ICS

133HThe National Standard

GB/T ××××—××××

INFORMATION TECHNOLOGY ADVANCED AUDIO/ VIDEO CODING - Part 2: Video (Draft)

Released at ××××-××-××

Implemented at ××××-××-××

Released by National Bureau of Commodity Quality Inspection and Quarantine

CONTENTS

Foreword

This specification is established by China Digital Audio/Video Coding Technical Specification Workgroup in 2003.

Preface

0.1 Purpose

This specification was developed in response to the growing need for advanced coding technique of moving pictures in various applications, such as digital television broadcasting, digital storage media, Internet stream media and multimedia communication.

0.2 Application

The applications of this specification cover, but are not limited to, areas as listed below:

CATV Cable TV
DBS Direct Boradcast Satellite Video Services
Wideband Video Sservices
DTTB Digital Terrestrial Television Broadcasting
ISM Interactive Storage Media
MMM Multimedia Mailing
MSPN Multimedia Services on Packet Networks
Realtime Commucation Services (videoconferencing, videophone)
RVS Remote Video Surveillance

0.3 Profiles and levels

This specification can serve a wide range of bitrates, resolutions and qualities. “Profile” and “level” are defined here, considering the practicality of implementing the specification.

A “profile” is a defined subset of the entire syntax in this specification.

A “level” is a defined set of the constraints imposed on parameters of syntax and decoding procedure, in a certain “profile”.

0.4 Technical Summary

This specification adopted a serial of techniques to achieve high efficient video coding, including intra-frame prediction, inter-frame prediction, transform, quantisation and entropy coding etc. Inter-frame coding utilizes block-based motion vector for redundancy reduction between pictures; while intra-frame coding utilizes spatial predictive mode for redundancy reduction between pictures. In the end, motion vector, predictive mode, quantised parameter and transform coefficients are compressed with entropy coding.

0.4.1 Prediction Technique

Intra-frame prediction needs not to consult other pictures; pictures that adopted intra-frame prediction coding can be used as random visiting positions for the encoded sequence.

Inter-frame prediction needs to consult previous decoded pictures, and the decoding order may be different from the acquisition and the process order of source pictures in encoder, or the display order of decoder output. The precision of motion vector in inter-frame coding can reach ¼ pixel, in which prediction coding is utilized.

0.4.2 **Picture blocking**

In this specification, the basic processing unit of video decoding procedure is macroblock, which includes a 16×16 luminance block and corresponding 8×8 chrominance blocks. Macroblock further can be minimally partitioned to 8×8 sample blocks for prediction.

0.4.3 **Transform and Quantisation**

The transform unit is an 8×8 sample block. Transform coefficients are quantised with scalar quantity. In addition, this specification does not specify the processing methods of transform and quantisation in the encoder.

0.5 **How to read this specification**

Readers are suggested to start from chapter 1 (scope), then turn to chapter 3 (definitions). And structure of encoded video bitstream in chapter 6 should be read. Chapter 7 (syntax and semantics) define the syntax and semantics of bitstream. In section 7.1 syntax description is presented, and the appearance order of syntax elements in bitstream is defined. Section 7.2 gives semantics description, namely the scope, restriction and condition of syntax elements. The parsing process of most syntax elements is defined in chapter 8. At last, chapter 9 defines how to map syntax elements to decoding samples. Readers can also refer to chapter 2 (normative references), chapter 4 (abbreviations), chapter 5 (conventions) and appendix while reading this specification.

INFORMATION TECHNOLOGY ADVANCED AUDIO/VIDEO CODING PART 2: VIDEO

1 Scope

This document specifies the video part of advanced audio/video coding technique.

2 Normative References

The following recommendations contain provisions which through reference in this text, constitute provisions of this recommendation. In references that labeled with date, all the successive amend list (excluding corrigendum) or revised edition should not be applicable to this specification. Whereas, parties to agreements based on this recommendation are encouraged to investigate the possibility of applying the most recent editions of the specifications. All references, which are not labeled with date, have their newest version adapted to the specification.

3 Definitions

The following definitions are applied in this specification.

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

Backward Prediction: To predict with subsequent **Reference Picture** (in display order).

Bidirection Prediction: Using of elapsed and forthcoming **Reference Pictures** for prediction (in display order).

Bidirectionally Predictive-coded Picture: The picture that encoded with **Dual-direction** prediction.

Bit string: Limited bits with ordered sequence, the leftmost bit is Most**-Significant Bit**, and the rightmost bit is **Least-Significant Bit**.

Bitstream: Binary data stream generated by **Encoded Picture**.

Bitstream Buffer: Buffer that stores **Bitstream**.

Bitstream Order: The order in which the pictures are decoded.

Block: A M×N **Sample Data** matrix (M column and N row), or a M×N **Transform Coefficient** matrix.

Block Scan: The specific linear order mode of **Quantised Coefficient.**

Byte: A sequence of 8-bit string.

Byte Aligned: If a certain bit's position is integral times of 8-bit from the first bit in **Bitstream**, then that bit is byte aligned.**Chrominance**: A kind of modifier that represents the **Sample Data** matrix or single **Sample Data**. Symbols for chrominance are Cr and Cb. Here not chromatogram but chrominance is used, aiming to avoid using the linear light transfer feature related to term of chromatogram.

Coding Process: The process that generates **Bitstream** according with this specification. This process is not specified in the specification.

Compensation: To sum the sample residuals obtained from syntax elements' decoding and their corresponding predictive values.

Component: A matrix or a single **Sample Data** in matrix of the three picture sample matrices (one **Luminance** and two **Chrominance**).

Decoded Picture: Picture reconstructed by **Decoder** according to **Bitstream**.

Decoded Picture Buffer: The buffer that stores **Decoded Picture** for prediction, output reording and output time setting.

Decoder: Entity that implements **Decoding Processing**.

Decoder of Some Profile: **Decoder** that could decode **Birstream** specified by some profile.

Decoding Order: The order that decoding process performed on each frame, according to the prediction relationship between pictures.

Decoding Process: The process that produce **Encoded Picture** from syntax elements.

Decoding Processing: Including **Parsing Process**, **Decoding Process** and **Output Processing Process**.

Direct Current Coefficient (**DC Coefficient**): The coefficient for which the frequency is zero in both dimensions.

Display Order: The order of showing **Decoded Picture**.

Encoded Picture: **Coding Representation** of a frame.

Encoded Representation: Data format after coding.

Encoder: Implementation of **Coding Process**.

Forbidden: Defines some special syntax element values, which should not appear in **Bitstream** that according with this specification. This aims to avoid the appearance of pseudo start code in **Bitstream**.

Forward Inter-coded Picture (P frame): The picture that encoded with **Forward Prediction.**

Forward Prediction: The prediction with forepassed **Reference Picture** (in display order).

Inter-frame Coding: Coding of a **Macroblock** or picture that uses **Inter-frame Prediction**.

Inter-frame Prediction: The process of generating the current picture's sample predictive values with previous **Decoded Picture**.

Intra Encoded Picture (**I-frame**): A picture encoded only with **Intra-frame Coding**.

Intra-frame Coding: Coding of a **Macroblock** or picture with **Intra-frame Prediction**.

Intra-frame Prediction: In the same **Decoded Pictures**, the process of generating current sample's preditive values with previous decoded **Sample Values**.**Inverse Qiantisation**: The process of getting transform coefficient with **Quantised Coefficient** zoom.

Inverse Transform: A part of **Decoding Process** which transforms **Transform Coefficient** matrix to spatial domain sample matrix.

Layer: Scaled structure in bitstream, in which upper layer covers all layers below this layer. From top to bottom, the coding layers are: sequence, picture, slice, macroblock and block.

Level: A restricted set of syntax and **Decoding Process** in a certain **Profile**.

Luminance: A modifier that represents the **Sample Data** matrix of single color signal in primary colours, or a single **Sample Data.** The symble for luminance is Y. Not brightness but luminance employed here is to avoid using linear light transfer features which is generally related to luminance.

Macroblock: Including a 16×16 **Luminance** sample data block and its correponding **Chrominance** sample data block.

Macroblock Address: The sequence number along **Raster Scan**, begins with the top left corner **macroblock** of image whose sequence number is 0.

Macroblock Position: The two-dimensional coordinate of M**acroblock** in pictures which is denoted as (x,y), and for top left corner **Macroblock** (x,y)=(0,0). X is increased 1 from left to right for every **macroblock** column, and y is increased 1 from top to bottom for every **Macroblock** row.

Macroblock Row: Contineous macroblocks perpendicularly in encoded picture, from left border to right border. Macroblock row is 16 samples high.

Motion Vector: A two-dimensional vector used for inter-frame prediction, which provides an offset from the coordinate position in the **Decoded Picture** to the coordinates in a reference frame.

None Referred Pictures: In **Coding Process**, the pictures that not used in succeeding inter-frame prediction.

Output Order: The order of output **Decoded Picture** which is the same with **Display Order.**

Output Processing: The process of getting the ouput frame or field from **Decoded Pictures**.

Output Reordering Delay: The time from decoding a frame of picture in **Bitstream** to outputing its **Encoded Picture**. This delay is caused by the differences between **Display Order** and **Decoding Order** of pictures.

Parsing Process: The process of getting syntax elements from **Bitstream**.

Partition: The process that parts set to subsets. Every element in the set can only belong to a single subset.

Picture Reordering: The process of **Decoded Picture** reordering, in case of **Decoding Order** and **Output Order** being different.

Prediction: Implementation details of **Predict Process**.

Prediction Process: Using predictor to estimate the current decoding **Sample Values** or **Data Elements**.

Predictor: The combination of data elements or previously decoded **Sample Values**, during successive **Decoding Process** of **Sample Values** or data elements.

Profile: Subset of syntax specified by this specification.

Quantised Coefficient: **Transform Coefficient** value before **Inverse Quantisation**.

Quantised Parameter: Parameter that be used to **Inverse Quantisation** for **Quantised Coefficient** in **Decoding Process**.

Random Access: The ability of decoding **Bitstream** from a certain point (maybe not start of **Bitstream**), and recovering the **Decoded Picture**.

Random Access Point: The point that could be accessed randomly in **Bitstream**.

Raster scan: Mapping two-dimensional rectangle raster to one-dimensonal raster whose entrance starts from the first line of two-dimensional raster, and followed with the second line, the third line, and so on. Lines in raster are scanned from left to right.

Reference Index Table: The index table of **Reference Pictures** in **Decoded Picture** buffer.

Reference Pictures: Pictures used for successive inter-frame prediction in **Decoding Process**.

Reserved: Defines some specific syntax elements for future extension of the specification, which should not appear in bitstream that agrees with this specification.

Residual: Subtract samples or syntax elements' reconstructed values from their predictive values.

Run: The same successive data elements in **Decoding Process**. One side it means the number of zero coefficient before none-zero coefficient in **Block Scan**; On the other hand it means the number of skipped **Macroblocks**.

Sample Aspect Ratio (SAR): In a frame, the ratio between luminance samples' vertical row distances and horizontal column distances, represented by *h*:*v*, in which *h* is the horizontal width and *v* is the vertical height.

Sample Value: Sample's magnitude.

Skipped Macroblock: The macroblock with no other coding data except the flag "**Skipped**".

Slice: A series of **Macroblocks** or **Super Macroblocks** in order of **Raster Scan**.

Slice Header: A part of coding **Slice**, which is the **Coding Representation** of **Macroblock** or **Super Macroblock**'s public data elements in **Slice**.

Source: The term used for description of video materials or their certain properties before coding.

Start Code: A 32-bit length code which has unique modality in **Bitstream**. **Start Code** has many applications, one of which is to mark the beginning of **Bitstream**'s systax structure.

Stuffing Bits: **Bit String** that inserted into **Bitstream** in coding process, which will be discarded in decoding process.

Symbol: A binary variable.

Syntax Element: The data element in **Bitstream**.

Transform Coefficient: A scalar quantity in transform domain, corresponding to a one or two dimensional component of transform domain in **Inverse Transform**.

Variable Length Coding: A reversible entropy coding process, which assigns short code word to symbols with high frequency of appearance, and assigns long code word to symbols with low frequency of appearance.

Video Sequence: The highest syntax structure of encoded **Bitstream**, including one or more successive **Encoded Pictures**.

4 Abbreviations

BBV: Bitstream Buffer Verifier
CBR: Constant Bit Rate
CIF: Common Intermediate Format
LSB: Least Significant Bit
MB: Macroblock
MSB: Most Significant Bit
QCIF: Quarter Common Intermediate Format
SMB: Supermacroblock
VBR: Variable Bit Rate
VLC: Variable Length Coding

5 Conventions

The mathematical operators used to describe this specification are similar to those used in the C programming language. However, integer divisions and arithmetic shift operations are specifically defined. Convention numbering and counting begin from zero except special illustration is shown.

5.1 Arithmetic Operators

Arithmetic operators are defined as follows:

. Addition operation.

– Subtraction (as a binary operator) or negation (as a unary operator) operation.

× Multiplication operation.

a^b Power operation, representing a^b , or superscript.

/ Integer division operation. For example, 7/4 and -7/-4 are truncated to 1, and -7/4 and 7/-4 are truncated to -1.

. □ Used to denote division in mathematical equations where no truncation or rounding is intended.

$\frac{a}{b}$ Used to denote division in mathematical equations where no truncation or rounding is intended.

$\sum_{i=a}^{b} f(i)$ The summation of the f(i) with i taking integral values from a up to b(including b).

a % *b* Modular operation, representing the remainder of subtract b from a, a and b are all positive integer.

5.2 Logical Operators

Logical operators are defined as follows:

a && b Logical AND operation between a and b.

a || b Logical AND operation between a and b.

! Logical NOT operation.

5.3 Relational Operators

Relational operators are defined as follows:

> Greater than.

>= Greater than or is equal to.

< Less than.

<= Less than or is equal to.

== Is equal to.

!= Is not equal to.

5.4 Bitwise Operators

Bitewise operators are defined as follows:

& AND operation.

| OR operation.

~ Negate operation.

a>>b Shift a b bits right in form of integral complement of 2, which is only defined when b is positive. When shifting right to the highest valid bit, its value is equal to the highest valid bit before shift operation of a.

a << b Shift a b bits left in form of integral complement of 2, which is only defined when b is positive. When shifting left to the lowest valid bit, its value is equal to zero.

5.5 Assignment

Assignment operators are defined as follows:

= Assignment operator.

x =a..b X is evaluated with the value from a to b (including b), and x, a and b are integer.

++ Add to itself, and x++ means x = x + 1. When used in array subscript, variable value should be calculated before this adding operation.

— Subtracting from itself, and x— means x = x – 1. When used in array subscript, variable value should be calculated before this subtracting operation.

+= Adding specified value to itself, for instance, x += 3 means x = x + 3, x += (–3) means x = x + (–3).

–= Subtracting specified value from itself, for instance, x –= 3 means x = x – 3, x –= (–3) means x = x – (–3).

5.6 Mathematical Functions

Mathematical functions are defined as follows:

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

Ceil(x) X is evaluated with its integer supremum (5-2)

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

$$\text{Clip3}(a, b, c) = \begin{cases} a & ; \quad c < a \\ b & ; \quad c > b \\ c & ; \quad \text{otherwise} \end{cases} \tag{5-4}$$

Floor(x) The maximum integer not greater than x (5-5)

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

Log2(x) Logarithm base-2 of x (5-7)

Log10(x) Natural logarithm of x (5-8)

Median(x,y,z) = x + y + z – Min(x, Min(y, z)) – Max(x, Max(y, z)) (5-9)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x >= y \end{cases} \quad (5\text{-}10)$$

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

Round(x) = Sign(x) × Floor(Abs(x) + 0.5) (5-12)

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x \geq 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (5\text{-}13)$$

$$\text{Sqrt}(x) = \sqrt{x} \quad (5\text{-}14)$$

5.7 Description of bitstream syntax, parsing process and decoding process

5.7.1 Description method

The description method of bitstream is similar to C programming language. Syntax elements of bitstream are denoted with bold font, and each syntax element is described by name (English letter groups partitioned with underlines, and all letters are in lower case), syntax and semantics. The decoder operates based on the current syntax element value and the incorparation with syntax elements decoded before. The syntax element values in syntax table and context are represented with normal fonts.

In some cases, syntax table can use other variable values derivated from syntax elements, and such values are named with mixture of none-underlined lower and upper case letters in syntax table or context. Variables start with upper case letters are used to decode the current or related syntax structure, or the following syntax structure. Variables start with lower case letters can only be used in the section where they are.

The mnemonics of syntax element value, variable value and the relationship between them are explained in context. Sometimes the two kinds of values can be used equally. Mnemonics is represented with letter groups which are separated with one or more underlines. Every group starts with upper case letter, and can include several upper case letters.

When bit number is integral times of 4, then hexadecimal symbols are used for representation. The prefix of hexadecimal is ‘0x’, for example, ‘0x1a’ stands for bit string ‘00011010’.

In conditional sentences zero denotes FALSE, and “non-zero” denotes TURE.

Syntax table describes all superset of bitstream syntax that according with this specification, the additive syntax restriction can be explained in other sections.

The following table shows a pseudo start code example of syntax description. The appearance of syntax elements means to read a data element from the bitstream.

	Descriptor
/* The sentence can be a descriptor of a syntax element, or be used to show the existence, type and value of syntax element. Next are two examples. */	

syntax_element	ue(v)
conditioning statement	
/* The sentence group in brackets is compound sentence，and can be treated as a single sentence functionally */	
{	
statement	
statement	
…	
}	
/* “While” sentence is used to test whether the condition is TRUE, if TRUE, the cycle part will be executed repeatedly until the condition is not TRUE. */	
while (condition)	
statement	
/* “Do … while” sentence executes the cycle part once，then test whether the condition is TRUE, if TRUE, the cycle part will be executed repeatedly until the condition is not TRUE. */	
Do	
Statement	
while (condition)	
/* “If … else” sentence tests condition first, if TRUE, then executes primary sentence, otherwise executes alternative sentence. If alternative sentence doesn’t need to be executed, then the “else” part in structure and related alternative sentence can be ignored. */	
if (condition)	
primary statement	
else	
alternative statement	
/* “For” sentence executes initial sentence, then tests condition, if TRUE, then executes primary and subsequent sentence repeatedly until condition is not TRUE. */	
for (initial statement; condition; subsequent statement)	
primary statement	

The parsing and decoding process is described with pseudo code that resembles C programming language.

5.7.2 **Function and descriptor**

The following functions are used for syntax description. A bitstream pointer is supposed to exist in decoder, which points to the position of next bit to be read in bitstream. A function is composed of its name and parameters between parentheses. There can also have no parameters in functions.

byte_aligned( )

If the current position of bitstream is byte aligned, then returns TRUE, otherwise returns FALSE.

next_bits(n)

Return the following n bits in bitstream, MSB foregoing and without change of bitstream pointer. If the remaining bit number is less than n, then returns zero.

next_mb_address(i)

Providing the address of macroblock which is next to the macroblock with address i in group (according to the macroblock address order).

next_start_code( )

Looking for the next starting code in bitstream, function definition is showed in the following table.

next_start_code() {	Descriptor
if (! byte_aligned())	
stuffing_bit	'1'
while (! byte_aligned())	
stuffing_bit	'0'
while (next_bits() != '0000 0000 0000 0000 0000 0001')	
stuffing_byte	'1000 0000'
}	

read_bits(n)

Return the following n bits in bitstream, MSB foregoing and at the same time shift the Bitstream pointer left with n bits. If n equals zero, then returns zero, and without shifting bitstream pointer.

The function can also be used to describe parsing and decoding process.

The following descriptors represent the parsing process of syntax elements.

b(8)

A randomly evaluated byte. The parsing process is specified by return value of function read_bits(8).

ce(v)

Variable length coding syntax elements start with left bit. The parsing process is defined in section 8.3.

f(n)

N successive bits evaluated with specific values. The parsing process is specified by the return value of function read_bits(n).

i(n)

A n-bit integer. In syntax table, the bit number is determined with the values of other syntax elements if n is 'v'. The parsing process is specified by the return value of function read_bits(n), which is represented with complement code of 2 with high bit in front.

me(v)

Syntax elements of mapping exponential Columbus code which start with left bit. Its parsing process is defined in section 8.2.

se(v)

Syntax elements of signed integral exponential Columbus code which start with left bit. Its parsing process is defined in section 8.2.

te(v)

Syntax elements of cut-offed exponential Columbus code which start with left bit. Its parsing process is defined in section 8.2.

u(n)

A n-bit integer. In syntax table, the bit number is determined by the values of other syntax elements if n is 'v'. The parsing process is specified by the return value of function read_bits(n), which is represented with high bit foregoing binary representation.

ue(v)

Syntax elements of unsigned integral exponential Columbus code which start with left bit. Its parsing process is defined in section 8.2.

5.7.3 Reserved and forbidden

The terms "reserved" and "forbidden" are used in some description of several syntax elements in the bitstream defined by the specification.

Reserved defines several special values of syntax elements may be used in the future extension of this specification, which should not appear in bitstream that according with the specification.

Forbidden defined several special values of syntax elements which should not appear in bitstream that according with the specification.

6 Structure of encoded bitstream

This chapter shows the structure of bitstream after coding, and also their layer relationship and processing order.

6.1 Video sequence

Video sequence is the highest syntactic structure of bitstream, starting with sequence head and followed by one or more encoded frames which should have a sequence head in front of the frame. The order of encoded picture in bitstream is the order of bitstream, which is the same as display order. But the decoding order may be different with the display order. The end code in sequence indicates the end of a video sequence.

6.1.1 Progressive and interlaced video sequence

There are two kinds of different sequences in this specification: progressive and interlaced sequence.

The frame consists of three integral sample matrices, including a luminance sample matrix (Y) and two chromaticity sample matrices (Cb and Cr). The relationship between these Y, Cb and Cr components and the primary (analogue) Red, Green and Blue signals (E'R, E'G and E'B), also the chromaticity of original signals and the transfer characteristics may be specified in the bitstream. The above information will not affect the decoding process.

A field is composed of interlaced lines of the above three sample matrices which represent a frame, namely the first line, third line, fifth line, etc., composing a field called top field; and the second line, fourth line, sixth line, etc., composing a field called bottom field.

The decoder output is a series of frames, and there is a time interval between two frames. For interlaced sequence, there is a time interval between two fields of every picture. For progressive sequence, the time interval between two fields is zero.

6.1.2 Sequence header

The video header starts with a sequence header code, and is followed by a series of encoded picture elements.

Sequence header may appear repeatedly in bitstream, called repeated sequence header. This allows the random access into the video sequence being possible.

The first encoded picture in video sequence must be an I frame. In circumstance of editing or random access bitstream, all data before repeated sequence header can be discarded, and then the resulting bitstream can be a legal bitstream that complies with this specification.

6.2 Picture

A picture is a frame with encoded data begins with the start code of picture, and ends with the start code of sequence, end code of sequence or start code of picture.

In bitstream, the encoded data of two fields in progressive scan picture can appear in turn, or with interlace turn. The order of two fields can be described in picture header.

The decoding process of a picture includes parsing and decoding processing.

6.2.1 Picture format

The width and height of luminance matrix should be integral times of 16 samples, and the width and height of chrominance matrix should be integral times of 8 samples. If there is a super macroblock in bitstream, then the width and height of luminance matrix should be integral times of 32 samples, and the width and height of chrominance matrix should be integral times of 16 samples. The width and height of decoder output picture need not to be integral times of 16 samples; a cut-off rectangle can be used instead.

6.2.1.1 4:2:0 format

In this format the Cb and Cr matrices can be one half the size of the Y-matrix in both horizontal and vertical dimensions.

The luminance and chrominance samples are positioned as shown in Figure 6-1, in which ‘x’ represents luminance samples and ‘o’ represents chrominance samples.

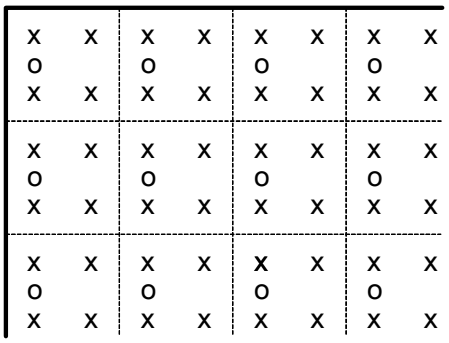


Figure 6-1 The position of luminance and chrominance samples in 4:2:0 format

6.2.1.2 4:2:2 format

For 4:2:2 format, the Cb and Cr matrices can be one half the size of the Y-matrix in horizontal dimensions, and be the same size of the Y-matrix in vertical dimensions.

The luminance and chrominance samples are positioned as shown in Figure 6-2, in which ‘x’ represents luminance samples and ‘o’ represents chrominance samples.

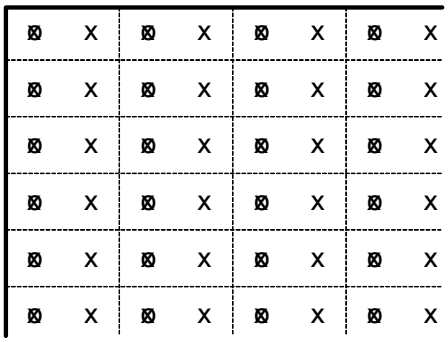


Figure 6-2 The position of luminance and chrominance samples in 4:2:2 format

6.2.1.3 4:4:4 format

In this format the Cb and Cr matrices can be the same the size of Y-matrix in both horizontal and vertical dimensions.

The luminance and chrominance samples are positioned as shown in Figure 6-3, in which ‘x’ represents luminance samples and ‘o’ represents chrominance samples.

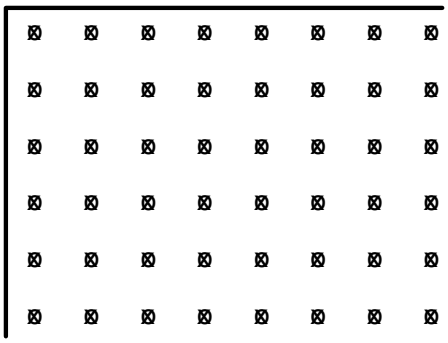


Figure 6-3 The position of luminance and chrominance samples in 4:4:4 format

6.2.2 Picture type

The specification defines three types of decoding pictures:

- Intra-coded picture (I frame): without referring to other pictures during decoding.
- Predictive inter-coded picture (P frame): the reference picture is before the current picture (in display order).
- Bidirectional inter-coded picture (B frame): the reference picture is before and after the current picture (in display order).

6.2.3 Picture re-ordering

If there are no B frames in the bitstream, then the bitstream and the decoding process have the same order. If there are B frames in the video sequence, then the bitstream, decoding and display have different order. Decoded picture should be reordered before output display. Picture reordering should obey the following rules:

- When the current picture is a B frame, then the output picture is decoded from this B frame.

- When the current picture is a I or a P frame, and there exists a picture decoded from past I or P frame, then the output is the current picture; when not exists a picture decoded from past I or P frame, then no picture is output.

The followings are examples of picture reordering: there are two B frames between I frame and P frame, and also two B frames between two successive P frames. Frame 1I is used to predict frame 4P, and frame 4P and 1I are used to predict frame 2B and 3B. Decoding order is 1I, 4P, 2B and 3B; display order is 1I, 2B, 3B and 4P.

At the encoder input:

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

Bitstream order:

1	4	2	3	7	5	6	10	8	9	13	11	12
I	P	B	B	P	B	B	I	B	B	P	B	B

Decoder output, namely display order:

1	2	3	4	5	6	7	8	9	10	11	12	13

6.2.4 Reference pictures

A P-frame or B-frame can at most have two reference pictures. A P-frame may refer to the forward two frames. In a frame, later decoded field can also refer to the other field of the current frame. A B-frame can refer to the previous and the next two frames.

Motion vector can exceed reference pictures' boundary, and in such case the boundary extension should be achieved by using the pixel in picture which is nearest to the position of motion vector. All pixels for constructing reference block can not exceed more than 16 pixels of reference picture borders in both horizontal and vertical directions.

6.3 Slice

A slice is composed of several successive macroblock lines in raster scan order, in which macroblock lines or slices cannot be overlapped. The decoding process of macroblock in slice should not refer to other slices in the picture.

Reference picture border extension method is the same as that of slice, see section 6.2.4 for details.

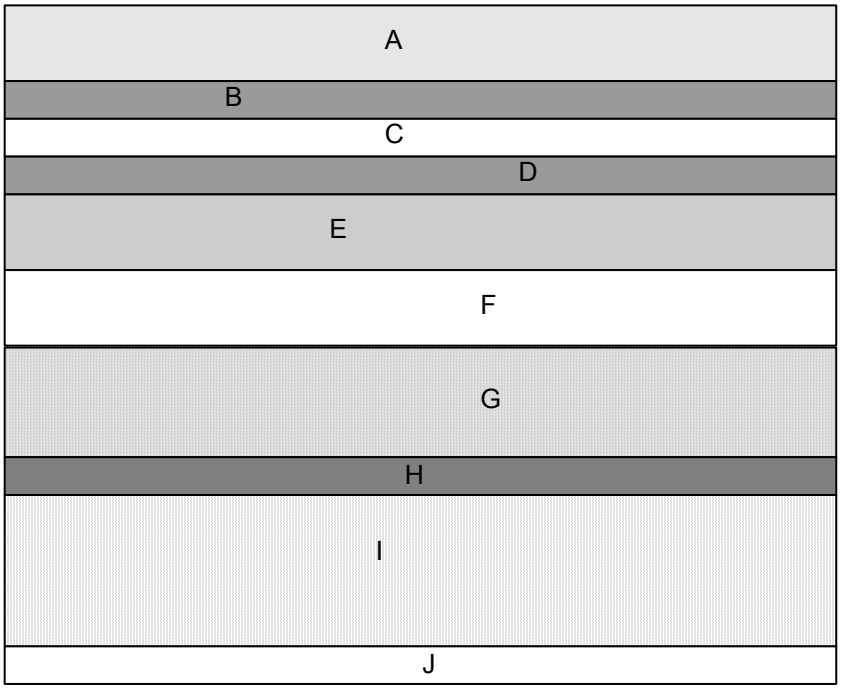


Figure 6-4. Slice structure

6.4 Macroblock

A macroblock has two modes: the unsampled and the sampled macroblock, as illustrated in figure 6-5a and 6-5b.

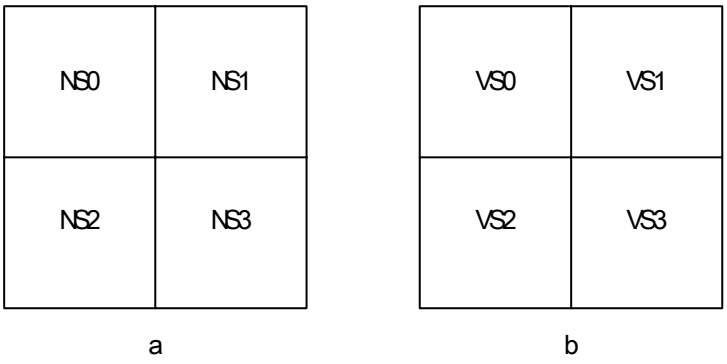


Figure 6-5 Macroblock sampling

An unsampled macroblock is obtained without any sampling of pictures. It is composed of NS0, NS1 and NS2 blocks, in which NS0 and NS1 are upper layer blocks, Ns2 and NS3 are lower layer blocks.

A sampled macroblock is obtained with sampling of every other line in pictures. It is composed of VS0, VS1 and VS2 blocks, in which VS0 and VS1 are upper layer blocks made up of even lines, Ns2 and NS3 are lower layer blocks made up of odd lines.

A picture is partitioned into macroblocks, representing spatial rectangle regions in the picture. Points at top left corner of a macroblock can not exceed picture border. In bitstream, when the two encoded frame data of interlaced scan picture appears in turn, pixels of any macroblock should come from the same field.

The macroblock partition used for motion compensation is showed in figure 6-6. The numbers in rectangle represent the order of motion vector and reference index in code stream after macroblock partition.

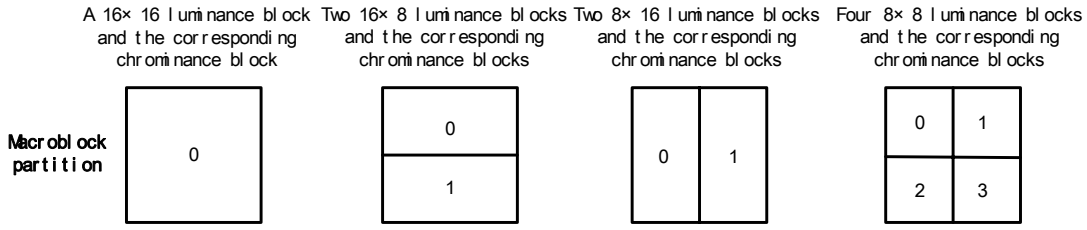


Figure 6-6. Macroblock partition

6.5 8×8 block

An 8×8 block is the basic element of transform and block scan, which can represent source picture data, reconstructed picture data and 8×8 integral transform coefficients.

In 4:2:0 format, a macroblock includes four 8×8 luminance blocks (Y) and two chromance blocks (Cb and Cr), just as figure 6-7 shows.

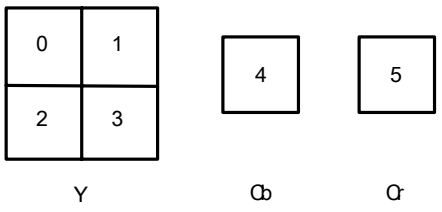


Figure 6-7. Macroblock partitioned into 8×8 block (4:2:0 format)

In 4:2:2 format, a macroblock includes four 8×8 luminance blocks (Y) and four chromance blocks (two Cb and two Cr), just as figure 6-8 shows.

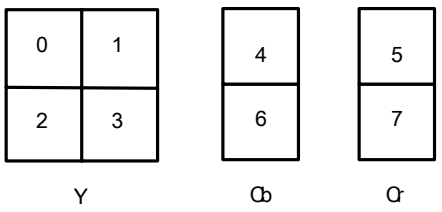


Figure 6-8. Macroblock partitioned into 8×8 block (4:2:2 format)

In 4:4:4 format, a macroblock includes four 8×8 luminance blocks (Y) and eight chromance blocks (four Cb and four Cr), just as figure 6-9 shows.

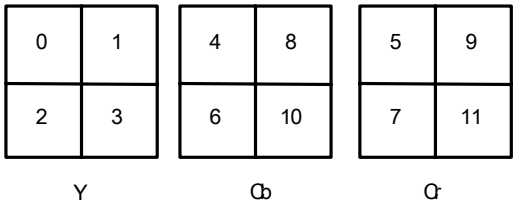


Figure 6-9. Macroblock partitioned into 8×8 block (4:4:4 format)

The block order in the bitstream is determined by numbers in figure 6-7 up to figure 6-9.

7 Video bitstream syntax and semantics

7.1 Syntax description

7.1.1 Start codes

Start codes are a specific bit patterns that do not appear in bitstream, otherwise occur in the video stream.

Each start code is composed of its prefix and value, and the prefix is a bit string of "0000 0000 0000 0000 0000 0001". All start codes should be byte aligned.

The value of start code is a 8-bit integer and is used to represent the type of start code, see table 7-1.

Table 7-1. Start code values

Start code names	**Start code values (Hexadecimal)**
Slice start code (**slice_start_code**)	From 00 to AF
Video sequence start code (**video_sequence_start_code**)	B0
Video sequence end code (**video_sequence_end_code**)	B1
User data start code (**user_data_start_code**)	B2
I picture start code (**i_picture_start_code**)	B3
Video error code (**video_error_code**)	B4
Video extension start code (**extension_start_code**)	B5
PB picture start code (**pb_picture_start_code**)	B6
Video editing code (**video_edit_code**)	B7
Reserved	B8 - XX
System start code *	XX -XX
* System start code is defined in the first part of this specification.	

When some syntax elements are evaluated with specified values, bit string of the same prefix with start code can be accquired, which are called "pseudo start code". Encoders and decoders according to this specification should use methods defined in Appendix C, to avoid the appearance of pseudo start code in the bitstream.

7.1.2 Video sequence

video_sequence() {	Descriptor
next_start_code()	
do {	
sequence_header)	
extension_and_user_data(0)	
do {	
if (next_bits(32) == **video_edit_code**)	
video_edit_code	u(32)
if (next_bits(32) == **i_picture_start_code**)	

i_picture_header()	
else	
pb_picture_header()	
extension_and_user_data(1)	
picture_data()	
} while ((next_bits(32) == **video_edit_code**) || (next_bits(32) == **pb_picture_start_code**) || (next_bits(32) == **i_picture_start_code**))	
} while (next_bits(32) != **sequence_end_code**)	
video_sequence_end_code	f(32)
}	

7.1.2.1 Sequence header

sequence_header() {	描述符
video_sequence_start_code	f(32)
profile_id	u(8)
level_id	u(8)
progressive_sequence	u(1)
horizontal_size	u(14)
vertical_size	u(14)
chroma_format	u(2)
sample_precision	u(3)
aspect_ratio_information	u(4)
frame_rate_code	u(4)
bit_rate	u(30)
low_delay	u(1)
next_start_code()	
}	

7.1.2.2 Extension and user data

extension_and_user_data(i) {	Descriptor
while ((next_bits(32) == **extension_start_code**) || (next_bits(32) == **user_data_start_code**)) {	
if (next_bits(32) == **extension_start_code**)	
extension_data(i)	
if (next_bits(32) == **user_data_start_code**)	
user_data()	
}	
}	

7.1.2.2.1 Extension data

extension_data(i) {	Descriptor
while (next_bits(32) == **extension_start_code**) {	
extension_start_code	f(32)

if (i == 0) { /* Following the sequence header */	
if (next_bits(4) == ‘0010’) /* Sequence display extension */	
sequence_display_extension()	
else if (next_bits(4) == ‘0100’) /* Copyright extension */	
copyright_extension()	
else if (next_bits(4) == ‘1011’) /* Camera parameter extension */	
camera_parameters_extension()	
else {	
while (next_bits(24) != ‘0000 0000 0000 0000 0000 0001’)	
reserved_extension_data_byte	u(8)
}	
}	
else { /* Following the picture header*/	
if (next_bits(4) == ‘0100’) /* Copyright extension */	
copyright_extension()	
else if (next_bits(4) == ‘0111’) /* Picture display extension */	
picture_display_extension()	
else if (next_bits(4) == ‘1011’) /* Camera parameter extension */	
camera_parameters_extension()	
else {	
while (next_bits(24) != ‘0000 0000 0000 0000 0000 0001’)	
reserved_extension_data_byte	u(8)
}	
}	
}	
}	

7.1.2.2.2 User data

user_data() {	Descriptor
user_data_start_code	f(32)
while (next_bits(24) != ‘0000 0000 0000 0000 0000 0001’) {	
user_data	b(8)
}	
next_start_code()	
}	

7.1.2.3 Sequence display extension

sequence_display_extension() {	Descriptor
extension_id	f(4)
video_format	u(3)
video_range	u(1)
colour_description	u(1)
if (**colour_description**) {	

colour_primaries	u(8)
transfer_characteristics	u(8)
matrix_coefficients	u(8)
}	
display_horizontal_size	u(14)
display_vertical_size	u(14)
next_start_code()	
}	

7.1.2.4 Copyright extension

copyright_extension() {	Descriptor
extension_id	f(4)
copyright_flag	u(1)
copyright_id	u(8)
original_or_copy	u(1)
reserved	f(7)
copyright_number	u(64)
next_start_code()	
}	

7.1.2.5 Camera parameter extension

camera_parameters_extension() {	Descriptor
extension_id	f(4)
reserved	f(1)
camera_id	u(7)
height_of_image_device	u(22)
focal_length	u(22)
f_number	u(22)
vertical_angle_of_view	u(22)
camera_position_x	i(32)
camera_position_y	i(32)
camera_position_z	i(32)
camera_direction_x	i(22)
camera_direction_y	i(22)
camera_direction_z	i(22)
image_plane_vertical_x	i(22)
image_plane_vertical_y	i(22)
image_plane_vertical_z	i(22)
reserved	f(32)
next_start_code()	
}	

7.1.3 Picture

7.1.3.1 I Picture header

i_picture_header() {	Descriptor
i_picture_start_code	f(32)
time_code_flag	u(1)
if (**time_code_flag** == '1')	
time_code	u(24)
picture_distance	u(8)
progressive_frame	u(1)
if (**progressive_frame** == '0')	
picture_structure	u(1)
top_field_first	u(1)
repeat_first_field	u(1)
fixed_picture_qp	u(1)
picture_qp	u(6)
skip_mode_flag	u(1)
loop_filter_disable	u(1)
if (! **loop_filter_disable**) {	
loop_filter_parameter_flag	u(1)
if (**loop_filter_parameter_flag**) {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
next_start_code()	
}	

7.1.3.2 PB Picture header

pb_picture_header() {	Descriptor
pb_picture_start_code	f(32)
picture_distance	u(8)
progressive_frame	u(1)
if (**progressive_frame** == '0') {	
picture_structure	u(1)
if (**picture_structure** == '0')	
advanced_pred_mode_disable	u(1)
}	
top_field_first	u(1)
repeat_first_field	u(1)
fixed_picture_qp	u(1)

picture_qp	u(6)
picture_coding_type	u(2)
if (! (**picture_coding_type** == '10' && PictureStructure == 1))	
picture_reference_flag	u(1)
skip_mode_flag	u(1)
if (PictureType != 2) {	
loop_filter_disable	u(1)
if (! **loop_filter_disable**) {	
loop_filter_parameter_flag	u(1)
if (**loop_filter_parameter_flag**) {	
alpha_c_offset	se(v)
beta_offset	se(v)
}	
}	
}	
next_start_code()	
}	

7.1.3.3 Picture display extension

picture_display_extension() {	Descriptor
extension_id	f(4)
for (i = 0; i < NumberOfFrameCentreOffsets; i ++) {	
frame_centre_horizontal_offset	i(16)
frame_centre_vertical_offset	i(16)
}	
next_start_code()	
}	

7.1.3.4 Picture data

picture_data() {	Descriptor
do {	
slice()	
} while (next_bits(32) == **slice_start_code**)	
next_start_code()	
}	

7.1.4 Slice

slice() {	Descriptor
if (next_bits(32) == **slice_start_code**) {	
slice_start_code	f(32)

if (**fixed_picture_qp** == '0') {	
fixed_slice_qp	u(1)
slice_qp	u(6)
}	
}	
do {	
if (PictureType != 0 || (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2)) {	
if (**skip_mode_flag** == '1')	
mb_skip_run	ue(v)
}	
if (MbIndex < MbWidth × MbHeight - 1)	
macroblock()	
} while (MbIndex < MbWidth × MbHeight - 1)	
next_start_code()	
}	

7.1.5 Macroblock

macroblock() {	Descriptor
if (PictureType != 0 || (PictureStructure == 0 && MbIndex >= MbWidth × MbHeight / 2))	
mb_type	ue(v)
if (MbType != 'P_Skip' && MbType != 'B_Skip') {	
if (MbType == 'B_8x8')	
for (i=0; i<4; i++)	
mb_part_type	u(2)
if (MbType == 'I_8x8') {	
for (i=0; i<4; i++) {	
pred_mode_flag	u(1)
if (! **pred_mode_flag**)	
intra_luma_pred_mode	u(3)
}	
intra_chroma_pred_mode	ue(v)
}	
if (**picture_reference_flag** == 0) {	
if (PictureType == 1 || (PictureType == 2 && PictureStructure == 0))	
for (i = 0; i<MvNum; i++)	
mb_reference_index	u(1)/u(2)
}	
for (i = 0; i < MvNum; i++) {	
mv_diff_x	ue(v)
mv_diff_y	ue(v)
}	
if (! ((MbTypeIndex >= 24 && PictureType == 2) || (MbTypeIndex >= 5 && PictureType != 2)))	
cbp	me(v)

if (MbCBP > 0) {	
if (! FixedQP)	
mb_qp_delta	se(v)
}	
for (i = 0; i < 6; i++)	
block(i)	
}	
}	

7.1.6 Block

block(i) {	Descriptor
if (MbCBP & (1 << i)) {	
do {	
trans_coefficient	ce(v)
if (**trans_coefficient** == 59) {	
escape_level	ce(v)
escape_run	ce(v)
}	
} while (**trans_coefficient** != 'EOB')	
}	
}	

7.2 Semantics description

7.2.1 Semantic rules for higher syntactic structures

This section defines how to combine the semantic rules of higher syntactic structures together, then produces a legal bitstream.

The specification defines several video extensions, and for different positions of syntax the corresponding extensions can be different too. Every video extension can only have one video extension identifier, see table 7-2. If a decoder encounters an extension with an extension identifier that is described as “reserved” in this specification, the decoder shall discard all subsequent data until the next start code.

Table 7-2. Video extension identifier

Video extension identifier	**Meaning**
0000	Reserved
0001	Reserved
0010	Sequence display extension
0011	Reserved
0100	Copyright extension
0101	Reserved
0110	Reserved
0111	Picture display extension

1000 - 1010	Reserved
1011	Camera parameter extension
1100 - 1111	Reserved

7.2.2 Video sequence

Video_edit_code

A bit string with value “0x000001B7”, which shows there is a lack of reference frames in the successive B frame after the first T frame, and can not be rightly decoded. **Video_edit_code** should not appear before **pb_picture_start_code**.

Video_sequence_end_code

A bit string with value “0x000001B1”, standing for the end of video sequence.

7.2.2.1 Sequence header

Video sequence start code - Video_sequence_start_code

A bit string with value “0x000001B0”, standing for the begining of video sequence.

Profile identifier - profile_id

A 8-bit unsigned integer, representing the profile of bitstream.

Level identifier - level_id

A 8-bit unsigned integer, representing the level of bitstream.

See profile and level in Appendix B.

Progressive sequence identifier - progressive_sequence

Specifing the scan format of video sequence. If progressive sequence is “1”, then the video sequence only includes progressive scan frames; If progressive sequence is “0”, then the video sequence includes progressive or interlaced scan frames.

Horizontal size - horizontal_size

A 14-bit unsigned integer, which identifies the width of the displayable part (aligned with left edge of the picture) of the pictures’ luminance component, namely horizontal samples’ number.

Based on macroblock, the width of display part is calculated as

$$MBWidth = (\textbf{horizontal\_size} + 15) / 16$$

Horizontal_size cannot be zero.

Vertical size - vertical_size

A 14-bit unsigned integer, which identifies the height of the displayable part (aligned with top edge of the picture) of the pictures’ luminance component, namely vertical lines’ number.

In bitstream, when two fields’ encoded data of interlaced scan picture appears in turn, then the height of display region based on macroblock is calculated as:

$$MBHeight = 2 \times ((\textbf{vertical\_size} + 31) / 32)$$

In other cases, the height of display region based on macroblock is calculated as:

$$MBHeight = (\textbf{vertical\_size} + 15) / 16$$

Vertical_size cannot be zero.

Chrominance format - chroma_format

A 2-bit unsigned integer, which identifies the format of chrominance component. See table 7-3.

Table 7-3. Chrominance format

chroma_format	Meaning
00	Reserved
01	4:2:0
10	Reserved
11	Reserved

Sample precision sample_precision

A 3-bit unsigned integer, which identifies the precision of luminance and chrominance samples. See table 7-4.

Table 7-4. Sample precision

sample_precision	Meaning
000	Forbidden
001	Luminance and chrominance are both 8-bit precision.
010 - 111	Reserved

Width/height ratio – aspect_ratio_information

A 4-bit unsigned integer, which identifies the sample aspect ratio (SAR) or the display aspect ratio (DAR) of the reconstructed picture. See table 7-5.

Table 7-5. **A**spect_ratio_information

aspect_ratio_information	SAR	DAR
0000	Forbidden	Forbidden
0001	1.0	–
0010	–	4 ÷ 3
0011	–	16 ÷ 9
0100	–	2.21 ÷ 1
0101 - 1111	–	Reserved

If sequence_display_extension is not present in the bitstream, then the entire reconstructed frame is intended to be mapped to the active region of display. The sample aspect ratio may be calculated as follows:

SAR = (**horizontal_size** / **vertical_size**) ÷ DAR

Note: In this case horizontal_size and vertical_size are constrained by the SAR of the source picture and the selected DAR.

If sequence_display_extension is presented in bitstream, then the sample aspect ratio may be calculated as follows:

SAR = (**display_horizontal_size** / **display_vertical_size**) ÷ DAR

Frame rate code - frame_rate_code

A 4-bit unsigned integer, which identifies the frame rate as showed in table 7-6.

Table 7-6. Frame rate code

frame_rate_code	Frame rate

0000	Forbidden
0001	24 000 ÷ 1001 (23.976…)
0010	24
0011	25
0100	30 000 ÷ 1001 (29.97…)
0101	30
0110	50
0111	60 000 ÷ 1001 (59.94…)
1000	60
1001 - 1111	Reserved

The period between two successive frames is the reciprocal of the frame_rate. The period between two successive fields in progressive scan frame is half of the reciprocal of the frame_rate.

Bitstream rate - bit_rate

A 30-bit unsigned integer. Bitrate of video bitstream is calculated with the speed of 400 bits/second, and integrited upward. Bitrate can not be zero.

low_delay

A flag, with value of "1" for excluding B frames in video sequence, and without bitstream reordering and picture reordering delay; with value of "0" for B frames may be included in video sequence, and with bitstream reordering and picture reordering delay.

Low_delay is used in decoding process and it can be ignored by decoder. This flag is mainly used to define low delay bitstream, and verify the coherence of low delay bitstream and specification.

bbv_buffer_size

A 18-bit unsigned integer, which identifies the bitstream buffer size of video sequence decoding, with the reference of decoder (see also Appendix D). BBS, the minimum size of the bitstream buffer needed by the reference decoder to decode the sequence (counted by bit), is defined as

$$BBS = 16 \times 1024 \times \textbf{bbv\_buffer\_size}$$

7.2.2.2 Extension and user data

7.2.2.2.1 Extension data

Video extension start code - extension_start_code

A bit string with value "0x000001B5", which indicates the beginning of video extension data.

Reserved_extension_data_byte

An 8-bit unsigned integer, which are reserved bits that should be discarded by decoder.

7.2.2.2.2 User data

User data start code - user_data_start_code

A bit string with value "0x000001B2", which identifies the start of user data. User data is stored successively until the next start code.

User data – user_data

An 8-bit integer, whose meaning is defined by user himself. There can not have over 21 successive zeros in user data.

7.2.2.3 Sequence display extension

Display process is not defined in this specification. The information in this extension has no effect on the decoding process, and may be ignored by decoder.

Video extension identifier - extension_id

A bit string with value “0010”.

Video format – video_format

A 3-bit unsigned integer, which identifies the video format before encoding according to this specification, see table 7-7. If the sequence display extension does not appear in bitstream, then the video format can be assumed as “unspecified video format”.

Table 7-7. Video format

video_format	Meaning
000	Component signal
001	PAL
010	NTSC
011	SECAM
100	MAC
101	Unspecified video format
110	Reserved
111	Reserved

Video range - video_range

Identifier which represents the sample range of luminance and chrominance signal. If no sequence display extension appears in bitstream, then video_range is supposed to be zero.

Color description - colour_description

Identifier with value “1” for colour_primaries, transfer_characteristics and matrix_coefficients exists in bitstream; and with value “0” for colour_primaries, transfer_characteristics and matrix_coefficients not exists in bitstream.

Colour promaries - colour_primaries

An 8-bit unsigned integer which describes the chromaticity coordinates of the source pictures, as defined in Table 7-8.

Table 7-8. Colour primaries

Value	Colour primaries
0	Forbidden
1	Recommendation ITU-R BT.709 primary x y green 0.300 0.600 blue 0.150 0.060 red 0.640 0.330 white D65 0.3127 0.3290
2	Unspecified video Image characteristics are unknown
3	Reserved

4	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	Recommendation ITU-R BT.470-2 System B, G primary x y green 0.29 0.60 blue 0.15 0.06 red 0.64 0.33 white D65 0.313 0.329
6	SMPTE 170M primary x y green 0.310 0.595 blue 0.155 0.070 red 0.630 0.340 white D65 0.3127 0.3290
7	SMPTE 240M (1987) primary x y green 0.310 0.595 blue 0.155 0.070 red 0.630 0.340 white D65 0.3127 0.3291
8	Normal film (colour filter, C light) primary x y green 0.243 0.692 (Wratten 58) blue 0.145 0.049 (Wratten 47) red 0.681 0.319 (Wratten 25)
9 - 255	Reserved

If the sequence display extension does not appear in bitstream, or the value of **colour_description** is zero, then the chrominance is assumed to have been connotatively defined by applications themselves.

Opto-electronic transfer characteristic

An 8-bit unsigned integer, which identifies opto-electronic transfer characteristics of the source picture. See table 7-9.

Table 7-9. Opto-electronic transfer characteristic

Value	Opto-electronic transfer characteristic
0	Forbidden
1	Recommendation ITU-R BT.709 $V = 1.099\ L_c^{0.45} - 0.099$ ，$1 >= L_c >= 0.018$ $V = 4.500\ L_c$ ，$0.018 > L_c >= 0$
2	Not specified video Picture characters unknown
3	Reserved
4	Recommendation ITU-R BT.470-2 System M Suppose display gamma is 2.2
5	Recommendation ITU-R BT.470-2 System B, G Suppose display gamma is 2.8

6	SMPTE 170M $V = 1.099\ L_c^{0.45} - 0.099$ ，$1 >= L_c >= 0.018$ $V = 4.500\ L_c$ ，$0.018 > L_c >= 0$
7	SMPTE 240M (1987) $V = 1.1115\ L_c^{0.45} - 0.1115$ ，$L_c >= 0.0228$ $V = 4.0\ L_c$ ，$0.0228 > L_c$
8	Linear transfer characteristic, i.e. $V = L_c$
9	Logarithmic transfer characteristic (range 100:1) V = 1.0-Log10(Lc)/2 ，1= Lc = 0.01 V= 0.0 ，0.01> Lc
10	Logarithmic transfer characteristic (range 316.22777:1) V = 1.0-Log10(Lc)/2.5 ，1= Lc = 0.0031622777 V= 0.0 ，0.0031622777> Lc
11 – 255	Reserved

If the sequence display extension does not appear in bitstream, or the value of **colour_description** is zero, then the transfer characteristic is assumed to have been connotatively defined by applications themselves.

Colour signal transfer matrix – matrix_coeffcients

An 8-bit unsigned integer, which identifies the adopted matrix used in the transfer from the red, green and blue primaries to luminance and chrominance signals. See table 7-10.

Table 7-10. Colour signal transfer characteristic

matrix_coefficients	Colour signal transfer characteristic
0	Forbidden
1	Recommendation ITU-R BT.709 $E'_Y = 0.7154\ E'_G + 0.0721\ E'_B + 0.2125\ E'_R$ $E'_{PB} = -0.386\ E'_G + 0.500\ E'_B - 0.115\ E'_R$ $E'_{PR} = -0.454\ E'_G - 0.046\ E'_B + 0.500\ E'_R$
2	Not specified video Picture characters unknown
3	Reserved
4	FCC $E'_Y = 0.59\ E'_G + 0.11\ E'_B + 0.30\ E'_R$ $E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{PR} = -0.421\ E'_G - 0.079\ E'_B + 0.500\ E'_R$
5	Recommendation ITU-R BT.470-2 System B, G $E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$ $E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{PR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$
6	SMPTE 170M $E'_Y = 0.587\ E'_G + 0.114\ E'_B + 0.299\ E'_R$ $E'_{PB} = -0.331\ E'_G + 0.500\ E'_B - 0.169\ E'_R$ $E'_{PR} = -0.419\ E'_G - 0.081\ E'_B + 0.500\ E'_R$
7	SMPTE 240M (1987) $E'_Y = 0.701\ E'_G + 0.087\ E'_B + 0.212\ E'_R$ $E'_{PB} = -0.384\ E'_G + 0.500\ E'_B - 0.116\ E'_R$ $E'_{PR} = -0.445\ E'_G - 0.055\ E'_B + 0.500\ E'_R$
8 - 255	Reserved

In table 7-10

- E_Y is the analogical variable between 0 and 1.

- E_{PB} and E_{PR} are analogical variables between -0.5 and 0.5.

- E_R, E_G and E_B are analogical variables between 0 and 1.

- The relation between Y, Cb and Cr is:

  If video_range is “0”:

  $$Y = (219 \times 2^{n-8} \times E'_Y) + 2^{n-4}$$

  $$Cb = (224 \times 2^{n-8} \times E'_{PB}) + 2^{n-1}$$

  $$Cr = (224 \times 2^{n-8} \times E'_{PR}) + 2^{n-1}$$

  If video_range is “1”:

  $$Y = ((2^n - 1) \times E'_Y)$$

  $$Cb = ((2^n - 1) \times E'_{PB}) + 2^{n-1}$$

  $$Cr = ((2^n - 1) \times E'_{PR}) + 2^{n-1}$$

  In which n is sample decision, for instance, when n=8 and video_range is 0, we get

  $$Y = (219 \times E'_Y) + 16$$

  $$Cb = (224 \times E'_{PB}) + 128$$

  $$Cr = (224 \times E'_{PR}) + 128$$

  Y lies in the range 16 to 235, Cb and Cr are in range 16 to 240.
  When n=8 and video_range is “1”, we get

  $$Y = (255 \times E'_Y)$$

  $$Cb = (255 \times E'_{PB}) + 128$$

  $$Cr = (255 \times E'_{PR}) + 128$$

  Y, Cb and Cr are all in range 0 to 255.

Note: The decoding process in this specification limits the output of Y, Cb and Cr samples lie in the range of [0:255].

If the sequence display extension does not appear in bitstream, or the value of **colour_description** is zero, then the transfer characteristic is hypothesised to have been connotatively defined by applications themselves.

Note: There may be different kinds of video signals in some applications, and different video signals may have different three colour primaries, transfer characteristics and/or transfer matrics too. In such cases, applications are suggested to transfer these different parameter sets to a uniform parameter set firstly.

Display horizontal size - display_horizontal_size

Display vertical size - display_vertical_size

Display_horizontal_size and **display_vertical_size** are all 14-bit unsigned integers, which define a rectangle together. If their size are smaller than that of the encoded pictures, then only a part of the picture should be displayed; and if their size are greater than that of the encoded pictures, then only a part of the display facility should be used to reconstructed the picture.

The unit of **display_horizontal_size** and **horizontal_size** should be the sample number of each line of the encoded picture.

The unit of **display_vertical_size** and **vertical_size** should be the line number of the encoded picture.

Display_horizontal_size and **display_vertical_size** have no effect on decoding process, and they can be used by display process. This specification didn't define the display process.

7.2.2.4 Copyright extension

Video extension identifier – extension_id

A bit string with value "0100".

Copyright flag – copyright_flag

A flag, with value "1" for period of validity of copyright information defined by the copyright extension, will extend to the next copyright extension or the end code of video sequence; and with value "0" for no copyright information in the copyright extension.

Copyright information is further illustrated by **copyright_id** and **copyright_number**.

Copyright indentifier – copyright_id

An 8-bit unsigned integer. All codes of copyright possessory are distributed by copyright registration institution. Zero of this id means no copyright information.

If the copyright_identifier is zero, then the copyright_number should be zero.

If the copyright_flag is zero, then the copyright_identifier should be zero.

Original or copy – original_or_copy

A flag with value "1" for the content of source video being original; and with value "0" for the content of source video being a copy.

Copyright number – copyright_number

A 64-bit unsigned integer. If copyright_flag is 1, then the CopyrightNumber is corresponding to the illustrated video source content of the copyright extension. "0" of CopyrightNumber stands for no related information. If copyright_flag is 1, then the CopyrightNumber should be "0" too.

7.2.2.5 Camera parameter extension

Video extension identifier - extension_id

A bit string with value "1011".

Camera identifier – camera_id

Identifier of camera.

Height of image device - height_of_image_device

A 22-bit unsigned integer, which presents the height of image device with 0.001mm as the unit, ranging from 0 to 4,194.303mm.

Focal length – focal length

A 22-bit unsigned integer, which presents the focal length of camera with 0.001mm as the unit, ranging from 0 to 4,194.303mm.

f_number

A 22-bit unsigned integer which presents the f_number of camera (f_number = focal_length ÷ valid aperture of lens), with 0.001mm as the unit, ranging from 0 to 4,194.303mm.

Vertical angle of view – vertical_angle_of_view

A 22-bit unsigned integer which presents the vertical angle of view determinated by picture device top and bottom, with 0.001mm as the unit, ranging from 0 to 180 degree.

Camera_position_x, camera_position_y and camera_position_z

CameraPositionX ,CameraPositionY and CameraPositionZ are a group of 32-bit integers represented with complement code of 2, illustrating the coordinate values of camera optics origin in global coordinate system defined by user. Each coordinate value ranges from -2,147,483.648mm to 2,147,483.647mm with 0.001mm as the unit.

Camera_direction_x, camera_direction_y and camera_direction_z

A group of 22-bit integer represented with complement code of 2, illustrating the camera direction. Each value ranges from -2,097,152 to 2,097,151. The camera direction is showed with a vector, from the camera optics origin to some point on optical axes before camera.

Image_plane_vertical_x, image_plane_vertical_y, image_plane_vertical_z

A group of 22-bit integer represented with complement code of 2, illustrating the camera upward direction. Each value ranges from -2,097,152 to 2,097,151. The camera upward direction is showed with a vector whose direction is from bottom to top, and parallels to bounday of the device.

Figure 7-1 and figure 7-2 give more details on this section.

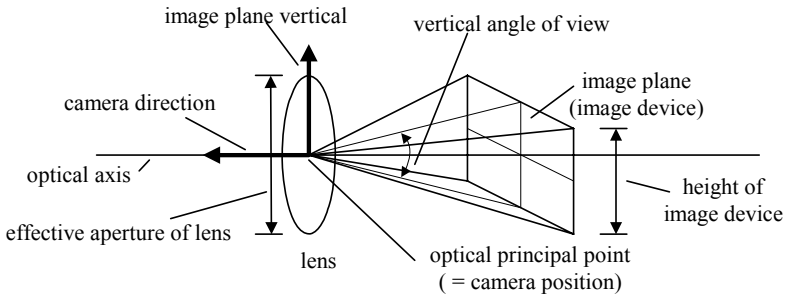


Figure 7-1.

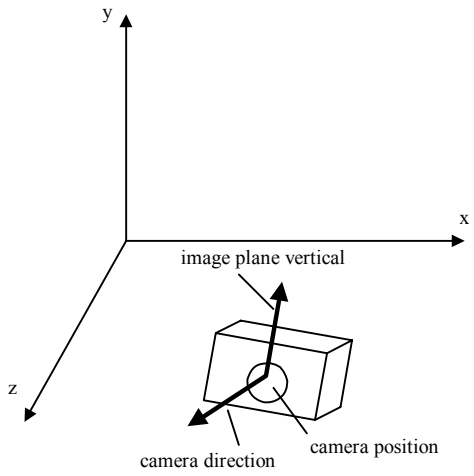


Figure 7-2.

7.2.3 Picture

7.2.3.1 I picture header

I picture start code - i_picture_start_code

A bit string with value “0x000001B3”, standing for the start code of an I frame.

Time code flag – time_code_flag

A flag representing the different meanings of time_code.

Time code – time_code

If the value of **time_code_flag** is “1”, then **time_code** is a 24-bit unsigned integer, including the following fields: DropFrameFlag, TimeCodeHours , TimeCodeMinutes , TimeCodeSeconds and TimeCodePictures, as defined in table 7-11. These parameters are unsigned integers indicated with inverse codes. These parameters are corresponding to that defined in “Time and control coding of video tape recorder” of IEC461 specification. **Time_code** describes the time of the first frame whose **picture_distance** is 0 from the current frame.

Table 7-21 Time code

Fields in time_code	Value	Descriptor
DropFrameFlag		u(1)
TimeCodeHours	0..23	u(5)
TimeCodeMinutes	0..59	u(6)
TimeCodeSeconds	0..59	u(6)
TimeCodePictures	0..59	u(6)

Picture distance - picture_distance

A 8-bit unsigned integer, which equals to **picture_distance** of former frame (in display order) with 1 pluse and 256 mode.

BBV dealy – bbv_delay

A 16-bit unsigned integer. When **bbv_dealy** does not equal to 0xFFFF, then it specifies the the waiting time from receiving the last byte of picture start code to the beginning of picture decoding. This time is represented with the clock period of 90 KHz derived from system clock of 27 MHz. If a certain picture's **bbv-delay** value equals to 0xFFFF, then all pictures in the video sequence must have **bbv-delay** value of 0xFFFF. See more details on **bbv_delay** in Appendix D.

Progressive frame – progressive_frame

If **progressive_frame** is set to 0, it indicates that the two fields of the frame are interlaced fields in which an field time interval exists between them. In this case **repeat_first_field** can be zero.

If **progressive_frame** is set to 1, then it indicates that the two fields actually come from the same time. In this case, the value of **picture_structure** can be 1.

Picture structure - picture_structure

A flag with "0" for the coding data of two fields in current picture appears in turn; and with "1" for appears alternatively. The value of PictureStructure is **picture_structure**.

Top field first - top_field_first

A flag whose meaning is determined by **progressive_sequence**, **progressive_frame** and **picture_structurrepeat_first_field**.

- If the value of **progressive_sequence** is 0, then **top_field_first** indicates the order of decoding processing output field.

- If the value of PicutreStructure is 0, and the value of **top_field_first** is 1, then decoding process firstly decodes and output top field, then decodes and outputs bottom field; if the value of **top_field_first** is 0, then decoding process firstly decodes and outputs bottom field, then decodes and output top field.
- If the value of PicutreStructure is 1, then decoding process firstly decodes the whole frame. If the value of **top_field_first** is 1, then firstly outputs top field, then outputs bottom field; if the value of **top_field_first** is 0, then firstly outputs bottomfield, then outputs top field.

- If the value of PicutreStructure is 1, then **top_field_first** and **repeat_first_field** explain the times of output frames in decoding process (one, two or three times).

- If the value of **repeat_first_field** is 0，and that of **top_field_first is** 0 too, then the decoding process outputs a frame.
- If the value of **top_field_first** is 0，and that of **repeat_first_field** is 1 too, then the decoding process outputs two totally different frames.
- If the values of **top_field_first** and **repeat_first_field** are both1, then the decoding process outputs three
  totally different frames.

Repeat first field – repeat_first_field

A flag works only when the value of PictureStructure is 1. If the value of PictureStructure is 0, then the value of **repeat_first_field** should be 0 too, and has no effect on the decoding process.

- If the values of **progressive_sequence** and **progressive_frame** are both 0, then the value of **repeat_first_field** should be 0 too. The decoding process will output two fields.

- If the value of **progressive_sequence** is 0, and the value of **progressive_frame** is 1, then

• if the value of **repeat_first_field** is 0, then the decoding process will output two fields, and the second field follows the first one (top field or bottom field, determinated by **top_field_first**);
• if the value of **repeat_first_field** is 1, then the decoding process will output three fields, and the second field follows the first one (top field or bottom field, determinated by **top_field_first**).

- If the value of **progressive_frame** is 0, then

• if the value of **repeat_first_field** is 0, then decoding process outputs one field;
• if the value of **repeat_first_field** is 1, then decoding process outputs two or three fields, which is determined by **top_field_first**.

Fixed picture quantised parameter – fixed_picture_qp

A flag, with value "1" for no change of quantised parameter in the frame, and with value "0" for change of quantised parameter is allowed in the frame.

Picture quantised parameter - picture_qp

A 6-bit unsigned integer, which presents the quantised **parameter** of picture. quantised **parameter** is limited to the range [0: 63].

Skip mode flag - skip_mode_flag

A flag with value "1" for skipping mode based on run length coding, and with value "0" for skipping mode based on fixed length coding.

Loop filter disable flag - loop_filter_disable

A flag with value "1" for not using loop filter, and with value "0" for using loop filter. This flag can only appear in the picture header of an I frame and a P frame, while a B frame can not use loop filter.

Loop filter parameter flag - loop_filter_parameter_flag

A flag with value "1" for including picture header in loop filter parameter; otherwise the default values AlphaCOffset and BetaOffset which equal to 0 are used.

Offsets of α and C index- alpha_c_offset

Indicting the offsets of loop filter α and C index in picture, **alpha_c_offset** lies in the range -8..8, and AlphaCOffset equals to **alpha_c_offset**。

Offset of β index - beta_offset

Indicting the offsets of loop filter **β** index, **beta_offset** lies in the range -8..8, and BetaCOffset equals to **beta_offset**。

7.2.3.2 PB picture header

PB picture start code - pb_picture_start_code

A bit serie equals to "0x000001B6", which represents the start code of a P frame or a B frame.

Advanced predictive mode disable flag - advanced_pred_mode_disable

A flag with value "1" for the advanced pridictive mode is disabled; and with "0" for it can be reserved.

Picture coding type - picture_coding_type

A 2-bit unsigned integer which specifies the coding tyoe of picture. See table 7-12.

Table 7-32 Picture coding type

picture_coding_type	Coding type
00	Forbidden
01	Forward prediction coding（P）
10	Backward prediction coding（B）
11	Reserved

Picture reference flag - picture_reference_flag

A flag with value “1” for all macroblocks can use the default reference pictures; and with value “0” for each macroblock can confirm the reference picture itself. See the confirmation method in section 9.4.3

See other syntax elements of PB picture header in section 7.2.3.1.

7.2.3.3 Picture display extension

The specification does not define the display process, and information in the extension which may be ignored by decoders has no effect on the decoding process.

Picture display extension permits rectangle (whose size is defined by sequence display extension) move according to pictures. One of the applications is to implement panorama scan.

Video extension identifier – extension_id

A bit string with value “0111”.

Frame center horizontal offset - frame_centre_horizontal_offset

A 16-bit integer, which presents the horizontal offset with 1/16 sample as the unit. Positive value indicates that the center of reconstructed picture is on the right side of display rectangle center.

Frame center vertical offset - frame_centre_vertical_offset

A 16-bit integer, which presents the vertical offset with 1/16 sample as the unit. Positive value indicates that the center of reconstructed picture is on the bottom of display rectangle center.

The size of display rectangle region is defined in sequence display extension. And region coordinates in encoded picture are defined by picture display extension.

The center point of reconstructed picture is the center of rectangle defined by **horizontal_size** and **vertical_size**.

In interlaced sequences, a encoded picture may related to one, two or three decoding fields, so picture display extension can define three offset variables at most.

The value of NumberOfFrameCentreOffsets in section 7.1.3.3 is defined as follows:

```
if ( progressive_sequence == '1' ) {
    if ( repeat_first_field == '1' ) {
        if ( top_field_first == '1' )
            NumberOfFrameCentreOffsets = 3
        else
            NumberOfFrameCentreOffsets = 2
    } else {
        NumberOfFrameCentreOffsets = 1
    }
} else {
```

```
    if ( picture_structure == '01' || picture_structure == '10' ) {
        NumberOfFrameCentreOffsets = 1
    } else {
        if ( repeat_first_field == '1' )
            NumberOfFrameCentreOffsets = 3
        else
            NumberOfFrameCentreOffsets = 2
    }
}
```

If there is no sequence display extension after the former sequence header, then the picture display exension should not appear in the bitstream.

If a picture has no display exension, then it uses center offset of the nearest decoding picture. Note that the center offset of lost frames are the same. All center offsets of the pictures are set to zero after the sequence header, until the appearance of the picture display extension.

A rectangle region may be defined based on the picture center offset, which shakes in the whole scope of reconstructed picture to achieve panorama scan.

The picture center offset parameter is described in figure 7-3.

Note 1: the size of the display rectangle may be greater than that of reconstructed picture.

Note 2: in a field picture, **frame_centre_vertical_offset** represents the center offset with a unit of 1/16 frame line.

Note 3: in figure7-3, the value of **frame_centre_horizontal_offset** and **frame_centre_vertical_offset** are both negative.

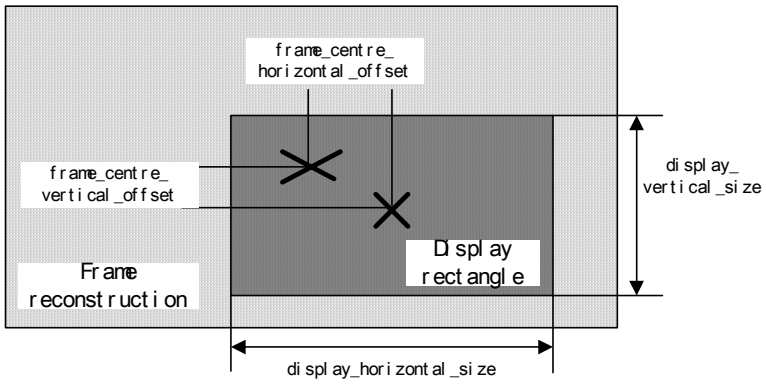


Figure 7-3. Frame center offset parameter

7.2.4 Slice

slice_start_code - A 32-bit string with first 24 bits being “0x000001”, and the last 8 bits indicate slice_vertical_position with a value lie in the range 0x00 through 0xAF.

slice_vertical_position - An 8-bit unsigned integer, which presents the vertical position of the first macroblock of slice in picture, with macroblock as the unit.

fixed_slice_qp - A flag with value "1" for no change of quantised parameter in the slice; and with value "0" for change of quantised parameter is allowed in the slice.

slice_qp - A 6-bit unsigned integer, which presents the quantised **parameter** in slice. Quantised **parameter** should lie in the range 0 to 63.

mb_skip_run - The number of macroblocks skipped. Parsing process is detailed in section 8.2.

7.2.5 Macroblock

Macroblock type – mb_type

Macroblock type with its semantic determined by picture type, PictureStructure and **skip_mode**. See the parsing process in section 8.2, and decoding process in section 9.5.

Macroblock part type – mb_part_type

A 2-bit unsigned integer, which represents the part type of macroblock, with its semantic determined by MbType. See decoding process in section 9.5.

Predictive mode flag – pred_mode_flag

A flag with value "1" for using intra-frame luminance predictive mode; with value "0" for identifying intra-frame luminance predictive mode based on **intra_luma_pred_mode**.

Intra-frame luminance predictive mode – intra_luma_pred_mode

A 3-bit unsigned integer used to identify the intra-frame predictive mode of the luminance block. See the decoding process in section 9.4.2.

Intra-frame chrominance predictive mode – intra_chroma_pred_mode

A flag used for identifying the intra-frame predictive mode of the chrominance blocks, see the decoding process in section 9.4.2.

Macroblock reference index – mb_reference_index

If the value of PictureStructure is 1 or the value of PictureType is 2, then **mb_reference_index** is a 1-bit unsigned integer. If the value of PictureStructure is 0 or the value of PictureType is 1, then **mb_reference_index** is a 2-bit unsigned integer. See section 9.4.3 for more details.

Motion vector horizontal component difference – mv_diff_x

Motion vector vertical component difference – mv_diff_y

Motion vector difference, see decoding process in section 9.4.4.

Macroblock coding template – cbp

If **cbp** does not exist, and PictureType is 1, then **cbp** equals to subtracting 5 from MbTypeIndex; if **cbp** does not exist, and PictureType is 2, then **cbp** equals to subtracting 24 from MbTypeIndex. A 6-bit unsigned MbCBP can be obtained after parsing **cbp**, which indicates whether the 4 8×8 luminance blocks and 2 8×8 chrominance blocks include non-zero transform coefficients, in which the higher 2 bits are Cbpc, and the lower 4 bits are Cbpy; the n-th bit of Cbpy and Cbpc (in binary form) being zero indicates there are no none-zero coefficients in corresponding 8×8 block, being 1 indicates there exists one or more non-zero coefficients in the corresponding 8×8 block. See block sequence in section 6.6.

Macroblock quantised parameter increment - mb_qp_delta

Which presents the increment of quantised paramenter of the current block relative to the preditive quantised paramenter. It lies in the range -32 to 31.

7.2.6 Block

Transform coefficient - trans_coefficient

A combined index used to identify the run length and the quantised non-zero coefficients, see the decoding process in section 9.5.1.

Escape coefficient – escape_level

When **trans_coefficient** can not identify the combined index of the run length and the quantised non-zero coefficients, **escape_level** is used to identify the quantised non-zero coefficients, see the decoding process in section 9.5.1.

Escape run length – escape_run

When **trans_coefficient** can not identify the combined index of run length and quantised non-zero coefficients, **escape_run** is used to identify the run length, see the decoding process in section 9.5.1.

8 Parsing process

This chapter defines the parsing process of syntax elements. See the parsing processes of syntax element discriptor, ue(v), se(v), me(v) and te(v), in section 8.2. See the parsing process of ce(v) in section 8.3.

8.1 K-rank exponential Columbus code

When parsing K-rank exponential Columbus code, above all, seeks the first non-zero bit from the current position of bitstream, and labels the number of zero bits found with leadingZeroBits, then CodeNum is calculated with leadingZeroBits. The description with pseudo code is:

```
leadingZeroBits = -1;
for ( b = 0; ! b; leadingZeroBits++ )
    b = read_bits(1)
codeNum = 2^(leadingZeroBits + k) – 2^k + read_bits( leadingZeroBits + k)
```

Table 8-1 presents the structures of 0-rank, 1-rank, 2-rank and 3-rank exponential Columbus codes. The bit string of exponential Columbus code include two parts of “prefix” and “suffix”. Prefix is composed of one “1” and leadingZeroBits sucessive “0”. Suffix is composed of leadingZeroBits + k bits, namely string x_i in the table. The value of x_i can be “0” or “1”.

Table 8-1 k ranks exponential Columbus code table

Ranks	Code word structure	Range of CodeNum value
k = 0	1	0
	0 1 x_0	1-2
	0 0 1 x_1 x_0	3-6
	0 0 0 1 x_2 x_1 x_0	7-14
k = 1	1 x_0	0-1
	0 1 x_1 x_0	2-5
	0 0 1 x_2 x_1 x_0	6-13
	0 0 0 1 x_3 x_2 x_1 x_0	14-29
k = 2	1 x_1 x_0	0-3
	0 1 x_2 x_1 x_0	4-11

	0 0 1 x_3 x_2 x_1 x_0	12-27
	0 0 0 1 x_4 x_3 x_2 x_1 x_0	28-59
k = 3	1 x_2 x_1 x_0	0-7
	0 1 x_3 x_2 x_1 x_0	8-23
	0 0 1 x_4 x_3 x_2 x_1 x_0	24-55
	0 0 0 1 x_5 x_4 x_3 x_2 x_1 x_0	56-119

8.2 ue(v), se(v) and me(v)

Syntax elements described by ue(v), se(v), me(v) and te(v) use 0-rank Columbus code, and the parsing process is as follows:

- ue(v): the value of syntax element equals to CodeNum;
- se(v): calculate the value of syntax elemets, based on mapping relationship of signed exponential Columbus codes presented in table 8-2;
- me(v): calculate the values of syntax elements **cbp** based on table 8-3.

Table 8-2 Mapping relationship between se(v) and CodeNum

CodeNum	**Syntax element value**
0	0
1	1
2	–1
3	2
4	–2
5	3
6	–3
k	(–1)k+1 Ceil(k÷2)

Table 8-3 Mapping relationship between cbp and CodeNum

CodeNum	**cbp** xxxxxx (543210)	
	Intra-frame coding mode	**Inter-frame coding mode**
0	63	0
1	15	15
2	31	63
3	47	31
4	0	16
5	14	32
6	13	47

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

43	32	37
44	58	39
45	24	45
46	20	58
47	17	43
48	18	42
49	48	46
50	22	36
51	33	33
52	25	34
53	49	40
54	40	52
55	36	49
56	34	50
57	50	56
58	52	25
59	54	22
60	41	54
61	56	57
62	38	41
63	57	38

8.3 ce(v)

The syntax elements described by ce(v) adopt 0-rank, 1-rank, 2-rank and 3-rank exponential Columbus code to parse, and rank level is determined by the following rules:

- **escape_level** of intra-coding block luminanace coefficients adopts 2-rank exponential Columbus code, and **escape_run** adopt 1-rank exponential Columbus code.
- **escape_level** of inter-coding block luminanace coefficients adopts 1-rank exponential Columbus code, and **escape_run** adopt 2-rank exponential Columbus code.
- **escape_level** of chrominance block adopts 0-rank exponential Columbus code, and **escape_run** adopt 3-rank exponential Columbus code.
- The specification defines 19 variable length code tables related to ce(v), including VLC0_Intra, VLC1_Intra, VLC2_Intra, VLC3_Intra, VLC4_Intra, VLC5_Intra, VLC6_Intra, VLC0_Inter, VLC1_Inter, VLC2_Inter, VLC3_Inter, VLC4_Inter, VLC5_Inter, VLC6_Inter, VLC0_Chorma, VLC1_Chorma, VLC2_Chorma, VLC3_Chorma and VLC4_Chorma, which are detailed in Appendix A. Different code tables identify the rank levels of exponential Columbus code adopted by

ce(v). In above code tables, VLC0_Inter utilizes 3-rank exponential Columbus code, VLC2_Chroma and VLC3_Chroma utilize 1-rank exponential Columbus code, VLC1_Chroma and VLC4_Chroma utilize 0-rank exponential Columbus code, the other code tables utilize 2-rank exponential Columbus code.

Code tables are selected in term of the following rules:

- for the first quantised coefficient,
  - the luminance coefficients of intra-predictive encoded block select CurrentVLCTable = VLC0_Intra, see table A-1;
  - the luminance coefficients of inter-predictive encoded block select CurrentVLCTable = VLC0_Inter, see table A-8;
  - the chrominance coefficients choose CurrentVLCTable = VLC0_Chroma, see table A-15.
  - maxAbsLevel equals to 0;
  - absLevele quals to the absolute value of the first quantised coefficient.
- Code table choose of other quantised coefficients,
  - for the luminance coefficients of intra-predictive encoded block, if absLevel is greater than maxAbsLevel, then the code table is switches in terms of the following rules:
    - if absLevel equals to 1，then CurrentVLCTable = VLC1_Intra，see table A-2;
    - if absLevel equals to 2，then CurrentVLCTable = VLC2_Intra，see table A-3;
    - if absLevel equals to 3 or 4，then CurrentVLCTable = VLC3_Intra，see table A-4;
    - if absLevel equals to 5, 6 or 7，then CurrentVLCTable = VLC4_Intra，see table A-5;
    - if absLevel equals to 8, 9 or 10，then CurrentVLCTable = VLC5_Intra，see table A-6;
    - if absLevel is greater than 1，then CurrentVLCTable = VLC6_Intra，see table A-7;
  - for the luminance coefficients of inter-predictive encoded block, if absLevel is greater than maxAbsLevel, then the code table is switches in terms of the following rules:
    - if absLevel equals to 1，then CurrentVLCTable = VLC1_Inter，see table A-9;
    - if absLevel equals to 2，then CurrentVLCTable = VLC2_Inter，see table A-10;
    - if absLevel equals to 3，then CurrentVLCTable = VLC3_Inter，see table A-11;
    - if absLevel equals to 4, 5 or 6，then CurrentVLCTable = VLC4_Inter，see table A-12;
    - if absLevel equals to 7, 8 or 9，then CurrentVLCTable = VLC5_Inter，see table A-13;

- if absLevel is greater than 9，then CurrentVLCTable = VLC6_Inter，see table A-14;

- for the chrominance coefficients, if absLevel is greater than maxAbsLevel, then code table is switches in terms of the following rules:
  - if absLevel equals to 1，then CurrentVLCTable = VLC1_Chroma，see table A-16;
  - if absLevel equals to 2，then CurrentVLCTable = VLC2_ Chroma，see table A-17;
  - if absLevel equals to 3 or 4，then CurrentVLCTable = VLC3_ Chroma，see table A-18;
  - if absLevel is greater than 4，then CurrentVLCTable = VLC4_ Chroma，see table A-19;
- maxAbsLevel equals to absLevel;
- absLevele quals to the absolute value of the first quantised coefficient.

The parsing process of syntax elements described by ce(v) is:

- if CodeNum is not equal to 59, then syntax element **trans_coefficient is** equal to CodeNum;

- if CodeNum is equal to 59, then
  - parsing the next syntax element ce(v) to get a new CodeNum, **escape_level** is equal to CodeNum;
  - parsing the following next syntax element ce(v) to get a new CodeNum, **escape_run** is equal to CodeNum.

9 Decoding process

This chapter defines the decoding process.

9.1 Higher syntactic structures

The various parameters and flags in the bitstream for macroblock and all syntactic structures above macroblock can be interpreted as indicated in chapter 7. Many parameters and flags affect the decoding process described in the following clauses.

If **progressive_sequence** is equal to 1, the reconstructed frames can be outputted by decoding process at intervals of the frame period.

If **progressive_sequence** is equal to 0, the reconstructed frames can be broken into two fields which may be outputted by decoding process at intervals of the field period. If the values **of repeat_first_field** and PictureStructure of the reconstructed picture are both 1, then the first field of the frame can be repeated after the second field.

9.2 Picture header decoding

The decoding process is described as follows:

If the start code of the current picture is ‘0x00001B3’, then that picture is an I frame, and PictureType equals to zero.

If the start code of the current picture is ‘0x00001B6’, and **picture_code_type equals to** ’01’, then that picture is an I frame, and PictureType equals to 1.

If the start code of the current picture is ‘0x00001B6’, and **picture_code_type equals to** ’10’, then that picture is a B frame, and PictureType equals to 2.

If current picture is an an I frame, or a B frame and PictureStructure equals to 1, then PictureRefFlag equals to 0; otherwise PictureRefFlag equals to **picture_reference_flag**.

The quantised parameter PreviousQP of slice is initialized with **picture_qp**. Fixed quantised parameter flag FixedQP equals to **fixed_picture_qp**. The macroblock index MbIndex of the current block is initialized with 0.

9.3 Slice decoding

The decoding process is described as follows:

If the slice start code exists, then the macroblock index MbIndex equals to **slice_vertical_position** × 16.

If **fixed_picture_qp** is equals to zero, then initial quantised parameter of the slice equals to **slice_qp**. Fixed quantised parameter flag FixedQP equals to **fixed_slice_qp**.

If either of the following conditions is fulfiled, then the macroblock index MbIndex can be added with **mb_skip_run**

- the current picture is a P or a B frame;
- the current frame is an I frame, then PictureStructure equals to 0 and MbIndex >= MbWidth × MbHeight / 2.

9.4 Macroblock decoding

9.4.1 Macroblock type

The decoding process of macroblock type MbType and macroblock part type MbPartType is:

- If current picture is an I frame
  - if the value of PictureStructure is 1, then MbType equals to ‘I_8x8’, and MvNum equals to 0;
  - if the value of PictureStructure is 0 and MbIndex < MbWidth × MbHeight / 2, then MbType equals to ‘I_8x8’, and MvNum equals to 0;
  - if the value of PictureStructure is 0 and MbIndex >= MbWidth × MbHeight / 2, then
    - If the value of **skip_mode_flag** is 0, then MbTypeIndex equals to **mb_type** pluses 1; otherwise MbTypeIndex equals to **mb_type**. See the values of MbType and MvNum in table 9-1.
- If current picture is a P frame
  - if the value of **skip_mode_flag** is 0, then MbTypeIndex equals to **mb_type** pluses 1; otherwise MbTypeIndex equals to **mb_type**. See the values of MbType and MvNum in table 9-1.
- If current picture is a B frame

- If the value of **skip_mode_flag** is 0, then MbTypeIndex equals to **mb_type** pluses 1; otherwise MbTypeIndex equals to **mb_type**. See the values of MbType and MvNum in table 9-2. If MbType equals to 'B_8x8', then the values of MbPartType and MbPartMvNum are as illustrated in table 9-3. MvNum is the sum of MbPartMvNum of all blocks in the macroblock.

Table 9-1 P macroblock type

MbTypeIndex	MbType	MvNum	MbPredMode
0	P_Skip	0	Forward
1	P_16x16	1	Forward
2	P_16x8	2	Forward
3	P_8x16	2	Forward
4	P_8x8	4	Forward
5	I_8x8	0	None

Table 9-2 B macroblock type

MbTypeIndex	MbType	MvNum	MbPredMode
0	B_Skip	0	Bidirectional
1	B_Direct_16x16	0	Bidirectional
2	B_Fwd_16x16	1	Forward
3	B_Bck_16x16	1	Backward
4	B_Sym_16x16	1	Bidirectional
5	B_Fwd_Fwd_16x8	2	Forward
6	B_Fwd_Fwd_8x16	2	Forward
7	B_Bck_Bck_16x8	2	Backward
8	B_Bck_Bck_8x16	2	Backward
9	B_Fwd_Bck_16x8	2	Bidirectional
10	B_Fwd_Bck_8x16	2	Bidirectional
11	B_Bck_Fwd_16x8	2	Bidirectional
12	B_Bck_Fwd_8x16	2	Bidirectional
13	B_Fwd_Sym_16x8	2	Bidirectional
14	B_Fwd_Sym_8x16	2	Bidirectional
15	B_Bck_Sym_16x8	2	Bidirectional
16	B_Bck_Sym_8x16	2	Bidirectional
17	B_Sym_Fwd_16x8	2	Bidirectional
18	B_Sym_Fwd_8x16	2	Bidirectional
19	B_Sym_Bck_16x8	2	Bidirectional
20	B_Sym_Bck_8x16	2	Bidirectional
21	B_Sym_Sym_16x8	2	Bidirectional
22	B_Sym_Sym_8x16	2	Bidirectional
23	B_8x8	0..4	Forward, backward , bidirectional
24	I_8x8	0	None

Table 9-3 B_8x8 part mode

mb_part_type	MbPartType	MbPartMvNum	MbPartPredMode
0	SB_Direct_8x8	0	Bidirectional
1	SB_Fwd_8x8	1	Forward
2	SB_Bck_8x8	1	Backward
3	SB_Sym_8x8	1	Bidirectional

9.4.2 Intra-frame predictive direction

The 6 blocks of current macroblock use the following method to identify predictive direction:

- if current block is luminance block
  - calculate predictive mode of current block
    - if left block "exists" and it is an intra-frame predictive block, then transfers the intra-frame predictive mode of left block to intraPredModeA; otherwise intraPredModeA equals to 0;
    - if top block "exists" and it is an intra-frame predictive block, then transfers the intra-frame predictive mode of top block to intraPredModeB; otherwise intraPredModeB equals to 0;
    - predictive direction estimation of current block:
      predIntraPredMode = Min(intraPredModeA, intraPredModeB);
  - if the value of **pred_mode_flag** is 1, then predictive direction IntraLumaPredMode equals to predIntraPredMode;
  - if the value of **pred_mode_flag** is 0, and if **intra_luma_pred_mode** is less than predIntraPredMode , then IntraLumaPredMode equals to **intra_luma_pred_mode**; otherwise IntraLumaPredMode equals to **intra_luma_pred_mode** plus 1.

The existence of left block or top block means that block and the current block blong to the same slice; and they can also belong to the same field, if the two coding data fields of interlaced scan picture appear in turn.

See the values of IntraLumaPredMode and meanings of luminance block predictive mode in table 9-4.

Table 9-4 Names and serial numbers of 8×8 intra-frame predictive mode

Serial number	Names
0	Intra_8x8_DC
1	Intra_8x8_Vertical
2	Intra_8x8_Horizontal
3	Intra_8x8_Down_Right
4	Intra_8x8_Up_Right
5	Intra_8x8_Down_Right_Down
6	Intra_8x8_Down_Left_Down
7	Intra_8x8_Right_Up_Right
8	Intra_8x8_Right_Down_Right

- If the current block is chrominance block, then the predictive direction IntraChromaPredMode equals to **intra_chroma_pred_mode.**

See the values of IntraChromaPredMode and meanings of chrominance block predictive mode in table 9-5.

Table 9-5 Names and serial numbers of chrominance intra-frame predictive mode

Serial number	Names
0	Intra_Chroma_Vertical
1	Intra_Chroma_Horizontal
2	Intra_Chroma_DC
3	Intra_Chroma_Plane

See meanings of 8×8 intra-frame predictive direction in figure 9-1.

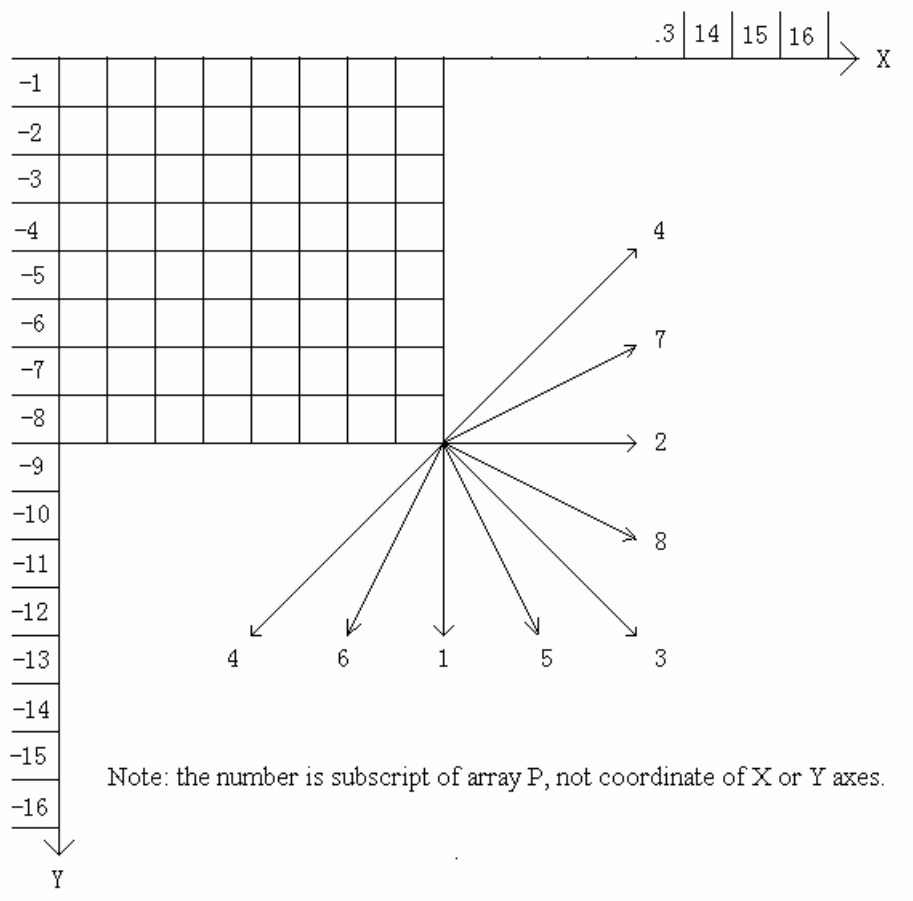


Figure 9-1 Variety directions of 8×8 intra-frame predictive mode

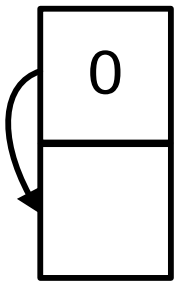
9.4.3 Selection of reference pictures

The reference pictures used in current picture can not exceed two, and they should be the nearest decoded I frame or P frames.

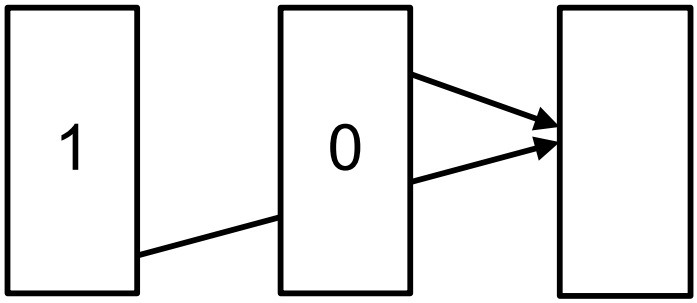
Reference index value is used to identify the reference pictures for the decoding process of the current picture, which lies in the range 0 to 3. Reference index value increases with the increase of distance between reference picture and current picture (in display order), a reference picture whose index value is 0 is the nearest

to current picture, a picture with index 1 is farer, and a picture with index 3 is the farest. The two fields of reference picture may have different reference index values, if the time interval of two fields in reference pictures is zero, then the index value of bottom field is less than that of the top fields. The identifying methods of refernece index value described as follows (numberss in the figure represents refernece index values, arrow pointed picture is the current picture):

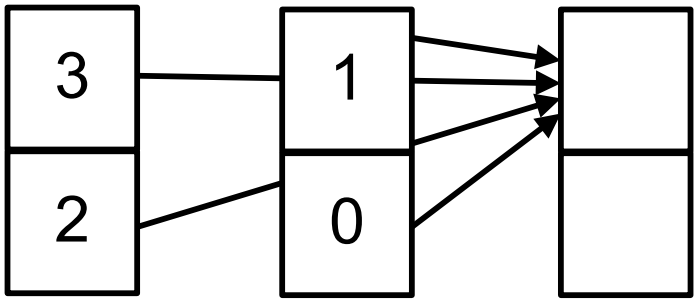
- if the current picture is an I frame and PictureStructure equals to 0, and current decoding field is the second field in display order, then see identifying method discribed in figure 9-2(a).
- if the current picture is a P frame and PictureStructure equals to 1, then see identifying method discribed in figure 9-2(b).
- if the current picture is a P frame and PictureStructure equals to 1, and the current decoding field is the first field in display order, then see identifying method discribed in figure 9-2(c).
- if the current picture is a P frame and PictureStructure equals to 0, and the current decoding field is the second field in the display order, then see identifying method discribed in figure 9-2(d).
- if the current picture is a B frame and PictureStructure equals to 1, then see the identifying method discribed in figure 9-2(e).
- if the current picture is a B frame and PictureStructure equals to 0, and the current decoding field is the first field in the display order, then see the identifying method discribed in figure 9-2(f).
- if the current picture is a B frame and PictureStructure equals to 0, and the current decoding field is the second field in display order, then see the identifying method discribed in figure 9-2(g).



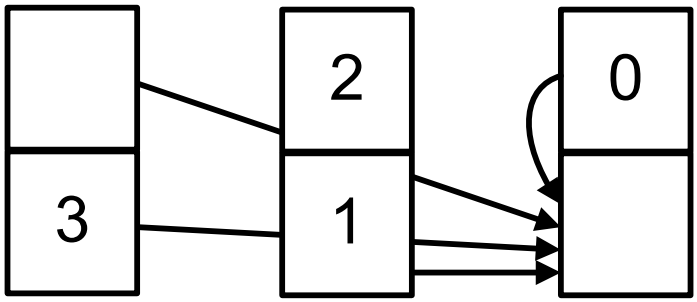
(a)



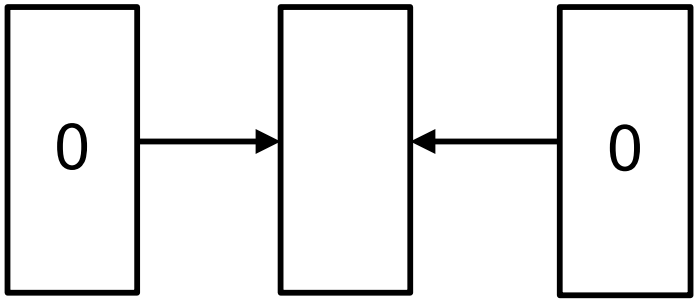
(b)



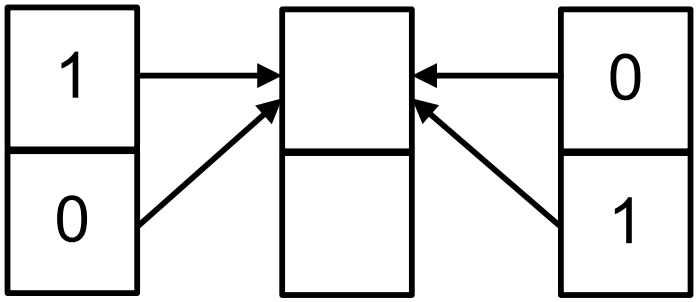
(c)



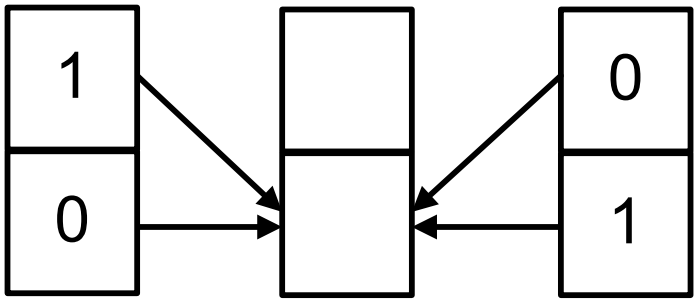
(d)



(e)



(f)



(g)

Figure 9-1 Identifying method of reference index

If the current picture is an I frame, or a B frame and PictureStructure equals to 1, or a P frame or a B frame and exists **picture_reference_flag** which equals to 1, then reference index can not appear in bitstream, default reference picture is that picture (or field)in figure 9-2 identified with 0. Otherwise the value of reference index equals to that of syntax element **mb_reference_index**.

9.4.4 Motion vector

The spatial positions between a luminance block E and its corresponding luminance blocks A, B, C and D are presented in figure 9-3. The size of E maybe 16×16, 16×8, 8×16 or 8×8. A, B and D are blocks tightly adjacent to the top left corner samples of E, and C is the block tightly adjacent to the top right corner samples of E. The DistanceIndex is defined as follows: if all pixels in block belong to the second field of interlaced scan picture (in display order), or belong to the bottom field progressive scan picture, then DistanceIndex equals to **picture_distance** multiplies 2 and pluses 1; otherwise DistanceIndex equals to **picture_distance** multiplies 2.

The distance BlockDistance between the current block (belonging to current picture) and its motion vector pointed reference block is calculated as follows:

- if the reference block is before current block (in display order), then BlockDistance equals to subtract DistanceIndex of the reference block from that of the current block, and the sum module 512 of addition with 512.
- if the reference block is after current block (in display order), then BlockDistance equals to subtract DistanceIndex of the current block from that of the reference block, and the sum module 512 of addition with 512.

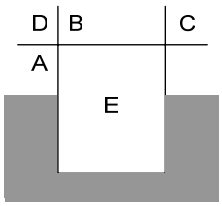


Figure 9-2 Spatial position relationship between luminance block E and adjacent luminance blocks

9.4.4.1 Luminance motion vector prediction

The original motion vectors of block A, B, C and D in figure 9-3 are mvA, mvB, mvC and mvD. Distance indexes of block A, B, C and D are DistanceIndexA, DistanceIndexB, DistanceIndexC and DistanceIndexD.

- if A is “not usable” or adopts intra-frame predictive model, then mvA is a zero vector, and DistanceIndexA equals to 1.
- if B is “not usable” or adopts intra-frame predictive model, then mvB is a zero vector, and DistanceIndexB equals to 1.
- if D is “not usable” or adopts intra-frame predictive model, then mvD is a zero vector, and DistanceIndexD equals to 1.
- if C is “not usable”, then mvC equals to mvD, and DistanceIndexA equals to DistanceIndexA.

Block “not usable” means that block belongs to the different slice with current block E, or when the two fields coding data of interlaced scan picture appear in turn, they don’t belong to the same field, or they don’t have motion vector with the same predictive direction; otherwise that block is “usable”.

The motion vector predictive value MVEPred of the current block E is calculated as follows:

Step 1, if one block of A, B and C is “usable”, then MVEPred euquals to mvX (X is A, B or C); otherwise moves to step 2.

Step 2, if E located macroblock is encoded with 16×8 or 8×16 mode, then the calculation process is (see figure 9-4):

- 8×16 mode:
  - E is the left block: if A is “usable” and is the same with the reference index of E, then MVEPredict=MVA; otherwise moves to step 3.
  - E is the right block: if C is “usable” and is the same with the reference index of E, then MVEPredict=MVC; otherwise moves to step 3.
- 16×8 mode:
  - E is the upper block: if B is “usable” and is the same with the reference index of E, then MVEPredict=MVB; otherwise moves to step 3.
  - E is the lower block: if A is “usable” and is the same with the reference index of E, then MVEPredict=MVA; otherwise moves to step 3.

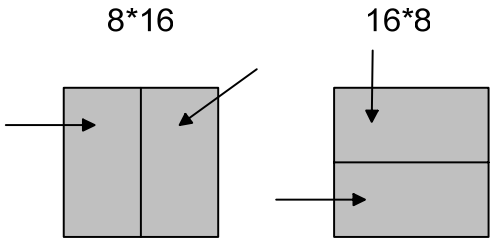


Figure 9-4 16×8 or 8×16 mode prediction

Step 3, firstly zooms mvA, mvB and mvC with their corresponding distance indexes DistanceIndexA, DistanceIndexB, DistanceIndexC and DistanceIndexE to obtain MVA, MVB and MVC, base on the following method:

MVA = (mvA × DistanceIndexE × (256 / DistanceIndexA) + 128) >> 8

MVB = (mvB × DistanceIndexE × (256 / DistanceIndexB) + 128) >> 8

MVC = (mvC × DistanceIndexE × (256 / DistanceIndexC) + 128) >> 8

Then calculates MVEPred. Defines distance Dist(MV1, MV2) = Abs(x1 – x2) + Abs(y1 – y2), in which motion vector MV1=[x1, y1] , MV2=[x2, y2]. Defines VA = Dist(MVA, MVB) + Dist(MVC, MVA), VB = Dist(MVB, MVC) + Dist(MVA, MVB) , and VC = Dist(MVB, MVC) + Dist(MVC, MVA). MVEPred is calculated as follows:

- calculates FWV which equals to Median(VA, VB, VC) ;
- otherwise, if FMV equals to VA, then MVEPred equals to MVA;
- otherwise, if FMV equals to VB, then MVEPred equals to MVB;
- otherwise, if FMV equals to VC, then MVEPred equals to MVC.

9.4.4.2 Luminance motion vector decoding

Motion vector mvE of the current block E equals to MVEPred plusing the sum of motion vector increments decoded from **mv_diff_x** and **mv_diff_y**.

9.4.5 Macroblock decoding template

Whether each block in the macroblock includes coding data is determined by **cbp**.

9.4.6 Quantised parameter

If **mb_qp_delta** exists, then quantised parameter of the current block CurrentQP equals to PreviousQP plusing **mb_qp_delta**; otherwise CurrentQP equals to PreviousQP.

9.5 Block decoding

9.5.1 Variable length code decoding

This section specifies the process of generating array level and array run of quantised coefficients. Array level includes the amplitude of non-zero quantised coefficients, and array run includes the number of

successive zero before the current non-zero quantised coefficients after block scan. The parsing process of generating syntax elements, needed in generating the quantised coefficients array, is detailed in section 8.3.

The rules of generating array run and array level are (initial value of array address is 0):

- if **trans_coefficient** is not equal to 59, then queries **trans_coefficient** indexed quantised coefficients and run in CurrentVLCTable, and puts them into array level and array run;
- if **trans_coefficient** is equal to 59, then
  - if **escape_level is even** , then put **escape_level** / 2 + 1 into array level; otherwise put -(**escape_level** / 2 + 1) into array level;
  - puts **escape_run into** array run;
- if **trans_coefficient** is equal to EOB, then ends block coefficients decoding; otherwise adds array address with 1, and updates CurrentVLCTable with methods defined in section 8.3, then decodes the next coefficient and run.

9.5.2 Inverse scan

Generating a array named with QuantCoeffArray which includes 64 quantised coefficients from the decoded array level and array run, with the following procedures:

- initializes array QuantCoeffArray with 0;
- transfers the values of non-zero quantised coefficients to the corresponding elements in QuantCoeffArray, then defines j and coeffNum, lets j equals to 0, and coeffNum equals to -1 ,

```
while ( not the last element of array run)
{
    coeffNum += ( Run[j] + 1 )
    QuantCoeffArray[63 - coeffNum] = Level[j]
    j++
}
```

- maps QuantCoeffArray to QuantCoeffMatrix by converse scan. See Converse scan methods in table 9-6, in which c_{ij} is the quantised coefficient in QuantCoeffMatrix.

Table 9-6 Reverse block scan

QuantCoeffArray index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
QuantCoeffMatrix coefficient	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{40}	c_{31}	c_{22}	c_{13}	c_{04}	c_{05}
QuantCoeffArray index	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
QuantCoeffMatrix coefficient	c_{14}	c_{23}	c_{32}	c_{41}	c_{50}	c60	c_{51}	c_{42}	c_{33}	c_{24}	c_{15}	c_{06}	c_{07}	c_{16}	c_{25}	c_{34}
QuantCoeffArray index	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
QuantCoeffMatrix coefficient	c_{43}	c52	c61	c70	c71	c62	c53	c44	c35	c26	c17	c27	c36	c45	c54	c63

QuantCoeffArray index	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
QuantCoeffMatrix coefficient	c_{72}	c_{73}	c_{64}	c_{55}	c_{46}	c_{37}	c_{47}	c_{56}	c_{65}	c_{74}	c_{75}	c_{66}	c_{57}	c_{67}	c_{76}	c_{77}

9.6 Inverse quantisation

9.6.1 Identification of quantised parameter

The luminance and chrominance quantised parameters lie in the range [0 : 63].

If the current block is luminance block, then quantised parameter QP = CurrentQP. If the current block is chrominance block, then QP of the chrominance block is gained by quering table 9-7 with CurrentQP as index.

Table 9-7 Mapping relationship between QP and CurrentQP of chrominance block

CurrentQP	<43	43	44	45	46	47	48	49	50	51	52
QP	=QP	42	43	43	44	44	45	45	46	46	47
CurrentQP	53	54	55	56	57	58	59	60	61	62	63
QP	47	48	48	48	49	49	49	50	50	50	51

9.6.2 Inverse quantisation

This section defines the transformation from 2D quantised coefficient matrix to 2D transform coefficient matrix, based on the quantised parameter QP.

Transformation coefficient w_{ij} is obtained with the following euqation:

$$w_{ij} = (c_{ij} \times \text{DequantTable}(QP) + 2^{\text{ShiftTable}(QP)-1}) >> \text{ShiftTable}(QP) \quad i,j = 0..7$$

in which I and j are row and column coordinate of the qiantization coefficient and the transform coefficient in 2D matrix separately.

DequantTable and ShiftTable are defined in table 9-8.

Table 9-6 DequantTable and ShiftTable

QP	0	1	2	3	4	5	6	7
DequantTable(QP)	32768	36061	38968	42495	46341	50535	55437	60424
ShiftTable(QP)	14	14	14	14	14	14	14	14
QP	8	9	10	11	12	13	14	15
DequantTable(QP)	32932	35734	38968	42495	46177	50535	55109	59933
ShiftTable(QP)	13	13	13	13	13	13	13	13
QP	16	17	18	19	20	21	22	23
DequantTable(QP)	65535	35734	38968	42577	46341	50617	55027	60097
ShiftTable(QP)	13	12	12	12	12	12	12	12
QP	24	25	26	27	28	29	30	31
DequantTable(QP)	32809	35734	38968	42454	46382	50576	55109	60056
ShiftTable(QP)	11	11	11	11	11	11	11	11
QP	32	33	34	35	36	37	38	39
DequantTable(QP)	65535	35734	38968	42495	46320	50515	55109	60076
ShiftTable(QP)	11	10	10	10	10	10	10	10
QP	40	41	42	43	44	45	46	47
DequantTable(QP)	65535	35744	38968	42495	46341	50535	55099	60087
ShiftTable(QP)	10	9	9	9	9	9	9	9

QP	48	49	50	51	52	53	54	55
DequantTable(QP)	65535	35734	38973	42500	46341	50535	55109	60097
ShiftTable(QP)	9	8	8	8	8	8	8	8
QP	56	57	58	59	60	61	62	63
DequantTable(QP)	32771	35734	38965	42497	46341	50535	55109	60099
ShiftTable(QP)	7	7	7	7	7	7	7	7

The transform coefficients obtained from decoding bitstream should lie in the range $[-2^{31} : 2^{31}-1]$.

The following matrix CoeffMatrix is composed of 64 transform coefficients:

$$CoeffMatrix = \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} & w_{04} & w_{05} & w_{06} & w_{07} \\ w_{10} & w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{20} & w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{30} & w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{40} & w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{50} & w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \\ w_{60} & w_{61} & w_{62} & w_{63} & w_{64} & w_{65} & w_{66} & w_{67} \\ w_{70} & w_{71} & w_{72} & w_{73} & w_{74} & w_{75} & w_{76} & w_{77} \end{bmatrix}$$

9.7 Inverse transform

This section defines the transformation from 8×8 transform coefficient matrix to 8×8 residual data sample matrix, with procedures defined as follows:

- first of all, transform coefficient matrix is horizontal inverse transformed with the equation

$$H' = CoeffMatrix \times T_8^T$$

In which T_8 is a 8×8 inverse transform matrix, T_8^T is the transpose matrix of T_8, and H' is the result after horizontal inverse transform.

- then adds every coefficient in matrix H' with 4 and shifts right 3 bits, so matrix H'' is obtained.

- and H'' is vertical inverse transformed as follows:

$$H = T_8 \times H''$$

in which H is transfer 8×8 matrix:

$$T8 = \begin{bmatrix} 8 & 10 & 10 & 9 & 8 & 6 & 4 & 2 \\ 8 & 9 & 4 & -2 & -8 & -10 & -10 & -6 \\ 8 & 6 & -4 & -10 & -8 & 2 & 10 & 9 \\ 8 & 2 & -10 & -6 & 8 & 9 & -4 & -10 \\ 8 & -2 & -10 & 6 & 8 & -9 & -4 & 10 \\ 8 & -6 & -4 & 10 & -8 & -2 & 10 & -9 \\ 8 & -9 & 4 & 2 & -8 & 10 & -10 & 6 \\ 8 & -10 & 10 & -9 & 8 & -6 & 4 & -2 \end{bmatrix}$$

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} & h_{03} & h_{04} & h_{05} & h_{06} & h_{07} \\ h_{10} & h_{11} & h_{12} & h_{13} & h_{14} & h_{15} & h_{16} & h_{17} \\ h_{20} & h_{21} & h_{22} & h_{23} & h_{24} & h_{25} & h_{26} & h_{27} \\ h_{30} & h_{31} & h_{32} & h_{33} & h_{34} & h_{35} & h_{36} & h_{37} \\ h_{40} & h_{41} & h_{42} & h_{43} & h_{44} & h_{45} & h_{46} & h_{47} \\ h_{50} & h_{51} & h_{52} & h_{53} & h_{54} & h_{55} & h_{56} & h_{57} \\ h_{60} & h_{61} & h_{62} & h_{63} & h_{64} & h_{65} & h_{66} & h_{67} \\ h_{70} & h_{71} & h_{72} & h_{73} & h_{74} & h_{75} & h_{76} & h_{77} \end{bmatrix}$$

The matrix elements of H derived from the bitstream decoding should lie in the range $[-2^{15} : 2^{15}-1]$.

- elements of the residual data sample matrix ResidueMatrix can be calculated with

$$r_{ij} = [h_{ij} + 2^6] >> 7 \qquad i, j = 0..7$$

9.8 Intra-frame prediction

The current intra-frame predictive block is predicted according to the above and left decoded pixels r[i] and c[i] , i=1..16, see figure 9-6. The details of intra-frame prediction is defined in this section. And a 8×8 predictive sample matrix predMatrix can be gained after the prediction.

9.8.1 Acquisition of reference samples

Supposing the picture sample matrix where the current block locates is I, then I can represents luminance or chrominance matrix. If a block where a certain sample locates is not “exists” or not decoded yet, then that sample is “not usable”; otherwise that sample is “usable”.

Supposing the coordinate of sample in top left corner of current block is (x0,y0), then the acquisition of reference samples in the current block should observe the following rules:

If the current block belongs to macroblock VS in supermacroblock, then:

- if sample with coordinate (x0+i-1, y0-1) (i=1..8) is “usable”, then r[i] equals to I[y0-1][x0+i-1]; otherwise r[i] is “not usable”;
- if sample with coordinate (x0+i-1, y0-1) (i=9..16) is “usable”, then r[i] equals to I[y0-1][x0+i-1]; otherwise r[i] equals to r[8];
- if sample with coordinate (x0-1, y0+i-1) (i=1..8) is “usable”, then c[i] equals to I[y0+i-1][x0-1]; otherwise c[i] is “not usable”;
- if sample with coordinate (x0-1, y0+i-1) (i=9..16) is “usable”, then c[i] equals to I[y0+i-1][x0-1]; otherwise c[i] equals to c[8];
- if sample with coordinate (x0-1, y0-1) (i=9..16) is “usable”, then r[0] equals to I[y0-1][x0-1]; otherwise:

- if r[1] is “usable” and c[1] is “usable”, then r[0] = (r[1]+c[1]+1)>>1 ;
- otherwise, if r[1] is “usable” and c[1] is “not usable”, then r[0] equals to r[1];
- otherwise, if c[1] is “usable”, then r[0] equals to c[1];
- otherwise r[0] is “not usable”.

9.8.2 Luminance block intra-frame prediction

The luminance block intra-frame prediction method is identified with IntraLumaPredMode.

- If IntraLumaPredMode equals to 0 (Intra_8x8_DC prediction)
  - if r[i], c[i] (i=1..8) are both “usable”, then predMatrix[x][y] = $(\sum_{i=1}^{8} r[i] + \sum_{i=1}^{8} c[i] + 8) >> 4$ (x,y=0..7);
  - otherwise, if r[i] (i=1..8) are “usable”, then predMatrix[x][y] = $(\sum_{i=1}^{8} r[i] + 4) >> 3$ (x,y=0..7);
  - otherwise, if c[i] (i=1..8) are both “usable”, then predMatrix[x][y] = $(\sum_{i=1}^{8} c[i] + 4) >> 3$ (x,y=0..7);
  - otherwise, predMatrix[x][y] = 128 (x,y=0..7).
- If IntraLumaPredMode equals to 1 (Intra_8x8_Vertical prediction)

  then predMatrix[x][y] = r[x+1] (x,y=0..7).
- If IntraLumaPredMode equals to 2 (Intra_8x8_Horizontal prediction)

  then predMatrix[x][y] = c[y+1] (x,y=0..7).
- If IntraLumaPredMode equals to 3 (Intra_8x8_Down_Right prediction)
  - if x is greater than or equals to y, then predMatrix[x][y] = r[x-y];
  - otherwise predMatrix[x][y] = c[y-x] (x,y=0..7).
- If IntraLumaPredMode equals to 4 (Intra_8x8_Up_Right prediction)

  then predMatrix[x][y] = (r[2+x+y]+c[2+x+y]+1)>>1 (x,y=0..7).
- If IntraLumaPredMode equals to 5 (Intra_8x8_Down_Right_Down prediction)
  - when y equals to 0, 2, 4 or 6, then let i = x–(y>>1),
    - if I is greater than or euals to 0, then predMatrix[x][y] = (r[i]+r[1+i]+1)>>1;
    - otherwise predMatrix[x][y] = c[y-1-2×x] （x=0..7);
  - when y is greater than or equals to 1, 3, 5 or 7 ，then let i = x–(y>>1),

- if I is greater than or equals to 0, then predMatrix[x][y] = r[i];
- otherwise predMatrix[x][y] = c[y-1-2×x].

- If IntraLumaPredMode equals to 6 (Intra_8x8_Down_Left_Down prediction)
  - when y equals to 0, 2, 4 or 6, predMatrix[x][y] = (r[1+x+(y>>1)]+r[2+x+(y>>1)]+1)>>1 （x=0..7）;
  - when y equals to 1, 3, 5 or 7, predMatrix[x][y] = r[2+x+(y>>1)]（x=0..7）.
- If IntraLumaPredMode equals to 7 (Intra_8x8_Right_Up_Right prediction）
  - when x equals to 0, 2, 4 or 6, predMatrix[x][y] = (c[1+y+(x>>1)]+c[2+y+(x>>1)]+1)>>1 （y=0..7）;
  - when x equals to 1, 3, 5 or 7, then predMatrix[x][y] = c[2+y+(x>>1)]（y=0..7）.
- If IntraLumaPredMode equals to 8 (Intra_8x8_Right_Down_Right prediction)
  - when x equals to 0, 2, 4 or 6, let i = (x>>1)-y,
    - if i is less than 0, then predMatrix[x][y] = (r[-i]+c[1-i])>>1;
    - if I equals to 0, then predMatrix[x][y] = (r[0]+c[1])>>1;
    - otherwise predMatrix[x][y]=r[-1-2×y+x]（y=0..7）;
  - when x equals to 1, 3, 5 or 7, let i = (x>>1)-y,
    - if I is less than 0, then predMatrix[x][y] = c[-i];
    - otherwise predMatrix[x][y] = r[-1-2×y+x]（y=0..7）.

9.8.3 Chrominance block intra-frame prediction

The chrominance block intra-frame prediction method is identified with IntraChromaPredMode.

- if IntraChromaPredMode equals to 0 (Chroma_Pred_Vertical prediction)

  then predMatrix[x][y] = r[1+x]（x,y=0..7）.

- if IntraChromaPredMode equals to 1 (Chroma_Pred_Horizontal prediction)

  then predMatrix[x][y] = c[1+y]（x,y=0..7）.

- if IntraChromaPredMode equals)to 2 (Chroma_Pred_DC prediction)
  - if r[i],c[i] (i=1..8) are all “usable”, predMatrix[x][y] = $(\sum_{i=1}^{8} r[i] + \sum_{i=1}^{8} c[i] + 8) >> 4$(x,y=0..7);

- othersie, if r[i] (i=1..8) are “usable”, then predMatrix[x][y] = $(\sum_{i=1}^{8} r[i] + 4) >> 3$ (x,y=0..7) ;

- otherwise, if c[i] (i=1..8) are “usable”, then predMatrix[x][y] = $(\sum_{i=1}^{8} c[i] + 4) >> 3$ (x,y=0..7);

- otherwise, predMatrix[x][y] = 128 (x,y=0..7) .

- if IntraChromaPredMode equals to 3 (Chroma_Pred_Plane prediction)

  let ih = $\sum_{i=1}^{8} i \times (r[i+8] - r[8-i])$, iv = $\sum_{i=1}^{8} i \times (c[i+8] - c[8-i])$,

  ia = (r[16]+c[16])<<4, ib = (5×ih+32)>>6, ic = (5×iv+32)>>6,

  then predMatrix[x][y] = Max(0,Min((ia+(x-7)×ib+(y-7)×ic+16)>>5,255)) (x,y=0..7) .

9.9 Interframe prediction

9.9.1 Derivation process of luminance motion vectors

If the current macroblock type is P_Skip, B_Skip or B_Direct_16x16, or if the current subblock type is SB_Direct_8x8, then its motion vectors are derived based on following definitions; otherwise motion vectors obtained as described in section 9.4.4, can be partitioned in macroblock partition order (defined in figure 6-6), and assigned to corresponding subblocks.

- If the current macroblock type is P_Skip:
  - if the upper macroblock B or left macroblock A of the current macroblock is “not usable”, then mvE equals to zero vector;
  - if mvA or mvB is zero vector, then mvE equals to zero vector;
  - in other cases, mvE = MVEPred.

- If the current macroblock type is B_Skip or B_Direct_16x16, or the current block type is SB_Direct_8x8:
  - In such case, the reference picture of the current block is default reference picture, namely picture (or field) labeled with 0 in figure 9-2.
    In backward reference frames, block which is corresponding to the top left corner sample positions of the current block has motion vector of mv (mv_x,mv_y) , distance index of DistanceIndex, and the forward distance index of current block is DistanceIndexFw,
    - if PictureStructure of the current block located picture equals to 1, then the block located picture with corresponding position has PictureStructure equal to 0, and mv_y = mv_y × 2;
    - if PictureStructure of the current block located picture equals to 1 ,1, then the block located picture with corresponding position has PictureStructure equal to 0, and mv_y = mv_y / 2.

- The forward distance index of the current block is DistanceIndexFw, then forward motion vector of current block mvFw(mvFw_x, mvFw_y) is:
  - if mv_x is less than 0, then mvFw_x= -(((2048/DistanceIndex)×(1-mv_x× DistanceIndexFw)) >>11); otherwise, mvFw_x =((2048/ DistanceIndex)×(1+mv_x×DistanceIndexFw))>>11.
  - if mv_y is less than 0, then mvFw_y=-(((2048/DistanceIndex)×(1-mv_y× DistanceIndexFw)) >>11); otherwise, mvFw_y=((2048/DistanceIndex)×(1+mv_y× DistanceIndexFw))>>11.
- The backward distance index of the current block is DistanceIndexBw, then the backward motion vector is:
  - if mv_x is less than 0, then mvBw_x=((2048/DistanceIndex)×(1-mv_x× DistanceIndexBw)) >>11;otherwise mvBw_x=-(((2048/DistanceIndex)×(1+mv_x× DistanceIndexBw))>>11);
  - if mv_y is less than 0, then mvBw_y=((2048/DistanceIndex)×(1-mv_y× DistanceIndexBw))>>11; otherwise mvBw_y=-(((2048/DistanceIndex)×(1+mv_y× DistanceIndexBw))>>11);
  - if in backward reference blocks, the macroblock type of the current block which has the corresponding top left corner sample positions in current block being I_8x8, then the motion vectors of that block are that decoded according to section 9.4.4.

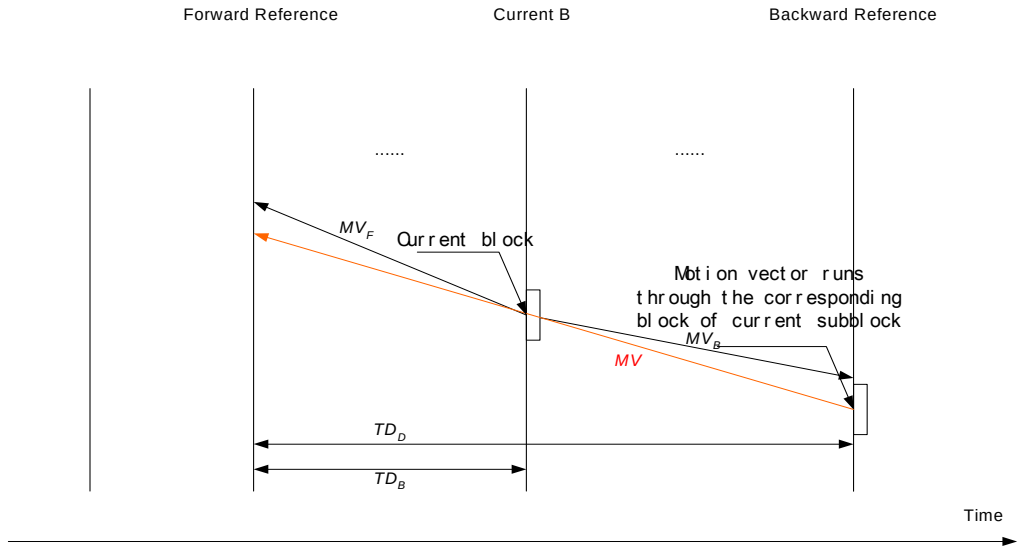
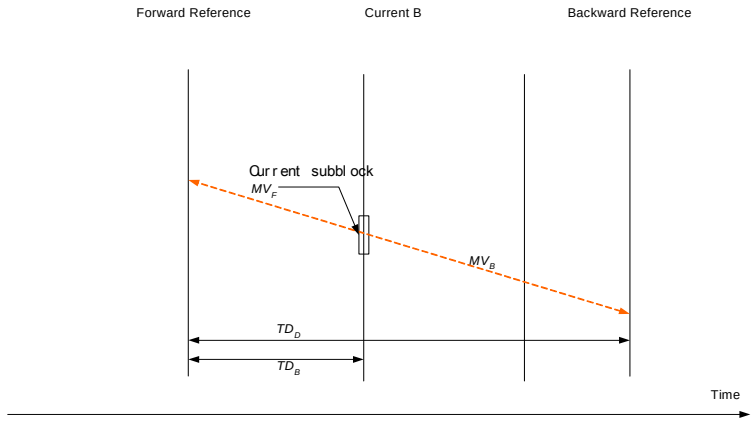


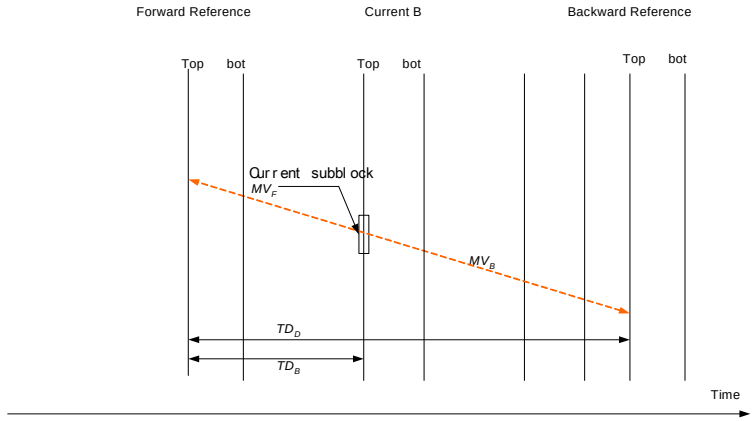
Figure 9-5 Derivation process of motion vector in the frame decoding

The current macroblock type is symmetric mode, its motion vector is derived based on the following methods:

If PictureStructure equals to 1, then its backward reference index is the same with its forward reference index; othrewise, its backward reference index equals to subtract its forward reference index from 1. Forward motion vector of the symmetric mode block mvFw can be obtained with method described in section 9.4.4, and backward motion vector of the current block mvBw=-(mvFw×DistanceIndexBw×(256/DistanceIndexFw) +128)>>8.



(a) Symmetric mode when PictureStructure of the current frame equals to 1



(b) Symmetric mode when PictureStructure of the current frame equals to 0

Figure 9-3 Symmetric mode

9.9.2 Derivation process of reference samples

First of all, luminance is processed with half sample and quarter sample interpolation, then the corresponding reference sample is derived from motion vector.

If the referenced integral sample in interpolation is beyond the reference picture, then it can be replaced with integral sample (border or corner sample) which is nearest to the reference sample in picture.

9.9.2.1 Interpolation process of luminance samples

Figure 9-7 presents the positions of the integral, half and quarter samples in reference picture, in which shadow blocks labeled upper case letters indicate integral sample positions, and blocks labeled lower case letters indicate half sample and quarter sample positions.

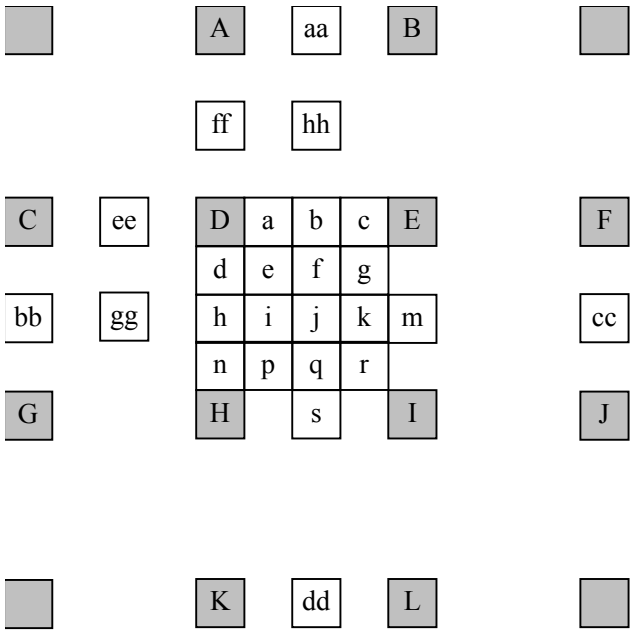


Figure 9-2 Positions of the integral, half and quarter samples

The predictive value at half sample position can be obtained with calculation of 4 tapping filter F_1(-1, 5, 5, -1), and the predictive value at quarter sample position can be obtained with calculation of 4 tapping filter F_1(1, 7, 7, 1).

The interpolation processes of half samples are:

- half sample b: at first, the nearest 4 integral sample values are filtered with F_1 in horizontal direction, then median value b'=(-C+5D+5E-F); and the final predictive value b=Clip1((b'+4)>>3).
- half sample h: at first, the nearest 4 integral sample values are filtered with F_1 in vertical direction, then median value h'=(-A+5D+5H-K); and the final predictive value h=Clip1((h'+4)>>3).
- half sample j: at first, the nearest 4 half sample values are median filtered with F_1 in horizontal or vertical direction, then median value j'=(-bb+5h'+5m'-cc), or =(-aa+5b'+5s'-dd). In which half sample median value labeled with aa，dd and s' can be obtained after horizontal filter with F_1 (which is the same with that of b'), and half sample median value labeled with bb，cc and m' can be obtained after vertical filter with F_1 (which is the same with that of h'). The final predictive value j = Clip1((j'+32)>>6). The value obtained with horizontal or vertical filter is the same.

The interpolation processes of quarter sample are:

- quarter sample a: at first, the nearest 4 samples, ee, D', b' and E', are filtered with F_2 in horizontal direction, then median value a'=(ee+7D'+7b'+E'); and the final predictive value a=Clip1((a'+64)>>7). In which ee and b' are half samples median values of corresponding positons, D' and E' are integral samples whose median values are magnified 8 times. The interpolation process of quarter sample c is the same with that of a.

- quarter sample d: at first, the nearest 4 samples, ff, D', h' and H', are filtered with F_2 in vertical direction, then median value d'=(ff+7D'+7h'+H'); and the final predictive value d=Clip1((d'+64)>>7). In which ff and h' are half samples median values of corresponding positons, D' and H' are integral samples whose median values are magnified 8 times. The interpolation process of quarter sample n is the same with that of d.

- quarter sample i: at first, the nearest 4 samples, gg, h'', j' and m'', are filtered with F_2 in horizontal direction, then median value i'=(gg+7h''+7j'+m''); and the final predictive value i=Clip1((i'+512)>>10). In which gg and j' are half samples median values of corresponding positons, h''' and m'' are integral samples whose median values are magnified 8 times. The interpolation process of quarter sample k is the same with that of i.

- quarter sample f: at first, the nearest 4 samples, hh, b'', j' and s'', are filtered with F_2 in vertical direction, then median value f'=(hh+7b''+7j'+s''); and the final predictive value f=Clip1((f'+512)>>10). In which hh and j' are half samples median values of corresponding positons, b'' and s'' are integral samples whose median values are magnified 8 times. The interpolation process of quarter sample q is the same with that of f.

- quarter sample e, g, p and r：

  e = (D'' + j' + 64) >> 7
  g = (E'' + j'+ 64) >> 7
  p = (H'' + j' + 64) >> 7
  r = (I'' + j'+ 64) >> 7

  in which D'', E'', H'' and I'' are integral samples of the corresponding positions which are magnified 64 times, and j' is the half sample median values of corresponding positon.

In figure 9-3, the pixel position at top left corner of the current luminance block E is (x_L,y_L). Predictive sample matrix predMatrix[y_L][x_L] is evaluated based on table 9-9.

Table 9-7 Predictive sample matrix elements

xFracL	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
yFracL	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
predMatrix [yL][xL]	D	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

In which xFracL equals to horizontal component of mvE - mvE_x & 3, and yFracL equals to vertical component of mvE - mvE_y & 3.

9.9.2.2 Interpolation process of chrominance samples

Chrominance sample interpolation utilizes the motion vector of its corresponding luminance block – mvE, and the horizontal component of mvE is mvE_x, the vertical component of mvE is mvE_y. Chrominance sample interpolation is described in figure 9-8, in which A, B, C and D are surrounding integral samples of the interpolation samples, d_x and d_y are the horizontal and vertical distance between predictive sample and A respectively, dx equals to mvE_x & 7, and dy equals to mvE_y & 7. The position relation of these variables and reference samples are presented in figure 9-8.

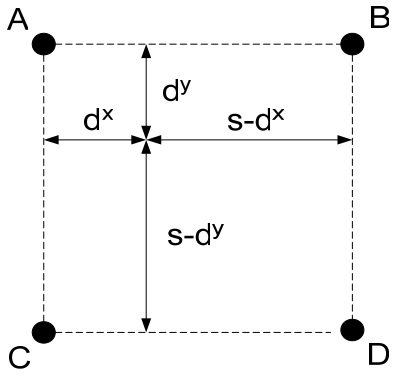


Figure 9-8 Eighth chrominance interpolation

The element predMatrix[y][x] of predictive sample matrix is calculated as follows:

predMatrix[y][x] = ((8–dx)×(8–dy)×A + dx×(8–dy)×B + (8–dx)×dy×C + dx×dy×D + 32) >> 6.

9.10 Loop filter

Every border between 8×8 luminance blocks has a "border intensity" Bs, and the border intensity of chrominance block is replaced with border Bs of lumiance block with the corresponding position, see figure 9-9. if Bs equals to 0, then need not to filter border, otherwise borders are filtered based on features of the local samples and Bs.

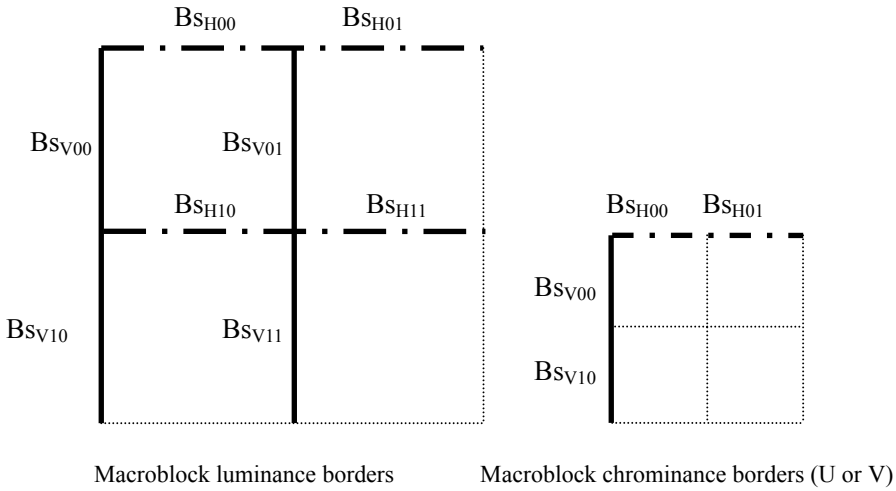


Figure 9-4 Illustration of borders in macroblock need to be filtered (black real line represents vertical border, and black dashed line represents horizontal border)

All borders of macroblock can be filtered, except borders of pictures and slices. Here macroblock borders are defined as borders of 8×8 blocks of the macroblock, and also the upper and left borders of the current and adjacent macroblocks.

With a macroblock as the unit, loop filter is performed in raster scan order. Every macroblock in a picture is filtered as follows:

Luminance and chrominance are loop filtered individually as showed in figure 9-9. First of all, vertical borders are filered from left to right, then horizontal borders are filered from top to bottom. The samples at top or left part of the curent block may have been adapted in former macroblock loop filter process. The loop filter inputs of the current macroblock are these adapted samples, and the current macroblock loop filter may adapt these samples further. The adapted samples in the vertical border filter process of the current macroblock are set to be the input of horizontal border filter process.

Intra-frame prediction utilizes reconstructed picture samples before the loop filter.

9.10.1 Derivation process of border filter intensity based on contents

Based on macroblock types and motion vector of 8×8 luminance block in a macroblock, the value Bs is calculated with following method:

- if one of the two 8×8 blocks on border sides or both of them belongs to intra-frame predictive macroblock, then Bs equals to 2.
- otherwise, if any of the following conditions is fulfiled, then Bs equals to 1
  - the reference pictures of two blocks are different;
  - the reference pictures of two blocks are the same，but the value of either component of the two motion vectors is greater than one pixel;
  - otherwise, Bs equals to 0.

9.10.2 Derivation process of block border thresholds

Figure 9-10 presents the 6 samples at horizontal or vertical borders of block p and q (borders are denoted with black bold line). P_0, P1, Q0 and Q_1 represents the filtered samples of p0, p1, q_0 and q_1 individually.

p_2 p_1 p_0 ▌ q_0 q_1 q_2

Figure 9-30 Representation of horizontal or vertical border samples in 8×8 block

If the following euqation is TRUE, then filters border samples as follows

$Bs != 0 \;\&\&\; Abs(p_0 - q_0) < \alpha \;\&\&\; Abs(p_1 - p_0) < \beta \;\&\&\; Abs(q_1 - q_0) < \beta$

in which α and β are block border thresholds. Based on QPav (mean values of QP) of two blocks and AlphaCOffset, BetaOffset in picture header, IndexA and IndexB can be calculated.

The mean values of QP, QP_{av}, of the two blocks are:

$QP_{av} = (QP_p + QP_q + 1) >> 1$

IndexA and IndexB are:

$IndexA = Clip3(0, 63, QP_{av} + AlphaCOffset)$

$IndexB = Clip3(0, 63, QP_{av} + BetaOffset)$

Based on the relationship between IndexA, IndexB and thresholds α, β, α and β can be evaluated with quering table 9-10.

Table 9-8 Relationship between IndexA, IndexB and block border thresholds α, β

Index	α	β
0	0	0
1	0	0

Index	α	β
16	4	2
17	4	2

Index	α	β
32	22	6
33	24	7

Index	α	β
48	46	15
49	48	16

2	0	0
3	0	0
4	0	0
5	0	0
6	1	0
7	1	0
8	1	0
9	1	0
10	1	0
11	2	0
12	2	0
13	2	0
14	3	0
15	3	0

18	5	2
19	5	3
20	6	3
21	7	3
22	8	3
23	9	4
24	10	4
25	11	4
26	12	5
27	13	5
28	15	5
29	16	5
30	18	6
31	20	6

34	26	7
35	28	7
36	30	8
37	33	8
38	33	8
39	35	9
40	35	9
41	36	10
42	37	10
43	37	11
44	39	11
45	39	12
46	42	13
47	44	14

50	50	17
51	52	18
52	53	19
53	54	20
54	55	21
55	56	22
56	57	23
57	58	23
58	59	24
59	60	24
60	61	25
61	62	25
62	63	26
63	64	26

9.10.3 Border filter process while Bs equals 2

Firstly defines $ap=|p_2-p_0|$, $aq=|q_2-q_0|$.

The filtering process for samples p_0, p_1, q_0 and q_1 on two sides of luminance block:

if ($ap < \beta$ && $|p_0 - q_0| < ((\alpha >> 2) + 2)$) {

$P_0 = (p_1 + 2 \times p_0 + q_0 + 2) >> 2$

$P_1 = (2 \times p_1 + p_0 + q_0 + 2) >> 2$

}

else

$P_0 = (2 \times p_1 + p_0 + q_0 + 2) >> 2$

if ($aq < \beta$ && $|p_0 - q_0| < ((\alpha >> 2) + 2)$) {

$Q_0 = (q_1 + 2 \times q_0 + p_0 + 2) >> 2$

$Q_1 = (2 \times q_1 + q_0 + p_0 + 2) >> 2$

}

else

$Q_0 = (2 \times q_1 + q_0 + p_0 + 2) >> 2$

For samples p_0 and q_0 on two sides of chrominance block, the above method is samely used.

9.10.4 Border filter process while Bs equals 1

When the value of the border filter intensity is 1, then p_0 and q_0 are calculated as follows:

$delta = Clip3(-C, C, (((q_0 - p_0) \times 3 + (p_1 - q_1) + 4) >> 3)$

$P_0 = Clip1(p_0 + delta)$

$Q_0 = Clip1(q_0 - delta)$

Then decide whether p_1 and q_1 can be filtered:

- for chrominance borders, p_1 and q_1 can not be filtered.

- if there exists ap less than β at luminance borders, then

$P_1 = Clip1(p_1 + Clip3(-C, C, (((P_0 - p_1) \times 3 + (p_2 - Q_0) + 4) >> 3)))$

- if there exists aq less than β at luminance borders, then

$Q_1 = Clip1(q_1 - Clip3(–C, C, (((q_1 – Q_0) × 3 + (P_0 – q_2) + 4) >> 3)))$

In the above filter process, the variable C is filter cut-off parameter, see the relationship between C and IndexA in table 9-11.

Table 9-9 The relationship between filter cut-off parameter C and IndexA

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

Appendix A
(Normative appendix)
Variable length code tables

This appendix defines variable length code tables. In tables, Run represents quantised coefficient values, and numbers in tables represents the values of syntax element **trans_coefficient**. Level and Run can be obtained from tables based on **trans_coefficient** when decoding. The corresponding numbers in EOB column of table represents **trans_coefficient** which stands for EOB.

To save specification length, only table of Level being greater than 0 is presented here. When Level is less than 0, the **trans_coefficient** value is obtained with the following method:

- let level = -Level;
- get the value of **trans_coefficient** from table with Level and Run;
- add the value of **trans_coefficient** with 1, then get the value when Level is less than 0.

Table A-1 VLC0_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	Level>0		
	1	2	3
0	0	22	38
1	2	32	-
2	4	44	-
3	6	50	-
4	8	54	-
5	10	-	-
6	12	-	-
7	14	-	-
8	16	-	-
9	18	-	-
10	20	-	-
11	24	-	-
12	26	-	-
13	28	-	-
14	30	-	-
15	34	-	-
16	36	-	-
17	40	-	-
18	42	-	-
19	46	-	-
20	48	-	-
21	52	-	-
22	56	-	-

Table A-2 VLC1_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0					
		1	2	3	4	5	6
	8	-	-	-	-	-	-
0	-	0	4	15	27	41	55
1	-	2	17	35	-	-	-
2	-	6	25	53	-	-	-
3	-	9	33	-	-	-	-
4	-	11	39	-	-	-	-
5	-	13	45	-	-	-	-
6	-	19	49	-	-	-	-
7	-	21	51	-	-	-	-
8	-	23	-	-	-	-	-
9	-	29	-	-	-	-	-
10	-	31	-	-	-	-	-
11	-	37	-	-	-	-	-
12	-	43	-	-	-	-	-
13	-	47	-	-	-	-	-
14	-	57	-	-	-	-	-

Table A-3 VLC2_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0								
		1	2	3	4	5	6	7	8	9
	8	-	-	-	-	-	-	-	-	-
0	-	0	2	6	13	17	27	35	45	55
1	-	4	11	21	33	49	-	-	-	-
2	-	9	23	37	-	-	-	-	-	-
3	-	15	29	51	-	-	-	-	-	-
4	-	19	39	-	-	-	-	-	-	-
5	-	25	43	-	-	-	-	-	-	-
6	-	31	53	-	-	-	-	-	-	-
7	-	41	-	-	-	-	-	-	-	-
8	-	47	-	-	-	-	-	-	-	-
9	-	57	-	-	-	-	-	-	-	-

Table A-4 VLC3_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0											
		1	2	3	4	5	6	7	8	9	10	11	12
	8	-	-	-	-	-	-	-	-	-	-	-	-
0	-	0	2	4	9	11	17	21	25	33	39	45	55
1	-	6	13	19	29	35	47	-	-	-	-	-	-
2	-	15	27	41	57	-	-	-	-	-	-	-	-
3	-	23	37	53	-	-	-	-	-	-	-	-	-
4	-	31	51	-	-	-	-	-	-	-	-	-	-
5	-	43	-	-	-	-	-	-	-	-	-	-	-
6	-	49	-	-	-	-	-	-	-	-	-	-	-

Table A-5 VLC4_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	0	2	4	7	9	11	15	17	21	23	29	33	35	43	47	49	57
1	-	13	19	27	31	37	45	55	-	-	-	-	-	-	-	-	-	-
2	-	25	41	51	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	53	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A-6 VLC5_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	1	3	5	7	9	11	13	15	17	19	23	25	27	31	33	37	41	45	49	51	55
1	-	21	29	35	43	47	53	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	39	57	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A-7 VLC6_Intra (Mapping table used for decoding Run of intra-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	35	37	39	41	43
1	-	33	45	55	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2		22	23	24	25	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	47	49	51	53	57	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A-8 VLC0_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	Level>0		
	1	2	3
0	0	26	40
1	2	46	-
2	4	-	-
3	6	-	-
4	8	-	-
5	10	-	-
6	12	-	-
7	14	-	-
8	16	-	-
9	18	-	-
10	20	-	-
11	22	-	-
12	24	-	-
13	28	-	-
14	30	-	-
15	32	-	-
16	34	-	-
17	36	-	-
18	38	-	-
19	42	-	-
20	44	-	-
21	48	-	-
22	50	-	-
23	52	-	-
24	54	-	-

25	56	-	-

Table A-9 VLC1_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0			
		1	2	3	4
	2	-	-	-	-
0	-	0	13	29	47
1	-	3	23	57	-
2	-	5	35	-	-
3	-	7	39	-	-
4	-	9	43	-	-
5	-	11	49	-	-
6	-	15	55	-	-
7	-	17	-	-	-
8	-	19	-	-	-
9	-	21	-	-	-
10	-	25	-	-	-
11	-	27	-	-	-
12	-	31	-	-	-
13	-	33	-	-	-
14	-	37	-	-	-
15	-	41	-	-	-
16	-	45	-	-	-
17	-	51	-	-	-
18	-	53	-	-	-

Table A-10 VLC2_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0					
		1	2	3	4	5	6
	2	-	-	-	-	-	-
0	-	0	5	11	23	35	47
1	-	3	13	27	49	-	-
2	-	7	21	45	-	-	-
3	-	9	29	55	-	-	-
4	-	15	37	-	-	-	-
5	-	17	41	-	-	-	-
6	-	19	53	-	-	-	-
7	-	25	-	-	-	-	-
8	-	31	-	-	-	-	-
9	-	33	-	-	-	-	-

10	-	39	-	-	-	-	-
11	-	43	-	-	-	-	-
12	-	51	-	-	-	-	-
13	-	57	-	-	-	-	-

Table A-11 VLC3_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0								
		1	2	3	4	5	6	7	8	9
	2	-	-	-	-	-	-	-	-	-
0	-	0	3	7	13	17	27	35	43	55
1	-	5	11	21	33	51	-	-	-	-
2	-	9	23	37	57	-	-	-	-	-
3	-	15	29	47	-	-	-	-	-	-
4	-	19	41	-	-	-	-	-	-	-
5	-	25	49	-	-	-	-	-	-	-
6	-	31	-	-	-	-	-	-	-	-
7	-	39	-	-	-	-	-	-	-	-
8	-	45	-	-	-	-	-	-	-	-
9	-	53	-	-	-	-	-	-	-	-

Table A-12 VLC4_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0											
		1	2	3	4	5	6	7	8	9	10	11	12
	2	-	-	-	-	-	-	-	-	-	-	-	-
0	-	0	3	5	9	11	17	21	25	33	41	45	55
1	-	7	13	19	29	35	49	-	-	-	-	-	-
2	-	15	27	43	57	-	-	-	-	-	-	-	-
3	-	23	37	51	-	-	-	-	-	-	-	-	-
4	-	31	53	-	-	-	-	-	-	-	-	-	-
5	-	39	-	-	-	-	-	-	-	-	-	-	-
6	-	47	-	-	-	-	-	-	-	-	-	-	-

Table A-13 VLC5_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	1	3	5	7	9	13	15	17	21	25	29	33	39	43	49	53
1	-	11	19	27	31	41	45	57	-	-	-	-	-	-	-	-	-

2	-	23	37	51	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	35	55	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A-14 VLC6_Inter (Mapping table used for decoding Run of inter-frame encoded luminance block and non-zero quantised coefficient values)

Run	EOB	Level > 0																				
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	1	3	5	7	9	11	13	17	19	21	23	25	29	33	35	39	41	43	47	49	57
1	-	15	27	37	45	55	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	31	51	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	53	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table A-15 VLC0_Chroma (Mapping table used for decoding chrominance block and non-zero quantised coefficient values)

Run	Level>0			
	1	2	3	4
0	0	14	32	56
1	2	48	-	-
2	4	-	-	-
3	6	-	-	-
4	8	-	-	-
5	10	-	-	-
6	12	-	-	-
7	16	-	-	-
8	18	-	-	-
9	20	-	-	-
10	22	-	-	-
11	24	-	-	-
12	26	-	-	-
13	28	-	-	-
14	30	-	-	-
15	34	-	-	-
16	36	-	-	-
17	38	-	-	-
18	40	-	-	-

19	42	-	-	-
20	44	-	-	-
21	46	-	-	-
22	50	-	-	-
23	52	-	-	-
24	54	-	-	-

Table A-16 VLC1_Chroma (Mapping table used for decoding chrominance block and non-zero quantised coefficient values)

Run	EOB	Level > 0				
		1	2	3	4	5
	0	-	-	-	-	-
0	-	1	5	15	29	43
1	-	3	21	45	-	-
2	-	7	37	-	-	-
3	-	9	41	-	-	-
4	-	11	53	-	-	-
5	-	13	-	-	-	-
6	-	17	-	-	-	-
7	-	19	-	-	-	-
8	-	23	-	-	-	-
9	-	25	-	-	-	-
10	-	27	-	-	-	-
11	-	31	-	-	-	-
12	-	33	-	-	-	-
13	-	35	-	-	-	-
14	-	39	-	-	-	-
15	-	47	-	-	-	-
16	-	49	-	-	-	-
17	-	51	-	-	-	-
18	-	55	-	-	-	-
19	-	57	-	-	-	-

Table A-17 VLC2_Chroma (Mapping table used for decoding chrominance block and non-zero quantised coefficient values)

Run	EOB	Level > 0								
		1	2	3	4	5	6	7	8	9
	2	-	-	-	-	-	-	-	-	
0	-	0	3	7	11	17	27	33	47	53
1	-	5	13	21	37	55	-	-	-	-

2	-	9	23	41	-	-	-	-	-	-
3	-	15	31	57	-	-	-	-	-	-
4	-	19	43	-	-	-	-	-	-	-
5	-	25	45	-	-	-	-	-	-	-
6	-	29	-	-	-	-	-	-	-	-
7	-	35	-	-	-	-	-	-	-	-
8	-	39	-	-	-	-	-	-	-	-
9	-	49	-	-	-	-	-	-	-	-
10	-	51	-	-	-	-	-	-	-	-

Table A-18 VLC3_Chroma (Mapping table used for decoding chrominance block and non-zero quantised coefficient values)

Run	EOB	Level > 0												
		1	2	3	4	5	6	7	8	9	10	11	12	13
	0	-	-	-	-	-	-	-	-	-	-	-	-	-
0	-	1	3	5	7	11	15	19	23	29	35	43	47	53
1	-	9	13	21	31	39	51	-	-	-	-	-	-	-
2	-	17	27	37	-	-	-	-	-	-	-	-	-	-
3	-	25	41	-	-	-	-	-	-	-	-	-	-	-
4	-	33	55	-	-	-	-	-	-	-	-	-	-	-
5	-	45	-	-	-	-	-	-	-	-	-	-	-	-
6	-	49	-	-	-	-	-	-	-	-	-	-	-	-
7	-	57	-	-	-	-	-	-	-	-	-	-	-	-

Table A-19 VLC4_Chroma (Mapping table used for decoding chrominance block and non-zero quantised coefficient values)

Run	EOB	Level > 0																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-		
0	-	1	3	5	7	9	11	13	15	19	21	23	27	29	33	37	41	43	51	55
1	-	17	25	31	39	45	53	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	35	49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	-	57	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Appendix B
(Normative appendix)
Profiles and levels

Profiles and levels are used to specify coded bitstream with a variety of restrictions, and thereby the decoding capabilities needed to decode a bitstream. They can also be used to specify the mutual operation point when implementing different decoders.

“Profile” is the subset of syntax specified in this specification.

“Level” is the restricted set of syntax and decoding process parameter values, under some profile.

“Profile” is a subset of syntax, semantics and algorithm features stated in this specification. Decoder which agrees with some “profile” should totally support this subset defined by the “profile”.

“Level” is the restriction of syntax element values defined in this specification. All “profiles” use the same group of “level” definitions, but different decoders implementation may support different “levels” under different “profiles”. When “profile” is given, different “level” generally means different requirements on the decoder capabilities and storage capability.

B.1 Decoder capability

To correctly decode the bitstream of some profile and level, the decoder according to this specification should fulfil the demands on some capabilities.

profile_id and **level_id** define the profile and level of bitstream.

B.2 Profiles

See profiles defined in this specification in table B-1.

Table B-1 The meaning of profile_id

profile_id	Profile
50	Forbidden
1 ~ 255	Reserved

B.3 Levels

See levels defined in this specification in table B-2.

Table B-2 The meaning of level_id

level_id	**Profile**
0	Forbidden
1 ~ 255	Reserved

Appendix C
(Normative appendix)
Pseudo start code

This appendix defines the approach to avoid the appearance of pseudo start code in bitstream. The form, meaning and filling method for start code alignment are illustrated in section 7.1.1.

To avoid the appearance of pseudo start code, the following steps are performed in coding: when writing a bit, if that bit is the second lowest valid bit of the byte, then check the 22 bits wrote before this bit. If these 22 bits are all “0”, then insert “10” before this bit, and this bit becomes the highest valid bit of next byte.

The following steps can be performed in the decoding: when read a byte, check the two bytes red before and also the current byte, if these three bytes make up of the bit string of ‘0000 0000 0000 0000 0000 0010’, then the lowest two valid bits of the current byte can be discarded. This discarding process may adopt any equivalent form, which is not stated in this specification.

Appendix D
(Reference appendix)
Bitstream reference decoder

References