 MV differential motion vector or not.

Let <Field/Frame transform> denote a ternary event that signals whether the residual of the macroblock is frame transform coded, field transform coded, or zero coded blocks (i.e. CBP = 0).

Then the MBMODE signals the following set of events jointly:

> MBMODE = { <1MV, MVP, Field/Frame transform>, <2 Field MV, Field/Frame transform>, <4 Frame MV, Field/Frame transform>, <4 Field MV, Field/Frame transform>, <INTRA>}; excluding the event where <1MV, MVP=0, CBP=0> which is signaled by the skip condition.

For inter-coded macroblocks, the CBPCY syntax element shall not be be decoded when the Field/Frame Transform event in MBMODE indicates no coded blocks. On the other hand, if the Field/Frame transform event in MBMODE indicates field or frame transform, then CBPCY shall be decoded.

For non-1MV inter-coded macroblocks, an additional field is sent to indicate the zero differential motion vectors event. In the case of 2 Field MV macroblocks, the 2MVBP field is sent to indicate which of the two motion vectors contain nonzero differential motion vectors. Similarly, the 4MVBP field is 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 are coded in separate fields.

10.7.2.4 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 99 shows the neighboring macroblock from which we gather the candidate motion vectors from. The order of the collection of candidate motion vectors is important and it always start from A, to B, and ends at C.

Second, the motion vector predictor(s) for the current macroblock is computed from the set of candidate motion vectors.

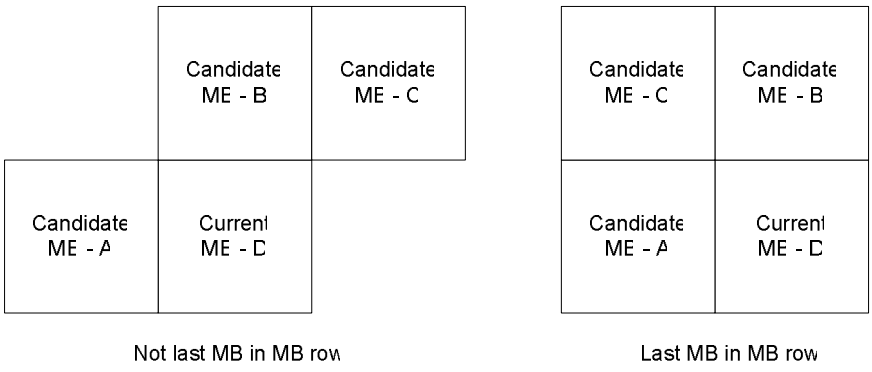


Figure 99: Candidate Neighboring Macroblocks for Interlace Frame Picture

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.

10.7.2.4.1 1 MV Candidate Motion Vectors Derivation

The following pseudocode is 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 vector.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vector.
    } else if (A is 2 Field MV) {
        Average the two field motion vectors of A and add the
        resulting MV to the set of candidate motion vector.
    } else if (A is 4 Field MV) {
        Average 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 vector.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Average the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vector.
    } else if (B is 4 Field MV) {
        Average 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 vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } else if (C is 2 Field MV) {
        Average the two field motion vectors of C and add the
        resulting MV to the set of candidate motion vector.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Average 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 vector.
        } else { // C is top left MB
            Average 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 vector.
        }
    }
}
```

10.7.2.4.2 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 following pseudocode is 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 vector.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vector.
    } else if (A is 2 Field MV) {
        Average the two field motion vectors of A and add the
        resulting MV to the set of candidate motion vector.
    } else if (A is 4 Field MV) {
```

```
        Average 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 vector.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Average the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vector.
    } else if (B is 4 Field MV) {
        Average 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 vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } else if (C is 2 Field MV) {
        Average the two field motion vectors of C and add the
        resulting MV to the set of candidate motion vector.
    } else if (C is 4 Field MV) {
        if (C is top right MB) {
            Average 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 vector.
        } else { // C is top left MB
            Average 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 vector.
        }
    }
}
```

The following pseudocode is 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 vector.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom right block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Average the two field motion vectors of B and add the
        resulting MV to the set of candidate motion vector.
    } else if (B is 4 Field MV) {
        Average 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 vector.
    }
}

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 vector.
```

```
            } else { // C is top left MB
                Add the bottom right block MV of C to the set of
                candidate motion vector.
            }
        } else if (C is 2 Field MV) {
            Average the two field motion vectors of C and add the
            resulting MV to the set of candidate motion vector.
        } else if (C is 4 Field MV) {
            if (C is top right MB) {
                Average 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 vector.
            } else { // C is top left MB
                Average 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 vector.
            }
        }
    }
```

The following pseudocode is 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 vector.
        } else if (A is 4 Frame MV) {
            Add the bottom right block MV of A to the set of
            candidate motion vector.
        } else if (A is 2 Field MV) {
            Average the two field motion vectors of A and add the
            resulting MV to the set of candidate motion vector.
        } else if (A is 4 Field MV) {
            Average 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 vector.
        }
    }
```

```
Add the top left block MV of the current MB to the set of
candidate motion vector.

Add the top right block MV of the current MB to the set of candidate
motion vector.
```

The following pseudocode is 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 vector.

Add the top left block MV of the current MB to the set of candidate motion
vector.

Add the top right block MV of the current MB to the set of candidate
motion vector.
```

10.7.2.4.3 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 following pseudocode is 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 vector.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vector.
    } else if (A is 2 Field MV) {
        Add the top field MV of A to the set of candidate motion
        vector.
    } else if (A is 4 Field MV) {
        Add the top right field block MV of A to the set of
        candidate motion vector.
    }
}

if (B exists and B is not intra coded) {
```

```
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of candidate motion
        vector.
    } else if (B is 4 Field MV) {
        Add the top left field block MV of B to the set of
        candidate motion vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } 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 vector.
        } else { // C is top left MB
            Add the top right field block MV of C to the set of
            candidate motion vector.
        }
    }
}
```

The following pseudocode is 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 vector.
    } else if (A is 4 Frame MV) {
        Add the bottom right block MV of A to the set of
        candidate motion vector.
    } else if (A is 2 Field MV) {
        Add the bottom field MV of A to the set of candidate
      motion vector.
    } else if (A is 4 Field MV) {
        Add the bottom right field block MV of A to the set of
        candidate motion vector.
    }
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Add the bottom field MV of B to the set of candidate
        motion vector.
    } else if (B is 4 Field MV) {
        Add the bottom left field block MV of B to the set of
        candidate motion vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } else if (C is 2 Field MV) {
        Add the bottom 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 bottom left field block MV of C to the set of
            candidate motion vector.
        } else { // C is top left MB
            Add the bottom right field block MV of C to the set
            of candidate motion vector.
        }
    }
}
```

10.7.2.4.4 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 following pseudocode is 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 vector.
    } else if (A is 4 Frame MV) {
        Add the top right block MV of A to the set of
        candidate motion vector.
    } else if (A is 2 Field MV) {
        Add the top field MV of A to the set of
        candidate motion vector.
    } else if (A is 4 Field MV) {
        Add the top right field block MV of A to the set of
        candidate motion vector.
    }
```

```
}

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom left block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of
        candidate motion vector.
    } else if (B is 4 Field MV) {
        Add the top left field block MV of B to the set of
        candidate motion vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } 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 vector.
        } else { // C is top left MB
            Add the top right field block MV of C to the set of
          candidate motion vector.
```

```
        }
    }
}
```

The following pseudocode is 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 vector.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
        Add MV of B to the set of candidate motion vector.
    } else if (B is 4 Frame MV) {
        Add the bottom right block MV of B to the set of
        candidate motion vector.
    } else if (B is 2 Field MV) {
        Add the top field MV of B to the set of
        candidate motion vector.
    } else if (B is 4 Field MV) {
        Add the top right field block MV of B to the set of
        candidate motion vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } 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 vector.
            } else { // C is top left MB
                Add the top right field block MV of C to the set of
               candidate motion vector.
            }
        }
    }
```

The following pseudocode is 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 vector.
        } else if (A is 4 Frame MV) {
            Add the bottom right block MV of A to the set of
            candidate motion vector.
        } else if (A is 2 Field MV) {
            Add the bottom field MV of A to the set of
            candidate motion vector.
        } else if (A is 4 Field MV) {
            Add the bottom right field block MV of A to the set of
            candidate motion vector.
        }
    }

    if (B exists and B is not intra coded) {
        if (B is 1 MV) {
            Add MV of B to the set of candidate motion vector.
        } else if (B is 4 Frame MV) {
            Add the bottom left block MV of B to the set of
            candidate motion vector.
        } else if (B is 2 Field MV) {
            Add the bottom field MV of B to the set of
            candidate motion vector.
```

```
    } else if (B is 4 Field MV) {
        Add the bottom left field block MV of B to the set of
        candidate motion vector.
    }
}

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 vector.
        } else { // C is top left MB
            Add the bottom right block MV of C to the set of
            candidate motion vector.
        }
    } else if (C is 2 Field MV) {
        Add the bottom 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 bottom left field block MV of C to the set of
          candidate motion vector.
        } else { // C is top left MB
            Add the bottom right field block MV of C to the set
            of candidate motion vector.
        }
    }
}
```

The following pseudocode is 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 vector.

if (B exists and B is not intra coded) {
    if (B is 1 MV) {
```

```
            Add MV of B to the set of candidate motion vector.
        } else if (B is 4 Frame MV) {
            Add the bottom right block MV of B to the set of
            candidate motion vector.
        } else if (B is 2 Field MV) {
            Add the bottom field MV of B to the set of
            candidate motion vector.
        } else if (B is 4 Field MV) {
            Add the bottom right field block MV of B to the set of
            candidate motion vector.
        }
    }

    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 vector.
            } else { // C is top left MB
                Add the bottom right block MV of C to the set of
                candidate motion vector.
            }
        } 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 bottom left field block MV of C to the set of
               candidate motion vector.
            } else { // C is top left MB
                Add the bottom right field block MV of C to the set
                of candidate motion vector.
            }
        }
    }
```

10.7.2.4.5 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) is:

$MVX_A = (MVX_1 + MVX_2 + 1) >> 1;$

$MVY_A = (MVY_1 + MVY_2 + 1) >> 1;$

10.7.2.4.6 Computing Frame MV predictor(s) from Candidate Motion Vectors

This section describes how we compute the MV predictor for frame MVs given the set of candidate motion vectors. The operation is 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 following pseudocode describes how the MV predictor (PMV_x, PMV_y) is 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;
}
```

10.7.2.4.7 Computing Field MV predictor(s) from Candidate Motion Vectors

This section describes how we compute the MV predictor(s) for field MVs given the set of candidate motion vectors. The operation is 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 we may check whether a candidate motion vector points to the same field by the following check on its y component:

```
if (ValidMVy & 4) {
     ValidMV points to the opposite field.
} else {
     ValidMV points to the same field.
}
```

Let SameFieldMV, OppFieldMV denote the two sets and let NumSameFieldMV and NumOppFieldMV denote the number of motion vectors that belongs to each set. The following pseudocode describes 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;
}
```

10.7.2.5 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 may be 0 up to 4 MVDATA syntax elements per macroblock. More specifically,

- For 1 MV macroblocks, there may be either 0 or 1 MVDATA syntax element present depending on the MVP field in MBMODE.

- For 2 Field MV macroblocks, there may be either 0, 1, or 2 MVDATA syntax element(s) present depending on 2MVBP.
- For 4 Frame / Field MV macroblocks, there may be either 0, 1, 2, 3, or 4 MVDATA syntax element(s) present depending on 4MVBP.

The motion vector differential is decoded the same way as one reference field motion vector differential for field P picture described in section 10.3.4.5.2.1 with no halfpel mode.

10.7.2.6 Reconstructing Motion Vectors

Given the motion vecotor differential dmv, the luminance motion vector is reconstructed by adding the differential to the predictor as follows:

$$mv\_x = (dmv\_x + predictor\_x) \text{ smod } range\_x$$

$$mv\_y = (dmv\_y + predictor\_y) \text{ smod } range\_y$$

The modulus operation "smod" is a signed modulus, defined as follows:

$$A \text{ smod } b = ((A + b) \,\&\, (2\,b - 1)) - b$$

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 66.

Given a luma frame or field motion vector, a corresponding chroma frame or field motion vector is derived to compensate a portion (could be the entire portion) of the Cb/Cr block. The FASTUVMC syntax element is ignored in interlace frame P, B pictures. The following pseudocode describes how a chroma 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};
CMV_X = (LMV_X + s_RndTbl[LMV_X & 3]) >> 1;
if (LMV is a field motion vector) {
    CMV_Y = (LMV_Y >> 4)*8 + s_RndTblField [LMV_Y & 0xF];
   } else {
   CMV_Y = (LMV_Y + s_RndTbl[LMV_Y & 3]) >> 1;
   }
```

10.7.3 Block Layer Decode

If the current macroblock is intra-coded, then the block layer is equivalent to decoding of an macroblock in interlace frame I pictures as described 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 is the same as described in section 10.3.5.2.

10.8 Interlace Frame B Picture Decoding

Interlace frame B picture decoding is very similar to interlace P picture decoding in terms of bitstream syntax. We will focus here on the differences in bitstream and decoding steps, and omit the elements that are identical, for the sake of brevity. The additional elements are: a) the BFRACTION at the picture level that tells us how to scale the direct mode MVs; b) the DIRECTMB bit plane coding that is sent at the picture layer for direct/non-direct MB's; c) the DIRECTBIT bit at the MB level in the case where the direct mode bit plane is coded raw; d) the BMVTYPE VLC at the MB level that indicates if the MB is forward, backward or interpolated; and e) one bit (MVSW) at the MB level if

we are in field mode and BMVTYPE is forward or backward, to indicate if we are going to switch mode from forward to backward (or backward to forward) in going from the top to the bottom field's MV.

As with B frame MV decoding in progressive and field coding, we maintain 2 buffers, one each for forward and backward motion, and use the rule "forward predicts forward, backward predicts backward". MV prediction follows similar logic as frame P pictures, but we retain forward and backward contexts separately. We also fill in the "holes" e.g. when backward, we fill in the forward buffer's MV with what would be the predicted MV.

10.8.1 B Macroblock Layer Decode

At the MB level, the B frame syntax is also similar to P frame MB (e.g see Figure 84 and Figure 85). We will once again focus on describing the deltas and avoid repeating the elements that remain the same.

10.8.1.1 Direct Bit

If the picture-level syntax element DIRECTMB is coded in raw mode, then the direct bit shall be used to signal direct/non-direct at the MB level.

10.8.1.2 BMV Type

In case the MB mode is not direct, we send additional bits in BMVTYPE to signal if the B-MB is forward, backward or interpolated. BMVTYPE is a variable sized syntax element present in B frame macroblocks that indicates whether the macroblock uses forward, backward or interpolated prediction. As Table 46 shows, the value of BFRACTION in the picture header along with BMVTYPE determine whether forward or backward prediction are indicated.

10.8.1.3 Field level MV Switch (MVSW)

If the MB is forward or backward AND the MB is "field" type, then we send an additional bit to signal if we are going to switch from forward to backward (or backward to forward) in going from the top to the bottom field's MV.

10.8.1.4 B Frame Modes

Macro blocks in B frames are identified as belonging to one of four modes, viz. *backward*, *forward*, *direct* and *interpolated*. Additionally, in frame mode B pictures we have the ability to switch from backward to forward or forward to backward at the field level – this applies to field mode MB's. The forward mode is akin to conventional P picture prediction. In the forward mode, the B macro block is interpolated from its temporally previous anchor frame. Likewise, backward mode macro blocks are entirely interpolated from their temporally subsequent anchor frame.

Direct mode and interpolated modes are implemented according to the description in 8.4.3.2.

The interpolation processes (e.g. bicubic, bilinear, quarter or half pel) are signaled and implemented exactly the same way as with frame P pictures.

10.8.1.5 Skipped MB's

Skipped MB's in interlace frame coded B frames are constrained to always use 1 MV mode (i.e. not field coded). So there may only be 1 MV for forward and backward, and 2 for direct and interpolated.

10.8.1.6 Motion Vector Prediction in Frame B Pictures

MV prediction for frame B pictures follows exactly the same rules as with frame P, and will not be repeated here. The only additional point to note is that two separate MV buffers are kept for forward and backward MV's, and the MV prediction rules are applied on each of these while decoding an MV of the like type, i.e. forward MV's are used to predict an incoming forward MV, and backward MV's are used to predict an incoming backward MV. In the interpolated mode we use both forward and backward prediction to predict the two incoming MV's, and in the direct mode we scale the next field P's collocated MV. If an MB is "skipped", the MB mode shall be signaled to identify whether the "skipped" MB uses direct, forward, backward or interpolated prediction. Skipping in the context of B frames shall imply that the MV prediction error is zero, i.e. decoding is performed as usual (treating each mode with the appropriate decoding rules), and the predicted MV's will be exactly the ones we use.

The scheme for frame B MV prediction is as follows –

1) If the MB is forward predicted, then median predict its MV from the neighborhood of the “forward MV buffer”. Store the forward MV (after adding the prediction error) in the forward buffer, and the backward predicted MV in the backward buffer. If we are in field mode and MVSW = 1, i.e. we switch from forward to backward mode in going from the top to the bottom field, then we fill in the forward MV into both the top and bottom MV “slots” of the forward MV buffer, and the backward MV into both the top and bottom MV slots of the backward buffer, i.e. although the forward MV is being sent only for the top field, we fill in the same MV into both top and bottom field MV slots for the forward MV buffer, and although the backward MV is being sent only for the bottom field, we insert it into both top and bottom field slots of the backward MV buffer.

2) If the MB is backward predicted, then median predict its MV from the neighborhood of the “backward MV buffer”. Store the backward MV (after adding the prediction error) in the backward buffer, and the forward predicted MV in the forward buffer. If we are in field mode and MVSW = 1, i.e. we switch from backward to forward mode in going from the top to the bottom field, then we fill in the forward MV into both the top and bottom MV “slots” of the forward MV buffer, and the backward MV into both the top and bottom MV slots of the backward buffer, i.e. although the backward MV is being sent only for the top field, we fill in the same MV into both top and bottom field MV slots for the backward MV buffer, and although the forward MV is being sent only for the top field, we insert it into both top and bottom field slots of the backward MV buffer.

3) If the MB is interpolated, then use the forward MV buffer to predict the forward component, the backward buffer to predict the backward component, and store the forward and backward MVs (after adding the two sets of prediction errors), once these have been calculated, in the forward and backward MV buffers respectively.

4) If the MB is direct predicted, we compute the direct mode MVs as follows -

For a basic description and for pseudo-code of how the direct mode scales MVs, please refer to section 8.4.3.2. Here are some differences for interlaced B frame’s direct mode decoding –

In frame interlaced coding (advanced profile), we buffer ALL the decoded luma MV's from the future P frame, i.e. all 4 * NumberOfMB's MV's. Each interlaced P-frame MB has four possible MV slots, where a slot is a buffer which may hold the X and Y components of the motion vector. If the MB is 1 MV then all 4 will have the same MV. If 2 MV then the two "top field" slots will have the same MV, and the "bottom field" slots will have another MV. If 4MV frame/field, then all 4 slots will have different MV's. Now when we get a B frame, we will generate different numbers of (forward/backward) MV pairs depending NOT on how the co-located MB was coded, but on how the current B's MB is being coded, i.e. if the direct mode MB is 1 MV coded then we will simply take MV from the top-left slot (and generate 1 pair of direct MV's), if it is field coded then we take the top-left and bottom-left and generate 2 pairs, one for each field and so on.In frame interlaced B frames, the direct mode computes 2 MV's in each direction, one for each field if the MB is field coded. Otherwise we have 1 MV in each direction (if the MB is frame-coded). Important note - when we say "if the MB is field coded" we are talking about the B frame's MB, and not the co-located MB from the future P picture. As with progressive and field mode, all direct mode MV’s (motion vectors) are treated as (0, 0) when the co-located macroblock (of the next anchor frame) is INTRA. This is also true when the previously decoded frame (i.e. the temporally "next" anchor frame) is an I-frame.

Also, in interlaced B frames, there shall be no calculation of direct mode MV’s for macroblocks (MB’s) that are not using the direct mode prediction (e.g. forward or backward). They are predicted based on the forward or backward MV buffer from the current (B) field/frame, plus the rules of predicting motion vectors as defined with P frames. Here's a simple example to make this clearer. Say we are decoding MB:(12,13) and it is forward predicted. Then its forward MV residual is explicitly sent in the bitstream. We decode this and insert it in the buffer of forward MV's (after adding to the predicted MV). We then take the backward MV buffer and use MV prediction logic to fill in the MV at position (12, 13) - e.g. if the prediction rule is a simple median of 3 neighbors, then we may take the median of backward MV's from (11, 13), (12, 12) and (13, 12) to fill in the backward MV for (12, 13).

10.8.2 B Block Layer Decode

Block decoding syntax and operations are the same as for P pictures and will not be repeated. I-MB’s in B frames are also the same as those in P frames.

10.9 Overlapped Transform

If the sequence layer syntax element OVERLAP is set to 1, then a filtering operation is conditionally performed across edges of two neighboring Intra blocks, for both the luminance and chrominance channels. This filtering operation (referred to as *overlap smoothing*) is performed subsequent to decoding the frame, and prior to in-loop deblocking. However, overlap smoothing may be done after the relevant macroblock slices are decoded as this is functionally equivalent to smoothing after decoding the entire frame.

Overlapped transforms are modified block based transforms that exchange information across the block boundary. With a well designed overlapped transform, blocking artifacts may be minimized. For intra blocks, VC-9 simulates an overlapped transform by coupling an 8x8 DCT-like block transform with overlap smoothing. Edges of an 8x8 block that separate two intra blocks are smoothed – in effect an overlapped transform is implemented at this interface.

Figure 100 shows a portion of a P frame with I blocks. This could be either the Y or U/V 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 100 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.

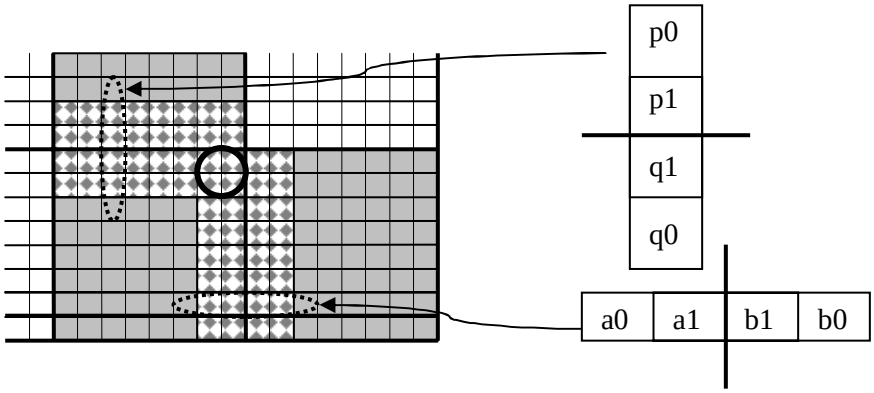


Figure 100: Example showing overlap smoothing

10.9.1 Overlap Smoothing

The overlap smoothing in interlace field pictures is identical to the overlap smoothing for progressive I-frames in advanced profile, and is described in Section 8.5.2.

10.9.2 Overlap Smoothing for Interlace Frame Pictures

The overlap smoothing process in interlace frame pictures is the same as the above description with the exception that only the vertical edges between I blocks are filtered and no horizontal block boundaries are filtered. The conditional overlap smoothing applies only to interlace frame I picture. The vertical edges between two horizontally adjacent macroblocks is smoothed if and only if both macroblocks have their respective OVERFLAG bits set to 1, or the entire frame is enabled for overlap filtering implicitly (based on PQUANT) or explicitly (based on CONDOVER).

10.10 In-loop Deblock Filtering

If the sequence layer syntax element LOOPFILTER = 1, then a filtering operation is performed on each reconstructed frame. This filtering operation is performed prior to using the reconstructed frame 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 operates on the pixels that border neighboring blocks. For P pictures, the block boundaries may occur at every 4th, 8th, 12th, 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 8th, 16th, 24th, etc pixel row and column.

10.10.1 I Field Picture In-loop Deblocking

For I pictures, deblock filtering is performed at all 8x8 block boundaries. Figure 59 and Figure 60 show the pixels that are filtered along the horizontal and vertical border regions. The figures show the upper left corner of a component (luma, Cb or Cr) plane. The crosses represent pixels and the circled crosses represent the pixels that are filtered.

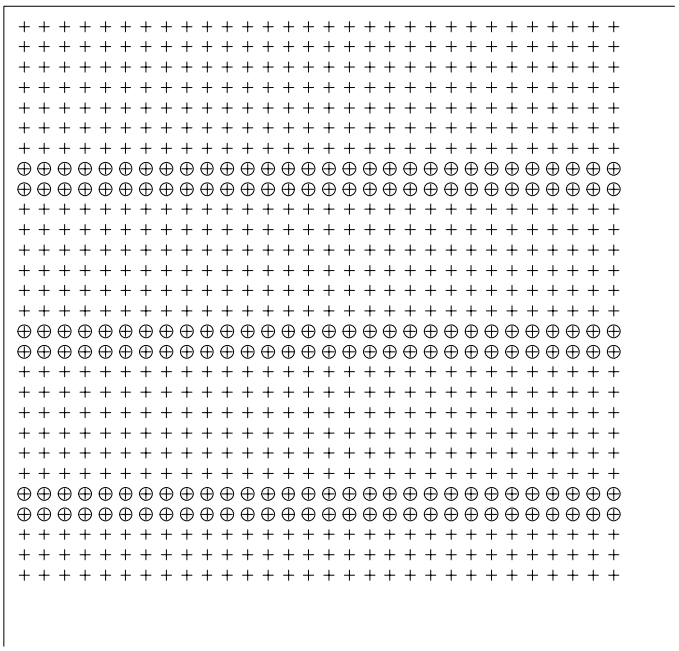


Figure 101: Filtered horizontal block boundary pixels in I picture

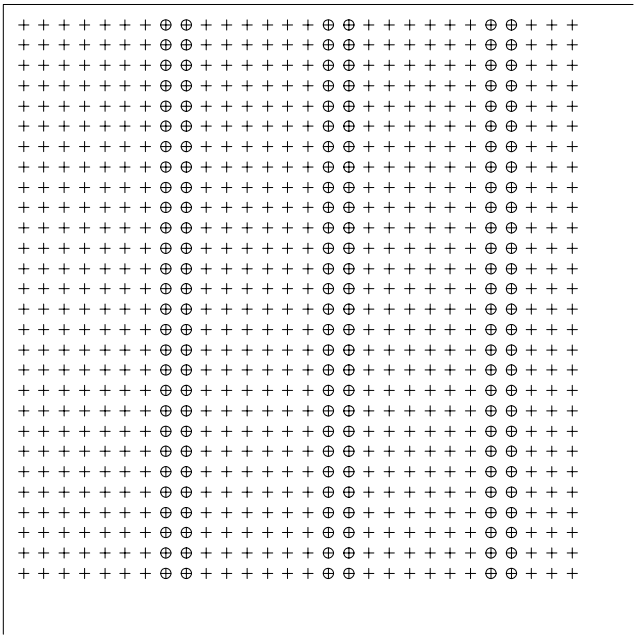


Figure 102: Filtered vertical block boundary pixels in I picture

As the figures show, the top horizontal line and first vertical line are not filtered. Although not depicted, the bottom horizontal line and last vertical line are also not filtered. In more formal terms, the following lines are 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 are filtered first followed by the vertical boundary lines.

10.10.2 P Field Picture In-loop Deblocking

Section 8.6.2 describes in-loop deblocking of P pictures and section 8.6.4 describes the filtering process. Note that the boundary between a block or subblock and a neighbouring 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, it is filtered.

10.10.3 B Field Picture In-loop Deblocking

This is exactly the same as P field pictures (and therefore progressive deblocking for P frames), with one important difference, which is that we do not use the motion vectors (and the presence of transform coefficients) in any loopfiltering decisions, i.e. the following criterion is not applied to B field pictures -

“The boundary between a block or subblock and a neighboring block or subblock is not filtered if both have the same motion vector and both have no residual error (no Transform coefficients). Otherwise it is filtered.”

10.10.4 Interlace Frame Pictures In-loop Deblocking

In interlace frame pictures, each macroblock may be frame transform coded or field transform coded according to its FIELDTX flag. The state of the FIELDTX flag along with the size of the transform (4x4, 4x8, 8x4, 8x8) used has an effect on where the in-loop deblocking takes place in the macroblock.

10.10.4.1 Field-based Filtering

The filtering process is the same as described in section 8.6.4 with one important difference, the filtering is always done using the same field lines, never mixing different field. Figure 103 illustrates the field-based filtering for horizontal and vertical block boundaries.

For a horizontal block boundary, we filter the two top field lines across the block boundary using top field lines only and the two bottom field lines across the block boundary using bottom field lines only. For a vertical block boundary, we filter the top field block boundary and the bottom field block boundary separately.

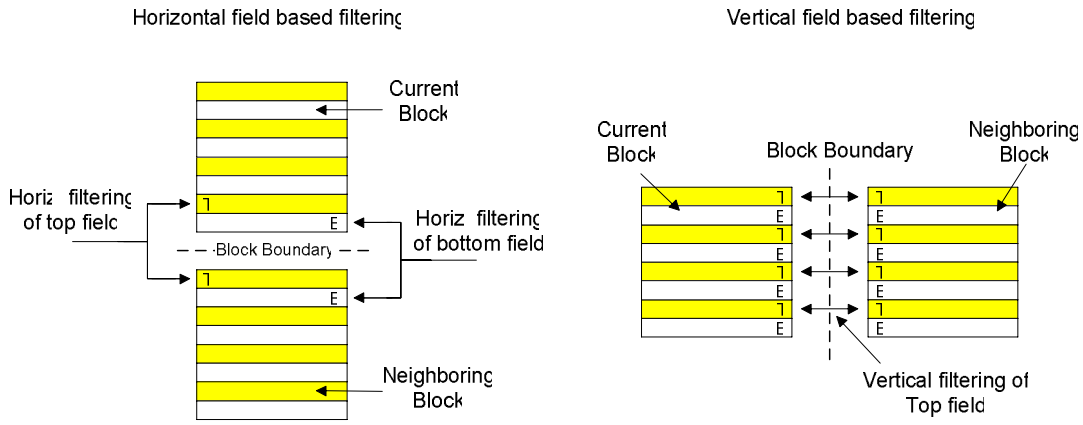


Figure 103: Field based horizontal / vertical block boundaries filtering

10.10.4.2 Filtering order

For both inter (P, B) and intra (I) frame picture, the in-loop deblocking process starts by processing all the horizontals edges first followed by all the vertical edges. The horizontal edges are processed a macroblock at a time following the raster scan order. Similarly, the vertical edges are processed a macroblock at a time following the raster scan order. The following pseudocode decribes this filtering process one macroblock at a time for the sake of simplicity, but alternate valid implementations of the filtering process may not follow this macroblock processing order. The only requirement is that multiple filtering operations on the same pixels shall follow the same filtering order as that given in the following pseudo-code:

```
// Processing horizontal edges
for (Y = 0; Y < number of MBs in a MB row; Y++) {
     for (X = 0; X < number of MBs in a MB col; X++) {
          Filter horizontal edges of MB located at Yth row, Xth col
     }
}

// Processing vertical edges
for (Y = 0; Y < number of MBs in a MB row; Y++) {
     for (X = 0; X < number of MBs in a MB col; X++) {
          Filter vertical edges of MB located at Yth row, Xth col
     }
}
```

10.10.4.3 Interlace Frame I Picture

In interlace frame I picture, each macroblock is 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 following pseudocode describes the process of horizontal filtering a macroblock:

```
// Horizontal filtering of MB
// Luminance
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.
}

- Filter all 16 pixels in row 14 and 16 of Y.
- Filter all 16 pixels in row 15 and 17 of Y.
```

```
// Chrominance
- 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.
```

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 following pseudocode describes the process of the vertical filtering a macroblock:

```
// Vertical filtering of MB
// Luminance
- 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.

- Filter the 8 even numbered pixels in column 15 and 16 of Y.
- Filter the 8 odd numbered pixels in column 16 and 16 of Y.

// Chrominance
- Filter the 4 even numbered pixels in column 7 and 8 of U.
- Filter the 4 odd numbered pixels in column 7 and 8 of U.
- Filter the 4 even numbered pixels in column 7 and 8 of V.
- Filter the 4 odd numbered pixels in column 7 and 8 of V.
```

10.10.4.4 Interlace Frame P, B Picture

In interlace frame P, B picture, each macroblock may be 4x4, 4x8, 8x4, or 8x8 transform coded.

For each macroblock, the horizontal block boundary filtering occurs in the order of block Y_0, Y_1, Y_2, Y_3, U, and then V. The luminance blocks are processed differently according to FIELDTX. The following pseudocode describes the process of horizontal filtering a macroblock:

```
// Horizontal filtering of MB
// Luminance
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.
     }
}

// Chrominance
if (current MB is not in the first or last MB row and the
    transform used for the U block is 8x4 or 4x4) {
     - Filter all 8 pixels in row 2 and 4 of U.
     - Filter all 8 pixels in row 3 and 5 of U.
}
If (current MB is not in the last MB column) {
     - Filter all 8 pixels in row 6 and 8 of U.
     - Filter all 8 pixels in row 7 and 9 of U.
}

if (current MB is not in the first or last MB row and the
    transform used for the V block is 8x4 or 4x4) {
     - Filter all 8 pixels in row 2 and 4 of V.
     - Filter all 8 pixels in row 3 and 5 of V.
}
```

```
If (current MB is not in the last MB column) {
     - Filter all 8 pixels in row 6 and 8 of V.
     - Filter all 8 pixels in row 7 and 9 of V.
}
```

Similarly, for each macroblock, the vertical block boundary filtering occurs in the order of block Y_0, Y_1, Y_2, Y_3, U, and then V. The luminance blocks are processed differently according to FIELDTX. The following pseudocode describes the process of vertical filtering a macroblock:

```
// Vertical filtering of MB
// Luminance
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.
     }
}

// Chrominance
if (the transform of U Block is 4x8 or 4x4) {
    - Filter the 4 even numbered pixels
      in column 3 and 4 of U.
    - Filter the 4 odd numbered pixels
      in column 3 and 4 of U.
}
if (current MB is not in the last MB column) {
    - Filter the 4 even numbered pixels
      in column 7 and 8 of U.
    - Filter the 4 odd numbered pixels
      in column 7 and 8 of U.
}

if (the transform of V Block is 4x8 or 4x4) {
    - Filter the 4 even numbered pixels
      in column 3 and 4 of V.
    - Filter the 4 odd numbered pixels
      in column 3 and 4 of V.
}
if (current MB is not in the last MB column) {
    - Filter the 4 even numbered pixels
      in column 7 and 8 of V.
```

```
        - Filter the 4 odd numbered pixels
          in column 7 and 8 of V.
    }
```

11 Tables

11.1 Interlace Pictures MV Block Pattern VLC Tables

11.1.1 4MV Block Pattern Tables

Table 106: 4MV Block Pattern Table 0

4MV 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 107: 4MV Block Pattern Table 1

4MV 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 108: 4MV Block Pattern Table 2

4MV 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 109: 4MV Block Pattern Table 3

4MV 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 2MV Block Pattern Tables

Table 110: 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 111: 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 112: 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 113: 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 114: 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 115: 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 116: 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 117: 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 118: 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 119: 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 120: 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 121: 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 122: 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 123: 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 124: 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 125: 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 126: 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 127: 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 128: 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 129: 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 130: 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 131: 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 132: 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 133: 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 134: 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 135: 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 136: 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 137: 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 138: 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 139: 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 140: 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 141: 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 142: 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 143: 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 144: 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 145: 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 146: 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 147: 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 148: 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 149: 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 / B Pictures 4MV MBMODE Tables

Table 150: Interlace Frame 4MV 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 151: Interlace Frame 4MV 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 152: Interlace Frame 4MV 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 153: Interlace Frame 4MV 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 4MV MBMODE Tables

Table 154: Interlace Frame Non 4MV 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 155: Interlace Frame Non 4MV 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 156: Interlace Frame Non 4MV 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 157: Interlace Frame Non 4MV 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 158: **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-Picture CBPCY Tables

Table 159: **P-Picture CBPCY VLC Table 0**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	6	13
1	1	6	33	7	13
2	1	5	34	54	7
3	4	6	35	103	8
4	5	6	36	8	13
5	1	7	37	9	13
6	12	7	38	10	13
7	4	5	39	110	8
8	13	7	40	11	13
9	14	7	41	12	13
10	10	6	42	111	8
11	11	6	43	56	7
12	12	6	44	114	8

13	7	5	45	58	7
14	13	6	46	115	8
15	2	3	47	5	3
16	15	7	48	13	13
17	1	8	49	7	12
18	96	8	50	8	12
19	1	13	51	9	12
20	49	7	52	10	12
21	97	8	53	11	12
22	2	13	54	12	12
23	100	8	55	30	6
24	3	13	56	13	12
25	4	13	57	14	12
26	5	13	58	15	12
27	101	8	59	118	8
28	102	8	60	119	8
29	52	7	61	62	7
30	53	7	62	63	7
31	4	3	63	3	2

Table 160: **P-Picture CBPCY VLC Table 1**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	14	32	9	13
1	1	3	33	240	8
2	2	3	34	10	13
3	1	5	35	11	13
4	3	3	36	121	7
5	1	4	37	122	7
6	16	5	38	12	13
7	17	5	39	13	13
8	5	3	40	14	13
9	18	5	41	15	13
10	12	4	42	241	8
11	19	5	43	246	8
12	13	4	44	16	13

13	1	6	45	17	13
14	28	5	46	124	7
15	58	6	47	63	6
16	1	8	48	18	13
17	1	14	49	19	13
18	1	13	50	20	13
19	2	8	51	21	13
20	3	8	52	22	13
21	2	13	53	23	13
22	3	13	54	24	13
23	236	8	55	25	13
24	237	8	56	26	13
25	4	13	57	27	13
26	5	13	58	28	13
27	238	8	59	29	13
28	6	13	60	30	13
29	7	13	61	31	13
30	239	8	62	247	8
31	8	13	63	125	7

Table 161: **P-Picture CBPCY VLC Table 2**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	13	32	201	8
1	1	5	33	102	7
2	2	5	34	412	9
3	3	5	35	413	9
4	2	4	36	414	9
5	3	4	37	54	6
6	1	6	38	220	8
7	4	4	39	111	7
8	5	4	40	221	8
9	24	6	41	3	13
10	7	4	42	224	8
11	13	5	43	113	7
12	16	5	44	225	8

13	17	5	45	114	7
14	9	4	46	230	8
15	5	3	47	29	5
16	25	6	48	231	8
17	1	8	49	415	9
18	1	10	50	240	8
19	1	9	51	4	13
20	2	8	52	241	8
21	3	8	53	484	9
22	96	7	54	5	13
23	194	8	55	243	8
24	1	13	56	3	12
25	2	13	57	244	8
26	98	7	58	245	8
27	99	7	59	485	9
28	195	8	60	492	9
29	200	8	61	493	9
30	101	7	62	247	8
31	26	5	63	31	5

Table 162: **P-Picture CBPCY VLC Table 3**

CBPCY	VLC Codeword	VLC Size	CBPCY	VLC Codeword	VLC Size
0	0	9	32	28	9
1	1	2	33	29	9
2	1	3	34	30	9
3	1	9	35	31	9
4	2	2	36	32	9
5	2	9	37	33	9
6	3	9	38	34	9
7	4	9	39	35	9
8	3	2	40	36	9
9	5	9	41	37	9
10	6	9	42	38	9
11	7	9	43	39	9
12	8	9	44	40	9

13	9	9	45	41	9
14	10	9	46	42	9
15	11	9	47	43	9
16	12	9	48	44	9
17	13	9	49	45	9
18	14	9	50	46	9
19	15	9	51	47	9
20	16	9	52	48	9
21	17	9	53	49	9
22	18	9	54	50	9
23	19	9	55	51	9
24	20	9	56	52	9
25	21	9	57	53	9
26	22	9	58	54	9
27	23	9	59	55	9
28	24	9	60	28	8
29	25	9	61	29	8
30	26	9	62	30	8
31	27	9	63	31	8

11.7 DC Differential Tables

11.7.1 Low-motion Tables

Table 163: **Low-motion Luminance 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 164: **Low-motion Chroma 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 165: **High-motion Luminance 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 166: **High-motion Chroma 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 167: 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 168: **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 169: **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 170: **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 171: **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 172: **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 173: **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 174: **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 175: **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 176: **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 177: **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 178: **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 179: **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 180: **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 181: **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 182: **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 183: **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 184: **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 185: **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 186: **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 187: **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 188: **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 189: **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 190: **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 191: **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 192: **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 193: **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 194: **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 195: **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 196: **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 197: **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 198: **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 199: **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 200: **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 201: **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 202: **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 203: **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 204: **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 205: **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 206: **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 207: **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 208: **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 209: **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 210: **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 211: **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 212: **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 213: **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 214: **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 215: **High Rate Intra Delta Run Indexed by Level Table (Last = 1)**

Level	Delta Run
1	16
2	14
3	2
4	0

11.8.7 High Rate Inter Tables

Table 216: **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 217: **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 218: **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 219: **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 220: **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 221: **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 222: **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 223: Intra Normal Scan

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

Table 224: **Intra Horizontal Scan**

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

Table 225: **Intra Vertical Scan**

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

11.9.2 Inter zigzag tables

Table 226: **Inter 8x8 Scan for Simple and Main Profiles and Progressive Mode in Advanced Profile**

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

Table 227: **Inter 8x4 Scan for Simple and Main Profiles**

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

Table 228: **Inter 4x8 Scan for Simple and Main Profiles**

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

Table 229: Inter 4x4 Scan for Simple and Main Profiles and Progressive Mode in Advanced Profile

0	3	7	11
1	4	8	12
2	6	9	14
5	10	13	15

Table 230: **Progressive Mode Inter 8x4 Scan for Advanced Profile**

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

Table 231: **Progressive Mode Inter 4x8 Scan for Advanced Profile**

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

Table 232: **Interlace Mode Inter 8x8 Scan for Advanced Profile**

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

Table 233: **Interlace Mode Inter 8x4 Scan for Advanced Profile**

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

Table 234: **Interlace Mode Inter 4x8 Scan for Advanced Profile**

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

Table 235: Interlace Mode Inter 4x4 Scan for Advanced Profile

0	4	7	11
1	5	9	13
2	6	10	14
3	8	12	15

11.10 Motion Vector Differential Tables

Table 236: **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 237: 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 238: **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 239: **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			

Annex A
Transform Specification

A.1 Inverse Transform

(This section forms an integral part of this recommendation, and is normative)

The formulas in Figure 108 through Figure 111 defines the Inverse Transform required for conformance. An exact match to the values produced as specified in this annex is required. Figure 108 defines the implentation for the 8x8 Inverse Transform. Figure 109 defines the implentation for the 4x8 Inverse Transform. Figure 110 defines the implentation for the 8x4 Inverse Transform. Figure 111 defines the implentation for the 4x4 Inverse Transform.

The size of the input and output samples is representable in 16 bits, although the input requires only 12 bits and the output requires only 10 bits. 16 bit modulo arithmetic is necessary and sufficient when calculating sums and differences. When multiplying two numbers, a 16 bit signed representation of the product is necessary and sufficient.

The transform matrices for a 1D 8 point inverse transformation and a 1D 4 point inverse transformation are presented in Figure 104 and Figure 105.

$$T_8 = \begin{bmatrix} 12 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 16 & 15 & 9 & 4 & -4 & -9 & -15 & -16 \\ 16 & 6 & -6 & -16 & -16 & -6 & 6 & 16 \\ 15 & -4 & -16 & -9 & 9 & 16 & 4 & -15 \\ 12 & -12 & -12 & 12 & 12 & -12 & -12 & 12 \\ 9 & -16 & 4 & 15 & -15 & -4 & 16 & -9 \\ 6 & -16 & 16 & -6 & -6 & 16 & -16 & 6 \\ 4 & -9 & 15 & -16 & 16 & -15 & 9 & -4 \end{bmatrix}$$

Figure 104: 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 105: Matrix for 1-D 4-point Inverse Transform

These matrices are split into even and odd components, with the even component divided by 2. The even components are of relevance to the definition of the normative inverse transform, and are shown in Figure 106 and Figure 107.

$$T_8^e = \begin{bmatrix} 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 8 & 7 & 4 & 2 & -2 & -4 & -7 & -8 \\ 8 & 3 & -3 & -8 & -8 & -3 & 3 & 8 \\ 7 & -2 & -8 & -5 & 5 & 8 & 2 & -7 \\ 6 & -6 & -6 & 6 & 6 & -6 & -6 & 6 \\ 4 & -8 & 2 & 7 & -7 & -2 & 8 & -4 \\ 3 & -8 & 8 & -3 & -3 & 8 & -8 & 3 \\ 2 & -5 & 7 & -8 & 8 & -7 & 5 & -2 \end{bmatrix}$$

Figure 106: Even component of 8-point Inverse Transform

$$T_4^e = \begin{bmatrix} 8 & 8 & 8 & 8 \\ 11 & 5 & -5 & -11 \\ 8 & -8 & -8 & 8 \\ 5 & -11 & 11 & -5 \end{bmatrix}$$

Figure 107: Even component of 4-point Inverse Transform

In the equations shown in Figure 108 through Figure 111, the dequantized transform coefficients forming the input to the inverse transform are represented as the matrix *D*. Matrix *R* represents the reconstructed output. D_1 is the intermediate result after row-wise transformation, which is always the first step. Bitshifts defined on a matrix are carried out componentwise on the matrix entries, in signed integer arithmetic. The prime operator applied to a matrix denotes its transpose.

$$D_1 = (D \cdot T_8 + 4) >> 3$$

Error! Objects cannot be created from editing field codes.

Figure 108: 8x8 Inverse Transform

$$D_1 = (D \cdot T_4 + 4) >> 3$$

Error! Objects cannot be created from editing field codes.

Figure 109: 4x8 Inverse Transform

$$D_1 = (D \cdot T_8 + 4) >> 3$$

$$[D_{1a} \quad D_{1b}] = D_1' \cdot \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{bmatrix}$$

$$D'_{2a} = D_{1a} >> 1$$

$$D'_{2b} = D_{1b} >> 1$$

$$R = \left(T_4'^{e} \cdot D_1 + \begin{bmatrix} D_{2a} \\ D_{2b} \\ D_{2b} \\ D_{2a} \end{bmatrix} + 32 \right) >> 6$$

Figure 110: 8x4 Inverse Transform

$$D_1 = (D \cdot T_4 + 4) >> 3$$

$$[D_{1a} \quad D_{1b}] = D_1' \cdot \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{bmatrix}$$

$$D'_{2a} = D_{1a} >> 1$$

$$D'_{2b} = D_{1b} >> 1$$

$$R = \left(T_4'^{e} \cdot D_1 + \begin{bmatrix} D_{2a} \\ D_{2b} \\ D_{2b} \\ D_{2a} \end{bmatrix} + 32 \right) >> 6$$

Figure 111: 4x4 Inverse Transform

A.2 Forward Transform

(This section is informative)

There is no requirement of bit-exactness on the forward transform and hence this section is only informative. The forward transform may be implemented in scaled integer arithmetic or using floating point or other means. The matrix-multiplication representation of the forward transform shown below is purely an analytical representation unlike for the inverse transform where it the matrix multiplies specifically referred to integer multiplications with 16 bit registers. Rounding between stages may be done as necessary and this choice is left to the encoder.

The 4x4, 4x8, 8x4 and 8x8 transforms of the data matrix *D* may be calculated using the following set of equations for these four cases:

$$\hat{D} = (T_4\, DT_4') \circ N_{44}$$
$$\hat{D} = (T_8\, DT_4') \circ N_{48}$$
$$\hat{D} = (T_4\, DT_8') \circ N_{84}$$
$$\hat{D} = (T_8\, DT_8') \circ N_{88}$$

where the operator $\circ$ is a componentwise multiplication. The normalization matrices N_{ij} are given by

$$N_{ij} = c_j\, c_i'$$

where 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

(This annex forms an integral part of this recommendation and is normative)

The following section describes the upsampling and downsampling process used in codec implementation. On the encoder side, downsampling is applied to the input frame if the current resolution is smaller than the original resolution as described in section 8.1.1.3. On the decoder side, upsampling is applied to the decoded frame if the current resolution is smaller than the original resolution. Since both of these operations occur outside the reconstruction loop, the implementer is free to use any method to upsample or downsample the frames. However, attention should be paid to the relative spatial positioning of the samples produced from the upsampling and downsampling processes. In particular, the video samples of the downsampled frame should have the following spatial alignment with respect to the video samples of the frame at the original resolution.

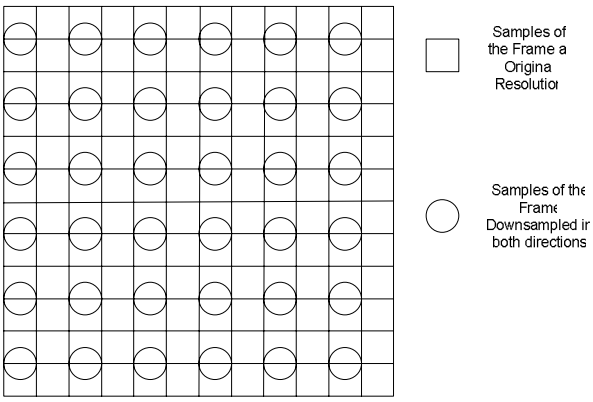


Figure 112: Relative Spatial Alignment of the video samples of the Downsampled Frame,

and video samples of the Original Frame.

The following definitions are used for the downsampling/upsampling pseudocode examples:

N_u = number of samples in upsampled (full resolution) line
N_d = number of samples in a downsampled (half resolution) line

The term 'line' refers to all the samples in a horizontal row or vertical column in a Y, Cb or Cr component plane. Upsampling or downsampling 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 upsampling or downsampling is performed, the horizontal lines are resampled first, followed by the vertical lines.

For luminance lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution luminance line)
if ((N_d & 15) != 0)
    $N_d = N_d + 16 – (N_d \& 15)$

For chroma lines:

$N_d = N_u / 2$ (where N_u is the number of samples in a full resolution chroma line)

if ((N_d & 7) != 0)
    $N_d = N_d + 8 – (N_d$ & 7)

Annex C
Hypothetical reference decoder

(This annex forms an integral part of this recommendation and is normative)

Coded video bit streams 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 C.1.

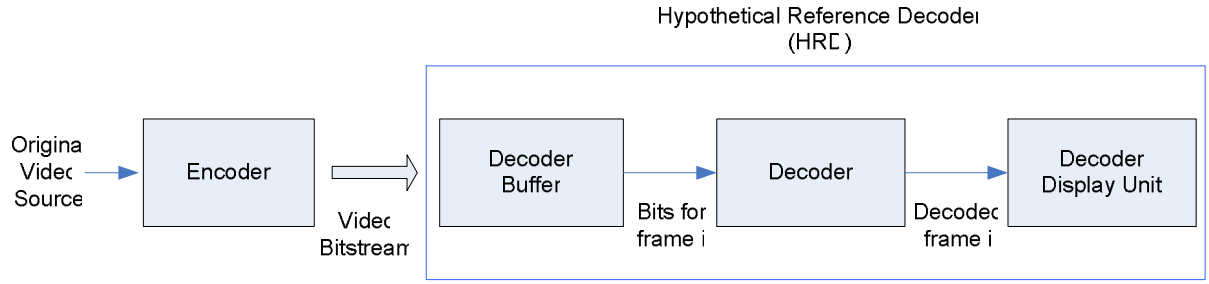


Figure C.1. Components of an HRD: decoder buffer, decoder, and display unit

The HRD does not mandate buffering, decoding, or display mechanisms for decoder implementations. Its main goal is to limit the encoder's bit rate fluctuations according to a basic buffering model, so that the resources necessary to decode the bit stream are predictable.

The HRD may 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 real-values, so that no rounding errors may propagate.

C.1 Leaky Bucket Model

C.1.1 This subclause is informative and defines a leaky bucket model.

The buffering model that governs the operation of the HRD is known as a leaky bucket and is described in this section. A leaky bucket is 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 bit stream is received at bit rate smaller than or equal to the peak transmission rate *R*, and it is stored into a decoder buffer of size B until the buffer fullness reaches F bits. Then, the decoder instantaneously removes the bits for the first video frame of the sequence from the buffer, and instantaneously decodes that frame. The bits for the following frames are also removed and decoded instantaneously at subsequent time intervals.

Figure C.2 illustrates the decoder buffer fullness as a function of time for a bit stream that is contained in a leaky bucket of parameters (*R*, *B*, *F*). The decoder buffer fullness β_i after removing frame *i*, with $i > 1$, may be expressed as follows:

$$\beta_1 = F - b_1$$

$$\beta_i = min(B, \beta_{i-1} + R_i (t_i - t_{i-1}) - b_i), \qquad \text{(C.1)}$$

where t_i is the decoding time for frame *i*, and b_i is the number of bits for frame *i*. 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 is such that $R_i <= R$ for all *i*. In Figure C.2, 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. To be more concrete, the buffer fullness at any time instant shall be less than or equal to B. As a result, in equation (C.1), observe that 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 C.3.

When the decoder buffer is full, it is assumed that the encoder will not send any more bits until there is room in the buffer. This phenomenon occurs frequently in practice. For example, a DVD includes a video coded bit stream 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.

[2] A leaky bucket may also be specified by parameters (*R*, *B*, F^e), where F^e is the initial encoder buffer fullness. Here, we have chosen to use the initial decoder buffer fullness *F*.

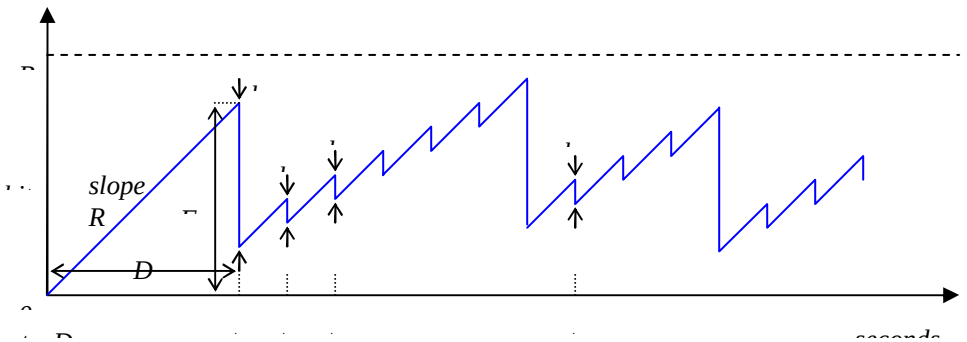


Figure C.2. The plot illustrates an example of decoder buffer fullness when decoding a generic video bit stream that is contained in a leaky bucket of parameters (*R*, *B*, *F*). *R* is the peak incoming (or channel) bit rate in bits/sec, and in this case the transmission rate is constant and equal to 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 *i*th 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.

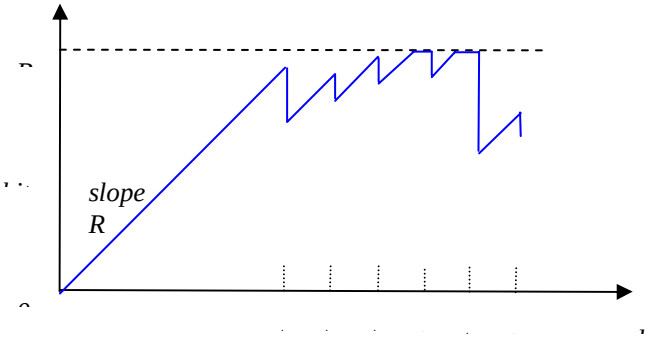


Figure C.3. Plot of decoder buffer fullness, where the fullness reaches the maximum buffer size *B* during some time segments. In this example, such segments are a subset of the intervals (t_2, t_3) and (t_3, t_4). When the decoder buffer is full, the encoder does not send any bits.

Decoder buffer underflow occurs usually if an encoder produces relatively large frames. The decoder buffer fullness may then be reduced to the point that the bits for the next frame are not available at the decoding time.

A leaky bucket with parameters (*R*, *B*, *F*) is said to *contain* a coded video bit stream if there is no underflow of the decoder buffer (i.e., $\beta_i >= 0$, for all *i*). To be more concrete, a leaky bucket with parameters (*R*, *B*, *F*) contains a coded video bit stream if the following constraints hold:

$$\beta_1 = F - b_1$$

$$\beta_i = min(B, \beta_{i-1} + R_i (t_i - t_{i-1}) - b_i), \qquad i > 1$$

$$R_i <= R \qquad \text{all } i$$

$\beta_i >= 0$ all i (C.2)

C.1.2 This subclause defines a requirement on all video bit streams when the HRD operates in constant-delay mode.

A compliant video bit stream shall meet the restrictions imposed by equation (C.2), so that at least one leaky bucket (R, B, F) contains the bit stream. The leaky bucket values (R, B, F) shall be signaled in the bit stream, so that the rate and buffer size resources necessary to decode this bit stream are predictable.

C.1.3 This subclause is informative only. It describes CBR and VBR bit streams.

A video bit stream that meets the constraints of the equations in (C.2) may be denoted a variable bit rate or VBR bit stream, e.g., see [MPEG2].

If the constraints in equation (C.2) apply to a bit stream 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 there is no buffer overflow (i.e., $\beta_i + b_i <= B$, for all i), the bit stream may be denoted a constant bit rate or CBR bit stream.

Since CBR bit streams are a special case of VBR bit streams, this recommendation does not make a distinction between them.

C.2 Multiple Leaky Buckets

This clause is informative only and explains the concept of multiple leaky buckets.

A given video stream may be contained in many leaky buckets. For example, if a video stream is contained in a leaky bucket with parameters (R, B, F), it will 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 may 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 may be as large as the bit stream itself. In short, a video bit stream may 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 delay are large enough.

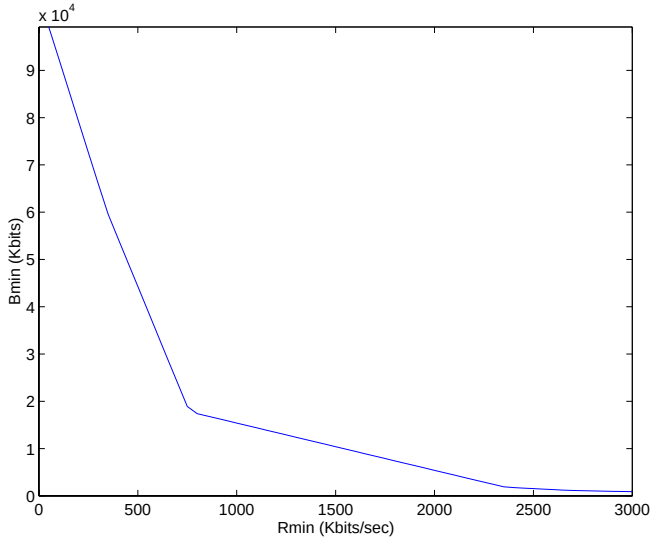


Figure C.4. Illustration of peak bit rate R_{min} and buffer size B_{min} values for a given video bit stream. This curve indicates that in order to transmit the stream at a peak bit rate R, the decoder needs to buffer at least B_{min}(R) bits. Observe that higher peak rates require smaller buffer sizes. Alternatively, if the size of the decoder buffer is B, the minimum peak rate required for transmitting the bit stream is the associated $R_{min}(B)$.

Further, as proven in [HRD], for any value of the peak transmission bit rate R, and assuming $R_i = R$ for all i in equation (C.2), one may find the minimum buffer size B_{min} and the minimum initial buffer fullness F_{min} that will contain the video bit stream. These minimum values may be computed using a simple search using the constraints in (C.2), as demonstrated in [HRD]. By computing B_{min} for each R, we may plot a curve of optimum R-B values such as the one in Figure C.4.

C.3 Bit Stream Syntax for the Hypothetical Reference Decoder

C.3.1 This subclause only applies when the HRD operates in constant-delay mode. It describes syntax required in a video bit stream that is compliant to the Advanced profile, when operating in such mode.

The encoder shall signal N leaky bucket models, each of which shall contain the video bit stream, as defined in (C.2). The desired value of N may be selected by the encoder (where N > 0).

The parameter values of these leaky buckets may be expressed as follows:

$$(R_1, B_1, F_1), (R_2, B_2, F_2), \ldots, (R_N, B_N, F_N), \qquad \text{(C.3)}$$

The HRD syntax element values are to be communicated to the decoder by the Transport Layer for video bit streams compliant to the Simple and Main profiles.

The HRD syntax element values shall be inserted at the sequence header for video bit streams compliant to the Advanced profile, except when the encoder is operating in variable-delay mode. The sequence header for the Advanced profile is described in Section 6.1.

Observe that 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 of the video bit stream, and hence the leaky bucket values may be modified throughout the video bit stream. Also, an encoder may want to use fewer leaky buckets later in the bit stream to avoid syntax overhead.

The HRD syntax elements are inserted in the video bit stream headers as follows:

hrd_parameters( )	**Descriptor**	**Range**
{		
hrd_num_leaky_buckets	FLC-5	(1, 32)
bit_rate_exponent	FLC-4	(6,21)
buffer_size_exponent	FLC-4	(4,19)
for(n=1; n <= hrd_num_leaky_buckets; n++)		
{		
hrd_rate[n **]**	FLC-16	$(1,2^{16})$
hrd_buffer[n **]**	FLC-16	$(1,2^{16})$
hrd_fullness[n **]**	FLC - 8	(0, 255)
}		

hrd_num_leaky_buckets – A number between 1 and 32 that specifies the number of leaky buckets N. The value of N-1 is encoded as a fixed length code in binary using 5 bits.

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 is 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 is encoded in the syntax element **bit_rate_exponent** in fixed length using 4 bits , and takes the range from 6 to 21.

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 is 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 is encoded in the syntax element **buffer_size_exponent** using a fixed length of 4 bits, and takes the range from 4 to 19.

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

hrd_fullness[n**]** – This syntax element defines the decoder buffer fullness as an upwards rounded fraction of the buffer size B_n, in units of $B_n/256$. This element may take values in the range 1 to 256 and is encoded in binary using the 8 bit values 0 through 255 to uniformly cover the range. Its value is computed as follows:

$$\textbf{hrd\_fullness[}n\textbf{]} = \left\lceil 256 \times \frac{\min(B_n, \beta_{i,n} + b_i)}{B_n} \right\rceil - 1 \qquad \text{(C.4)}$$

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. Here we use 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 $F_n = (\beta_{1,n} + b_1)$.

C.3.2 This subclause is informative only.

In practice, an encoder may do the following:

(a) Pre-select the leaky bucket values in (C.3) and encode the bit stream with a rate control that makes sure that all of the leaky bucket constraints are met.
(b) Encode the bit stream and then use the equations in (C.2) to compute a set of leaky buckets containing the bit stream 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 bit stream is encoded.

Approach (a) may be applied to live or on-demand transmission applications, while (b) and (c) only apply to on-demand. Observe that the *N* leaky bucket approach used in the HRD of this recommendation is also used in the H.264 standard (c.f., [HRD]). If *N*=1, the hypothetical reference decoder is a subset of MPEG-2's Video Buffering Verifier [MPEG2].

C.4 Interpolating Leaky Buckets

This clause is informative for the HRD, although equations (C.5) and (C.6) are normative for time-conformant decoders.

Another key observation proven in [HRD] is that the curve of (R_{min}, B_{min}) pairs, or that of (R_{min}, F_{min}) for any bit stream (such as the one in Figure C.4) is piecewise linear and convex. Because of the convexity, if *N* points of the

curve are provided, the decoder may 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 C.5 illustrates a set of N leaky bucket models and their interpolated or extrapolated (R, B) values.

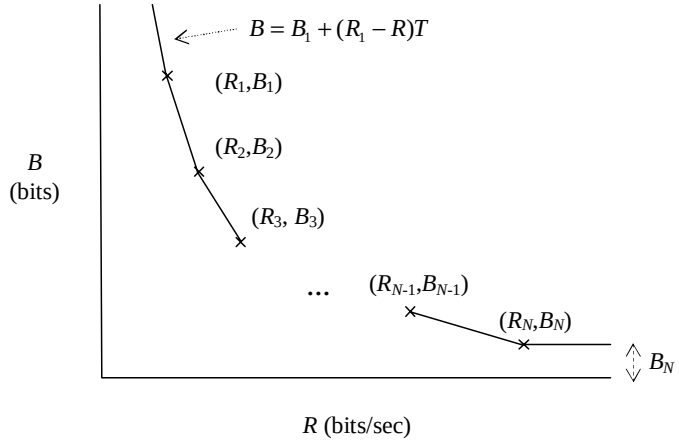


Figure C.5. Example of (R, B) values available for the generalized hypothetical reference decoder (GHRD), all of which are guaranteed to contain the bit stream. T is the time length or duration of the encoded video sequence.

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 may 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 bit stream, because, as proven in [HRD], 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 bit stream, 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 C.5.

Using equations (C.5) and (C.6), when the transmission peak rate of a given encoding/decoding system is known, the decoder may determine a nearly minimum leaky bucket buffer size and delay. Alternatively, knowing the physical buffer size, a smart decoder may 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) may be linearly interpolated or extrapolated to be able to determine nearly optimum leaky buckets.

C.5 Display issues

This clause is informative only.

The leaky bucket model does not address when a video frame is displayed in the HRD display unit. Any compliant decoder, including this HRD, should 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 should also be re-ordered properly before display. If 3:2 pull-up occurs after decoding, the correct fields should be repeated to produce an accurate 3:2 telecine pattern on the display. Nevertheless, constraints on display times (e.g., according to the decoding times t_1, t_2, etc.) are beyond the scope of this draft and belong to the system layer. The objective of the HRD in this recommendation is only to impose some basic buffering constraints on the video bit stream, rather than mandate any decoding, buffering, or display mechanisms to implementers.

C.6 Time-Conformant Decoders

This clause is normative for decoder implementations that wish to be time-conformant. This is the only clause in the HRD that refers to and constraints practical decoder implementations.

Time-conformant decoders are of interest for broadcast or other applications with a fixed end-to-end delay, where a practical decoder needs to decode the bit streams without suffering from buffer underflow. If a practical decoder wishes to be time-conformant, the HRD parameters provide some helpful constraints.

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 (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, shall confirm that the physical buffer size in the decoder is 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 clause 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 parameter are not signaled in the sequence header. In this mode:

- The syntax element **hrd_num_leaky_buckets** may only take values 0 or 1.

  If equal to 0, the leaky bucket used is (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 bit stream, as defined in Annex D. The initial buffer fullness equals the buffer size, i.e., F_1=B_1.

  If equal to 1, the leaky bucket used is (R_1, B_1, F_1) and is signaled in the bit stream according to the syntax defined in subclause **C.3.1**.

- The initial buffer fullness is F_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 is examined every T seconds, where T is the inverse of the maximum frame rate in the example column of Table D.2, for the respective profile and level of the given bit stream. If at least one complete coded picture is in the buffer, then all the data for the earliest picture in the bit stream order is instantaneously removed. Immediately before removing the picture, the buffer fullness shall be less than B_1.

In this mode, the HRD waits 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 MPEG-2 [MPEG2], or the default HRD operating mode in H.263 [H263].

C.8 Benefits of multiple leaky buckets

This clause is informative only and describes the benefits of indicating multiple leaky buckets that contain a video bit stream.

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 modem, ISDN, DSL, cable, etc.) and also fluctuates in time according to network conditions (e.g., congestion, the number of

users connected, etc.) In addition, the video bit streams 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 specification is more flexible than prior HRDs and enables a system to decode a bit stream 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 will select the smallest buffer size and delay (according to the available leaky bucket data) that will be able to decode the bit stream without suffering from buffer underflow. Conversely, for a given buffer size, the hypothetical decoder will select and operate at the minimum required peak transmission bit rate.

There are multiple benefits of this generalized hypothetical reference decoder. For example, a content provider may create a bit stream once, and a server may 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 may negotiate the best leaky bucket for the given networking conditions, e.g., the one that will produce the lowest start-up (buffer) delay, or the one that will require the lowest peak transmission bit rate for the given buffer size of the device. The multiple leaky bucket approach has been shown to provide large savings in peak rate, buffer size, delay and even quality in encoding/decoding systems [HRD].

Annex D
Profile and Levels

(This annex forms an integral part of this recommendation and is normative)

Profiles and levels define subsets of the syntax and semantics of this recommendation.

A profile defines constraints on the algorithms or compression features used to create a bit stream, and a level defines additional restrictions on a given profile. The profiles in VC-9 are not hierarchical. For example, main profile is not a subset of advanced profile, nor is it a subset. However, levels are hierarchical. Within the same profile, higher levels generally imply higher requirements in processing speed and memory within a profile.

Encoders shall produce bit streams compliant to a given profile and level, and decoders shall decode bit streams compliant to a given profile and level. Note that a bitstream compliant to a particular profile/level combination is also compliant to all higher levels at the same profile. Therefore, profiles and levels are critical to ensure interoperability between encoders, coded bit streams, and decoders.

D.1 Overview

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 of conformance in this profile.
- The Main profile targets high-rate internet applications such as streaming, movie delivery via IP, HD DVD for PC playback, or TV/VOD over IP. This profile contains three levels of conformance.
- The Advanced profile targets broadcast applications, such as digital TV or HDTV. It is the only profile that supports interlaced content. In addition, this profile contains the required syntax elements to transmit video bit streams compliant to this recommendation into generic systems, such as MPEG-2 Transport or Program Streams [MPEG2]. This profile contains six levels.

Table D.1 lists all the profiles and levels, and the label associated to each of them.

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 D.1: List of profiles and levels in this recommendation.

D.2 Profiles

Table D.2 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". Otherwise, such feature is used in all profiles.

Compression Feature	Section in spec	Simple profile	Main Profile	Advanced Profile
Loop filter	8.6		X	X
Dynamic resolution change	8.1.1.3, 8.3.4.2		X	
Adaptive macroblock quantization	7.1.1.30,7.1.3.6, 7.1.3.7		X	X
Bidirectional (B) Frames	8.4,10.4,10.8		X	X
Intensity compensation	8.3.8,10.3.7		X	X
Range Reduction	7.1.1.8,8.1.1.4, 8.3.4.12		X	
Range Mapping	6.2			X
Interlace: Field coding mode	10.1, 10.3,10.4			X
Interlace: Frame coding mode	10.5,10.7,10.8			X
Syntax elements for transmission over generic systems layer	6.1			X

Table D.2. 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.

As explained in Annex C, the encoder may define multiple leaky buckets in (C.4) that contain a given video bit stream. The HRD is able to decode a bit stream operating according to any of those leaky bucket parameters, or even according to interpolations or extrapolations of such parameters.

For a bit stream to be compliant to a given profile and level, at least one of the leaky bucket parameters in (C.4) shall 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	Tools (B frames + loop filter	Inter-lace support	Rmax	Bmax	MV [H] x [V]
SP@LL	1,485	99	QCIF, 176x144 15 Hz			96	20	[-64, 63¾] x [-32, 31¾]
SP@ML	5,940	396	QVGA, 320x240, 24 Hz; CIF, 352x288, 15 Hz			384	77	[-64, 63¾] x [-32, 31¾]
MP@LL	7,200	396	CIF, 352x288,30Hz; SIF 352x288, 25Hz	x		2,000	306	[-128, 127¾] x [-64, 63¾]
MP@ML	40,500	1,620	480p, 704x480, 30Hz 576p, 720x576, 25Hz	x		10,000	611	[-512, 511¾] x [-128, 127¾]
MP@HL	245,760	8,192	1080p, 1920x1080, 25 Hz / 30 Hz	x		20,000	2,442	[-1024, 1023¾] x [-256, 255¾]
AP@L0	11,880	396	CIF, 352x288, 25 Hz, 30 Hz	x		2,000	250	[-128, 127¾] x [-64, 63¾]
AP@L1	48,600	1,620	525 SD, 720x480, 30Hz 625 SD, 720x576, 25 Hz	x	x	10,000	1,250	[-512, 511¾] x [-128, 127¾]
AP@L2	108,000	3,600	480p, 704x480, 60Hz 720p, 1280x720, 30Hz	x	x	20,000	2,500	[-512, 511¾] x [-128, 127¾]
AP@L3	245,760	8,192	1080i, 1920x1080, 25Hz/30Hz 1080p, 1920x1080, 25Hz/30Hz 720p, 1280x720, 50Hz/60Hz 2048x1024, 30Hz	x	x	45,000	5,500	[-1024, 1023¾] x [-256, 255¾]
AP@L4	491,520	16,384	1080p, 1920x1080, 50Hz/60Hz 2048x1536, 24 Hz 2048x2048, 30Hz	x	x	135,000	16,500	[-1024, 1023¾] x [-256, 255¾]

Table D.3 Limitations of profiles and levels. 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. In AP@ML, “480p/i, 30 Hz” indicates that both 480p at 30 Hz and 480i at 30 Hz (60 fields/sec) 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[Y] Motion vector range in full pixel units. [H] = horizontal, [V] = vertical.

D.4 Syntax

The Simple, Main and Advanced profiles are signaled to the decoder in the bit stream, by the syntax element **Profile**, which is included in the sequence header as described in Section 6.1.1. The following codes are used to signal profiles.

00	Simple
01	Main
11	Advanced
10	Forbidden

The levels for Simple and Main profile are to be communicated to the decoder by the Transport Layer. The levels for Advanced profile are indicated 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

(This annex forms an integral part of this recommendation and is normative)

An Independently Decodable Unit (IDU) of compressed video data shall begin with an identifier called Start Code (SC). An IDU could refer to a single picture, or a group of macroblocks in a picture (also called slice), or a group of pictures (GOP), or a sequence header.

This recommendation mandates a sequence of four bytes as the start code, which consists of an unique three-byte Start Code Prefix (SCP), and an 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 IDU 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 are always byte-aligned.

A non-normative Encapsulation Mechanism (EM) is described to prevent emulation of the start code prefix in the bitstream. The compressed data before encapsulation is called Raw Independently Decodable Unit (RIDU), while Encapsulated IDU (EIDU) refers to the data after encapsulation.

Section E.1 provides an encoder-side perspective on how start code and encapsulation operates, and is informative. Section E.2 specifies detection of start codes and EIDUs at the decoder, and is normative. Section E.3 deals with extraction of an RIDU from an EIDU, and is also normative. Section E.4 specifies start code suffixes for various IDU types, and is also normative.

E.1 Start-codes and Encapsulation – An encoder viewpoint (Informative)

The encapsulation of a RIDU to obtain an EIDU is described below.

Step 1: A trailing ‘1’ bit is added to the end of the RIDU. The EM now stuffs between 0 and 7 bits onto the end of the IDU such that the IDU ends in a byte-aligned location. The value of these stuffing bits is ‘0’. As a result, at the end of this step, the IDU is represented in an integer number of bytes, in which the last byte of the IDU cannot be a zero-valued byte. The resulting string of bytes is called the payload bytes of the IDU.

Step 2: The three-byte start code prefix (0x000001), and the appropriate start code suffix that identifies the IDU type, are placed at the beginning of the EIDU.

Step 3: The remainder of the EIDU is formed by processing the payload bytes of the IDU through the following emulation prevention process. The emulation of start code prefixes in the IDU is eliminated via byte-stuffing. The emulation prevention process is equivalent to the following operation:

1) Replace each string within the payload of 2 consecutive bytes of value 0x00 followed by a byte that contains zero values in its six MSBs (regardless of the LSB values) with 2 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 E-1.

Table E-1: 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 3: The three-byte start code prefix (0x000001), and the appropriate start code suffix that identifies the IDU type, are attached to the beginning of the IDU. The resulting payload is an encapsulated IDU.

The encoder is also allowed to insert any number of zero-valued stuffing bytes after the end of an EIDU. Equivalently, any number of zero-valued stuffing bytes may be inserted before a start code prefix. The start code is structured such that it may be detected by a decoder even in the presence of these zero-valued stuffing bytes. In some transmission environments such as H.320, the encoder may use this feature to insert extra zero-valued stuffing bytes as desired, which may enable the decoder to quickly recover the location of the start- codes even if it has lost track of the intended alignment of the bitstream to byte boundaries. Further, these zero-valued stuffing bytes may also be useful in splicing bitstreams, filling a constant bit-rate channel, etc. **Zero-Valued Stuffing bytes prior to start codes, or at the end of an EIDU, are not processed through the encapsulation mechanism – only RIDU data requires such processing.**

E.2 Detection of Start codes and EIDU (Normative)

The detection of an EIDU starts with the search for the start code prefix.

E.2.1 Detection of Start Codes Starting from Byte-Aligned Positions (Normative)

In a decoder that cannot lose byte-alignment, or once byte alignment has been established, start code detection is conducted as follows.

1. Whenever a string of two or more bytes of value 0x00 followed by a byte of value 0x01 is found, a start code prefix detection is declared.

When 2 successive start-codes prefixes are detected, the payload bitstream between them is declared as a new EIDU.

E.2.2 Detection of Start Codes After Loss of Byte Alignment in a Decoder (Informative)

In a decoder that has lost byte-alignment (as may happen in some transmission environments), start-code prefix detection and byte-alignment detection are conducted as follows:

Whenever a string of three or more 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 be recovered such that the first non-zero bit in the non-zero byte shall be the last bit of a byte-aligned start code.

E.3 Extraction of RIDU from EIDU (Normative)

The extraction of a raw IDU from an encapsulated IDU is described below.

Step 1: The start-code suffix is used to identify the type of IDU.

Step 2: The first step is to remove the zero-valued stuffing bytes at the end of EIDU. After this step, the last byte of the IDU shall have a non-zero value.

Step 3: The bytes used for emulation prevention are detected and removed. The process is as follows:

Whenever a string of two bytes of value 0x00 is 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 E-2.

Table E-2: 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

The following byte patterns, if seen within the bitstream, represent error conditions (noting that loss of proper byte alignment by the decoder is considered an error condition):

a) A string of two bytes of value 0x00 followed by a byte equal to 0x02 indicates error condition.
b) A string of three or more bytes of value 0x00, if not followed by a byte of 0x01, is an error condition (note that if two or more bytes equal to zero are followed by a byte of value 0x01 and byte alignment has not been lost, detection of a subsequent start code is declared).
c) A string of two bytes of value 0x00, followed by a byte of value 0x03, followed by a byte which is not one of 0x00, 0x01, or 0x02, or 0x03.

Step 4: In the last byte of the IDU, the last non-zero bit is identified, and that non-zero bit, and all the zero bits that follow, are discarded. The result is a raw IDU.

E.4 Start-code Suffixes for IDU Types (Normative)

The start code suffixes for various IDU types are presented in Table E-3.

Table E-3 Start Code Suffixes for Various IDU Types

Start-code Suffix	IDU Type
0x0F	Sequence Header
0x0E	Entry-point Header
0x0D	Frame
0x0C	Field
0x0B	Slice
0x1F	Sequence Level User Data
0x1E	Entry-point Level User Data
0x1D	Frame Level User Data
0x1C	Field Level User Data
0x1B	Slice Level User Data
0x0A	End-of-Sequence

0x00	SMPTE Reserved
0x01-0x09	SMPTE Reserved
0x10-0x1A	SMPTE Reserved
0x20-0x7F	SMPTE Reserved
0x80-0xFF	Forbidden

SequenceHeader IDU type is sent to identify IDUs which carry sequence header. See Section 6.1 for more details on sequence headers.

Entry-point Header IDU type is sent to identify IDUs which carry entry-point header. See Section 6.2 for more details on entry-point header.

Picture IDU type is sent to identify IDUs which contain the picture header, and the picture data.

Field IDU type is sent to identify IDUs which contain the second field of a picture that is coded as two separate fields.

Slice IDU type is sent to identify IDUs 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 IDU types are used to transmit any user defined data asscociated with the Sequence, Entry-point, Frame, Field, and Slice respectively. See Annex F for more details.

"End-of-sequence" is an optional IDU 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" is not mandatory, and the end of a sequence may be inferred from the header of the next sequence.

Annex F
User data

(This annex forms an integral part of this recommendation and is normative)

A VC-9 bitstream may convey user data as independent data units. The syntax of the user data IDUs is specified in this annex. The User Data IDUs will be identified by the corresponding start codes. There are 5 values for user data start codes, which identify if the user-data belongs to the sequence, entry-point, frame, field or slice. These user-data start codes are defined in Annex E.

The User Data syntax elements are inserted in the video bit stream headers as follows:

User_data_parameters( )	**Descriptor**
{	
User_data_identifier	FLC-32
for(n=1; n <= end_of_idu-1; n++)	
{	
User_data[n]	FLC-8
}	
Flushing_byte	**0x80**

User_data_identifier is a fixed-length syntax element that identifies the type of user data. This syntax element is encoded using 32 bits.

User_data is an array of 8-bit fixed length syntax elements that represent the user data.

Flushing_byte is an 8-bit field set to the constant value ‘0x80’.

Closed Captions and auxiliary user data, if any, should be transmitted in User Data IDUs. It is important to emphasize that the user data shall be accounted for in the HRD buffer model. Note that user data is part of the meta-data, and does not affect the decoding of the bitstream. Modulo HRD effects, the insertion and removal of user-data to and from any compliant bit-stream does not affect the decoding of the bitstream, and b) compliant decoders presented the same bit-stream with, and without any user-data, shall produce the same decoded video output.

Annex G
Bitstream Entry Points and Start-Codes

(This annex forms an integral part of this recommendation and is normative)

The VC-9 syntax allows the insertion of start codes and related headers. There are 11 distinct start code values: 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.

The presence of start codes in a VC-9 bitstream shall obey the rules and guidelines mentioned below.

Conventions

The Figures below reference VC-9 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

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 or a frame 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 picture coding type is set to Progressive or Frame Interlace - or I and P, or P and I, or I and I - if the picture coding type is set to Field Interlace.

A sequence start code and a sequence header may be inserted at regular or irregular intervals in the VC-9 bitstream. Therefore, a VC-9 encoder may adopt various policies to govern the insertion of sequence start codes and associated headers in a VC-9 bitstream.

End—of-Sequence start code

"End-of-sequence" is an optional IDU 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" is not mandatory, and the end of a sequence may be inferred from the header of the next sequence. However, certain applications may benefit from the flexibility of explicitly indicating the end-of-sequence using this start-code. No data follows this start-code. Its value is 0x0000000A.

Entry point start code

An entry point start code (value 0x0000010E) shall always be followed immediately by an entry point header. In a VC-9 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 VC-9 bitstream. Therefore, a VC-9 encoder may adopt various policies to govern the insertion of entry point start codes and associated headers in a VC-9 bitstream. Insertion of any entry point start code and associated header shall always be done to signal a valid entry point in the VC-9 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 VC-9 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 picture coding type is set to Progressive or Frame Interlace - or I and P, or P and I, or I and I - if the picture coding type is set to Field Interlace.

Case of I frame in Progressive mode

Figure G.1 below illustrates how an entry point start code and an entry point header may be present before an I frame when the Picture Coding Type (FCM field) 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 VC-9 bitstream. The entry point applies to the I frame that follows the entry point start code and header but it does not apply to any B frames data or B fields data that follow that I frame in the bitstream and for which the presentation time comes earlier than the presentation time for that I frame.

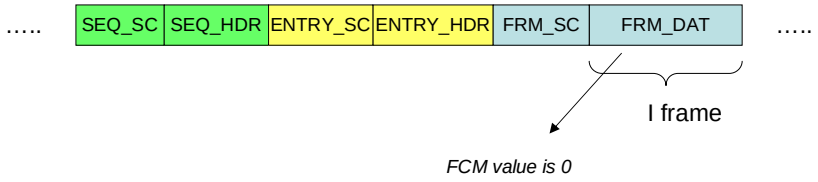


FIGURE G.1
Entry Point Signaled before an I frame (Progressive Picture Coding)

Case of I/P frame in Field Interlace mode

Figure G.2 below illustrates how an entry point start code and header may be present before an I/P frame when the Picture Coding Type (FCM field) is set to '10' (Field Interlace mode).

Since the frame is made 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 VC-9 bitstream:

- The value of the "numref" field in the Field header of the P field of the entry I/P frame shall be '0'
- The value of the "reffield" field 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 not dependency on frames or fields before the entry point.

The entry point applies to the I/P frame that follows the entry point start code and header but it does not apply to any B frames data or B field data that follow that I/P frame in the bitstream and for which the presentation time comes earlier than the presentation time for that I/P frame.

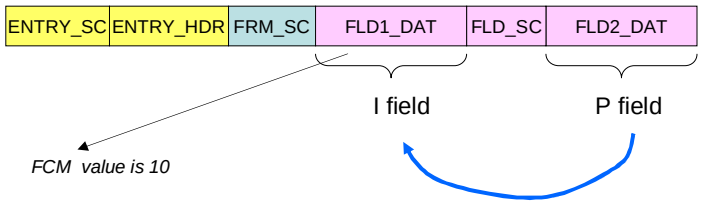


FIGURE G.2
Entry Point Signaled before an I/P frame (Field Interlace Picture Coding)

Case of P/I frame in Field Interlace mode

Figure G.3 below illustrates how an entry point start code and header may be present before an P/I frame when the Picture Coding Type (FCM field) is set to '10' (Field Interlace mode).

Since the frame is made 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 VC-9 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" field in the Field header of the first P field following the entry P/I frame shall be '0'
- The value of the "reffield" field 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 also that 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.

The entry point applies to the I field that follows the entry point start code and header but it does not apply to any B frames data that follow that I field in the bitstream and for which the presentation time is earlier than the presentation time for that I field. Furthermore, the entry point does not apply to the P field data located between the entry point start code and the following I field.

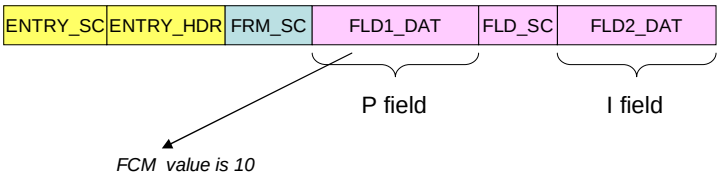


FIGURE G.3
Entry Point Signaled before an P/I frame (Field Interlace Picture Coding)

Case of I/I frame in Field Interlace mode

Figure G.4 below illustrates how an entry point start code and header may be present before an I/I frame when the Picture Coding Type (FCM field) is set to '10' (Field Interlace mode). The Figure does not shows a sequence start code and a sequence header before the entry point start code but it may be the case that such structures 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 VC-9 bitstream.

The entry point applies to the I/I frame that follows the entry point start code and header but it does not apply to any B frames data or B field data that follow that I/I frame in the bitstream and for which the presentation times come earlier than the presentation times for that I/I frame.

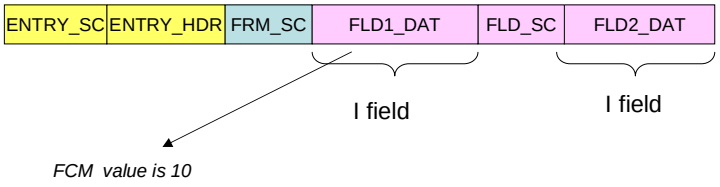


FIGURE G.4
Entry Point Signaled before an I/I frame (Field Interlace Picture Coding)

Case of I frame in Frame Interlace mode

Figure G.5 below illustrates how an entry point start code and header may be present before an I frame when the Picture Coding Type (FCM field) is set to '11' (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 VC-9 bitstream. The entry point applies to the I frame that follows the entry point start code and header but it does not apply to any B frames data or B fields data that follow that I frame in the bitstream and for which the presentation time come earlier than the presentation time for that I frame.

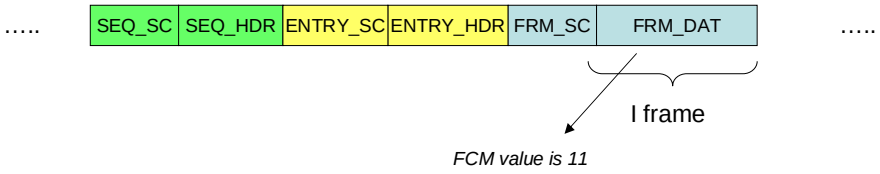


FIGURE G.5
Entry Point Signaled before an I frame (Frame Interlace Coding)

Frame Start Code

A frame start code (value 0x0000010D) shall always be followed immediately by a frame header. In a VC-9 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.

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. The use of field start codes is optional in Field Interlace frames. The use of field start codes is forbidden in any frames encoded according to a Progressive or a Frame Interlace mode.

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.

User Data Start Code

A user data start code 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. The last user data byte of a user data structure (byte with value '0x80') shall always be followed by 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).

User data is typically used to carry closed caption data, bar data and metadata like the Active Format Description defined by DVB and ATSC.

User data may be present at various locations in a VC-9 bitstream. Although the value of any user data start code also specifies its scope (either sequence-level, entry point-level, frame-level, field-level, or slice-level user data), its location in a VC-9 bitstream shall follow the rules described below. The user data structure at any level may be duplicated as many time as is needed meaning that a user start code and the its user data bytes may be followed immediately by another user start code and its user data bytes having the same scope.

Sequence-level user data

Figure G.6 below shows sequence-level user data. As a general rule, 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 Independently Decodable Unit (IDU). 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 G.6, the top bitstream illustrates the case where the next IDU is an entry point start code followed by an entry point header while the bottom bitstream illustrates the case where the next IDU 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 sequence code or an another sequence start code is encountered in the bitstream.

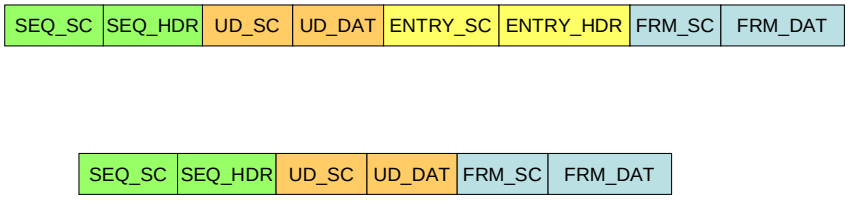


FIGURE G.6
Sequence-level User Data

Entry Point-level user data

Figure G.7 below shows entry point-level user data. As a general rule, entry point-level user data shall be located in the bitstream after the entry point-level header and immediately before the start code signaling the beginning of the start code for the next Independently Decoded Unit (IDU) – 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 VC-9 bitstream until another entry point start code or a sequence start code is encountered.

FIGURE G.7
Entry Point-level User Data

Frame-level User Data

Figure G.8 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 the RFF field 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 IDUs shall always be a multiple of the number of displayed fields/frames. An equal number of user data IDUs shall be then assigned to each field/frame; The user data IDUs shall be placed in the bitstream in the order of the field/frame to which they are assigned. Some of the user data IDUs may be empty.

The top two bitstreams in Figure G.8 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 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 shall appear immediately before the start code signaling the second slice within the frame.

The bottom two bitstreams in Figure G.8 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 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 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.

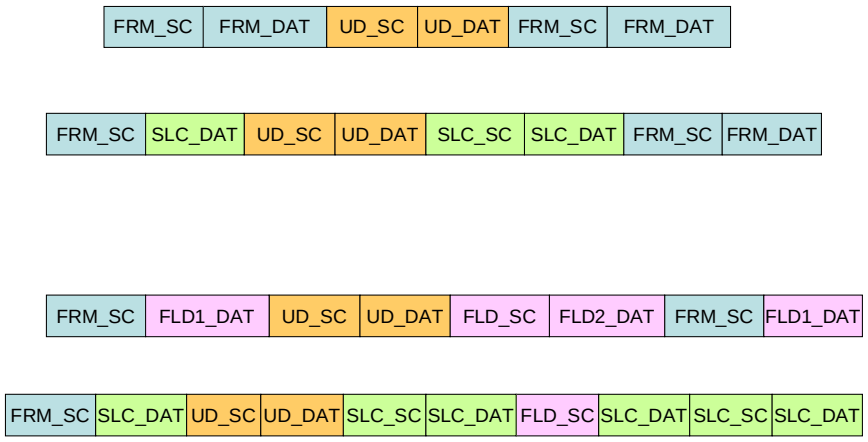


FIGURE G.8
Frame-level User Data

Field-level user data

Figure G.9 below shows field-level user data. Field-level user data shall only be allowed in frames that have 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 Y9, the top two bitstreams consider 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 shall appear before any frame-level user data. The value of the frame-level and field-level start code 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 shall appear before the start code of the second slice within the field. The fourth bistream shows the situation where frame-level user data is also present. In this case, field-level user data for the first field 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.

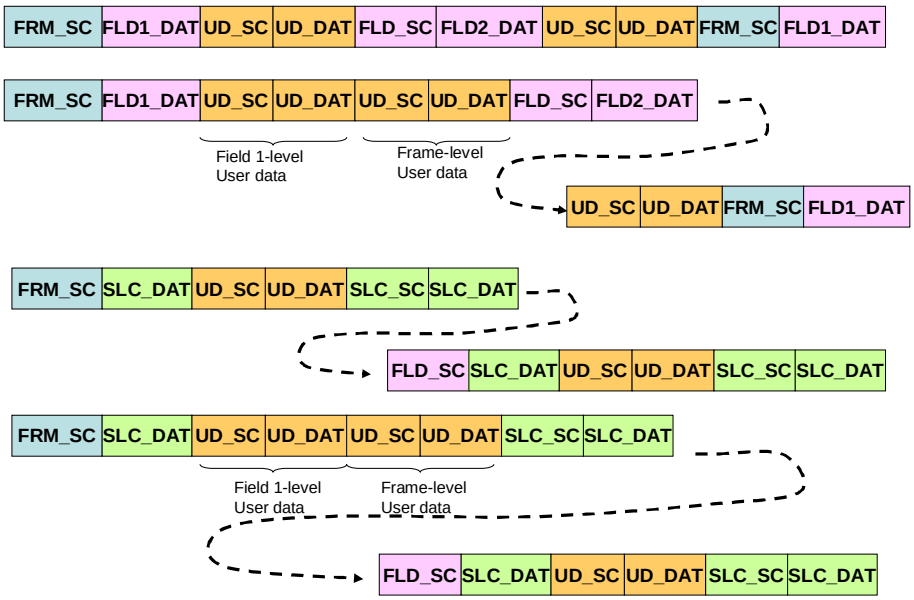


FIGURE G.9
Field-level User Data

Slice-level user data

Figure G.10 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 constraint in the design. 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 G.10, 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 applies 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 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 shall appear before any field-level user data. The value of the frame-level, field-level and slice start codes are distinct so the scope of the user data is unambiguous.

The third bistream 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 applies to a frame. In this case, the slice-level user data shall appear immediately before the start code for the next IDU – 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.

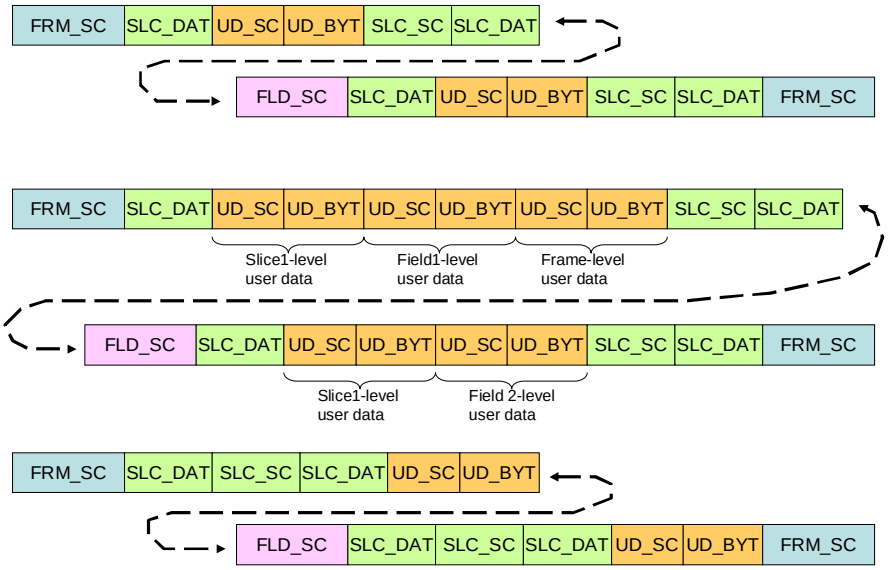


FIGURE G.10
Slice-level User Data

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 time and at any level. Use of sequence, entry point, frame, field and slice start codes is optional. 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.

To facilitate implementation of trick modes, the following additional constraint shall be observed:

- 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), then a field start code shall be present between the last data byte of the first “P” field and the field header of the second “I” field.