
**Information technology — Coding of
audio-visual objects —**

**Part 33:
Internet video coding**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 33: Codage vidéo Internet*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword	v
Introduction	vi
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviations	7
5 Conventions	7
5.1 Arithmetic operators	7
5.2 Logical operators	8
5.3 Relational operators	8
5.4 Bitwise operators	8
5.5 Assignment	8
5.6 Order of operation precedence	9
5.7 Mathematical functions	9
5.8 Variables, syntax elements and tables	10
5.9 Text description of logical operations	11
5.10 Processes	12
5.11 Description of bitstream syntax parsing process and decoding process	12
5.11.1 Method of describing bitstream syntax	12
5.11.2 Syntax functions	14
5.11.3 Syntax descriptors	15
5.11.4 Reserved, forbidden and marker bit	16
6 Source, coded, decoded and output data formats	16
6.1 Source	16
6.2 Colour format	16
6.3 Coded bitstream format	17
6.4 Sequence header	17
6.5 Frame	17
6.6 Frame types	17
6.7 Slice	18
6.8 Macroblock	18
6.9 Block	18
6.10 Frame re-ordering	19
6.11 Reference frames	19
6.12 Inverse scanning processes and derivation processes for neighbours	20
6.12.1 General	20
6.12.2 Inverse macroblock scanning process	20
6.12.3 Inverse macroblock partition scanning process	20
6.12.4 Inverse 8x8 luma block scanning process	21
6.12.5 Inverse 4x4 luma block scanning process	21
6.12.6 Derivation process of the availability for macroblock addresses	21
6.12.7 Derivation process for neighbouring macroblock addresses and their availability	22
6.12.8 Derivation processes for neighbouring macroblocks, blocks, and partitions	23
6.12.9 Derivation process for neighbouring locations	25
7 Syntax and semantics	26
7.1 Bitstream syntax	26
7.1.1 Start codes	26
7.1.2 Video sequence	27
7.1.3 Frame	28
7.1.4 Slice	30
7.1.5 Macroblock	30

	7.1.6	Block.....	33
7.2		Video bitstream semantics.....	34
	7.2.1	Start code.....	34
	7.2.2	Video sequence.....	34
	7.2.3	Frame.....	37
	7.2.4	Slice.....	38
	7.2.5	Macroblock.....	38
	7.2.6	Block.....	41
8		Decoding process.....	41
8.1		General.....	41
8.2		Intra prediction.....	42
	8.2.1	General.....	42
	8.2.2	Intra_4x4 prediction process for luma samples.....	42
	8.2.3	Intra_8x8 prediction process for luma samples.....	45
	8.2.4	Intra_16x16 prediction process for luma samples.....	47
	8.2.5	Intra prediction for 8x8 chroma block.....	49
8.3		Inter prediction.....	51
	8.3.1	General.....	51
	8.3.2	Derivation process for motion vector components and reference indices.....	52
	8.3.3	Decoding process for inter prediction samples.....	60
8.4		Transform coefficient decoding process and frame reconstruction process.....	69
	8.4.1	General.....	69
	8.4.2	Inverse scanning.....	70
	8.4.3	Inverse quantization.....	71
	8.4.4	Inverse transform process.....	74
	8.4.5	Reconstruction.....	79
8.5		Loop filtering.....	79
8.6		Reference frame buffer management.....	81
9		Parsing process.....	82
9.1		General.....	82
9.2		ue(v).....	82
9.3		Parsing process for transform coefficient levels.....	82
9.4		ae(v).....	83
	9.4.1	General.....	83
	9.4.2	Description.....	83
	9.4.3	Initialization.....	84
	9.4.4	Binarization process.....	84
	9.4.5	Parsing binary string.....	87
10		Profiles and levels.....	97
10.1		General.....	97
10.2		Profiles.....	98
10.3		Levels.....	98

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 14496 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document specifies Internet video coding, a video compression technology that is intended to be suitable for video distribution models currently adopted on the Internet.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from:

Nokia Technologies Oy
Joensuunkatu 7E
FIN-24100 Salo
FINLAND
Telephone : +358 50 366 2022

Apple Inc.
Intellectual Property and Licensing
1 Infinite Loop, MS 169-3IPL
Cupertino, CA 95014
USA
Telephone: +1(408) 974-0015

Industry-University Cooperation Foundation Hanyang University
222 Wangsimni-ro, Seongdong-gu
Seoul 04763
REPUBLIC OF KOREA
Telephone: +82-2-2220-2212

Mitsubishi Electric Corporation
Corporate Licensing Division
2-7-3 Marunouchi, Chiyoda-ku
Tokyo 100-8310
JAPAN
Telephone: +81-3-3218-3465

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121
USA
Telephone: +1 (858) 587-1121

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Coding of audio-visual objects —

Part 33: Internet video coding

1 Scope

This document specifies MPEG-4 Internet video coding.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Rec. ITU-T H.262 | ISO/IEC 13818-2: 2013, *Information technology — Generic coding of moving pictures and associated audio information — Part 2: Video*

IEC 60461, *Time and control code*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

3.1

B frame

bidirectional frame

frame (3.28) that is coded using motion compensated prediction from past or future *reference frames* (3.53) in *output order* (3.40)

3.2

backward prediction

process of predicting the current *frame* (3.28) by using future *frames* in an *output order* (3.40) as *reference frames* (3.53)

3.3

bin

bit of a *bin string* (3.4)

3.4

bin string

intermediate binary representation of values of *syntax elements* (3.65) resulting from the *binarization* (3.5) of the *syntax element*

3.5

binarization

set of *bin strings* ([3.4](#)) for all possible values of a *syntax element* ([3.65](#))

3.6

binarization process

unique mapping process of all possible values of a *syntax element* ([3.65](#)) onto a set of *bin strings* ([3.4](#))

3.7

bitstream

ordered series of bits that forms the *coded representation* ([3.14](#)) of the data

3.8

block

MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients* ([3.66](#))

3.9

byte

sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right, such that when represented in a sequence of data bits, the most significant bit of a byte is first

3.10

byte-aligned

positioning of a bit or *byte* ([3.9](#)) or *syntax element* ([3.65](#)) when the position at which it appears in a *bitstream* ([3.7](#)) is an integer multiple of 8 bits from the position of the first bit in the *bitstream*

3.11

byte stream

ordered series of bytes that forms the *coded representation* ([3.14](#)) of the data

3.12

chroma

sample array or single sample, identified symbolically by Cb or Cr, representing one of the two colour difference signals related to the primary colours

Note 1 to entry: The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

3.13

coded frame

coded representation ([3.14](#)) of a *frame* ([3.28](#))

3.14

coded representation

series of data elements as represented in coded form in the *bitstream* ([3.7](#))

3.15

component

array or single sample from one of the three arrays (*luma* ([3.37](#)) and two *chroma* ([3.12](#))) that make up a *frame* ([3.28](#)) in 4:2:0 colour format

3.16

DC coefficient

transform coefficient ([3.66](#)) for which the *frequency index* ([3.27](#)) is zero in all dimensions

3.17

decoded frame

frame ([3.28](#)) derived by decoding a *coded frame* ([3.13](#))

3.18**decoder**

embodiment of the *decoding process* ([3.20](#))

3.19**decoding order**

order in which syntax elements are processed by the *decoding process* ([3.20](#))

3.20**decoding process**

process that derives *decoded frames* ([3.17](#)) from the syntax elements in the *bitstream* ([3.7](#))

3.21**dequantization**

process of *scaling* ([3.57](#)) the *quantized transform coefficients* ([3.49](#)) after their representation in the *bitstream* ([3.7](#)) has been *parsed* ([3.42](#)) and before they are presented to the *inverse transform* ([3.34](#)) part of the *decoding process* ([3.20](#))

3.22**encoder**

embodiment of an *encoding process* ([3.23](#))

3.23**encoding process**

process that produces a *bitstream* ([3.7](#))

Note 1 to entry: This document does not specify an encoding process.

3.24**forbidden**

specification that a value shall never be used

Note 1 to entry: This is usually to avoid emulation of a *start code* ([3.63](#)) pattern.

3.25**forward prediction**

process of predicting the current frame by the past *reference frames* ([3.53](#)) in output order

3.26**flag**

binary variable that can take one of the two possible values, 0 and 1

3.27**frequency index**

one-dimensional or two-dimensional index associated with a *transform coefficient* ([3.66](#)) prior to an *inverse transform* ([3.34](#)) part of a *decoding process* ([3.20](#))

3.28**frame**

successive lines, numbered from the top-most line to the bottom-most line, containing samples numbered from the left-most sample to the right-most sample, representing the spatial information of a video signal from a single time instant

3.29**I frame****intra frame**

frame ([3.28](#)) coded using information only from itself

3.30**inter macroblock**

macroblock ([3.38](#)) which is coded using *inter prediction* ([3.31](#))

3.31

inter prediction

prediction (3.44) derived from data elements (e.g. sample value or *motion vector* (3.39)) of *reference frames* (3.53) other than the current frame

3.32

intra macroblock

macroblock (3.38) which is coded using *intra prediction* (3.33)

3.33

intra prediction

prediction (3.44) derived from only data elements (e.g. sample values) of the same decoded *slice* (3.60)

3.34

inverse transform

part of the *decoding process* (3.20) by which a set of *transform coefficients* (3.66) are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC coefficients* (3.16)

3.35

layer

one of a set of syntactical structures in a non-branching hierarchical relationship, such that higher layers contain lower layers, with such coded layers being the *coded frame* (3.13), *slice* (3.60), *macroblock* (3.38) and *block* (3.8)

3.36

level

defined set of constraints on the values that may be taken by *syntax elements* (3.65) and variables; or in a different context, the value of a *transform coefficient* (3.66) prior to *scaling* (3.57)

Note 1 to entry: The same set of levels is defined for all *profiles* (3.47), with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*.

3.37

luma

sample array or single sample, identified symbolically by Y or L, ordinarily representing the brightness signal related to the primary colours

Note 1 to entry: The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

3.38

macroblock

16 × 16 *luma* (3.37) sample value block and its corresponding two *chroma* (3.12) sample value blocks

3.39

motion vector

two-dimensional vector used for *inter prediction* (3.31) that provides an offset from the coordinates in the *decoded frame* (3.17) to the coordinates in a *reference frame* (3.53)

3.40

output order

order in which the *decoded frames* (3.17) are output from the *decoded frame buffer* in case the *decoded frames* are to be output from the *decoded frame buffer*

3.41

P frame

predictive frame

frame (3.28) that is coded using motion compensated prediction from past *reference frames* (3.53) in *output order* (3.40)

3.42**parse**

procedure of obtaining the value of a *syntax element* ([3.65](#)) from a *bitstream* ([3.7](#))

3.43**partitioning**

division of a set into subsets such that each element of a set is in exactly one of the subsets

3.44**prediction**

embodiment of a *prediction process* ([3.45](#))

3.45**prediction process**

use of a *predictor* ([3.46](#)) to provide an estimate of a data element (e.g. sample value or *motion vector* ([3.39](#))) currently being decoded

3.46**predictor**

combination of specified values or previously decoded data elements (e.g. sample value or *motion vector* ([3.39](#))) used in the *decoding process* ([3.20](#)) of subsequent data elements

3.47**profile**

specified subset of the syntax

3.48**quantization parameter**

variable used by the *decoding process* ([3.20](#)) for *scaling* ([3.57](#)) of *transform coefficient levels* ([3.67](#))

3.49**quantized transform coefficients**

transform coefficients ([3.66](#)) before *dequantization* ([3.21](#))

3.50**random access**

starting the *decoding process* ([3.20](#)) for part of a *bitstream* at some point other than the beginning of the *bitstream* ([3.7](#))

3.51**raster scan**

mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the top-most row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc., top-most rows of the pattern (proceeding downwards), with each row scanned from left to right

3.52**reference index**

order indication of the *reference frames* ([3.53](#)) in the frame buffer in the *decoding process* ([3.20](#))

3.53**reference frame**

frame that contains samples that may be used for *inter prediction* ([3.31](#)) in the *decoding process* ([3.20](#)) of subsequent *frames* ([3.28](#)) in *decoding order* ([3.19](#))

3.54**reserved**

specification that some values of a particular *syntax element* ([3.65](#)) are for future use by ISO/IEC, such that these values shall not be used in *bitstreams* ([3.7](#)), but may be specified for use in future extensions by ISO/IEC

3.55

residual

decoded difference between a *prediction* ([3.44](#)) of a sample or data element and its decoded value

3.56

run

number of data elements with the same value or the same treatment in the *decoding process* ([3.20](#))

Note 1 to entry: In one context, it means the number of zero coefficients before a non-zero coefficient in the block scan, and in another context, it means the number of consecutive *skipped macroblocks* ([3.59](#)).

3.57

scaling

process of multiplying *transform coefficient levels* ([3.67](#)) by a factor, resulting in *transform coefficients* ([3.66](#))

3.58

sequence

highest layer syntax structure of the *bitstream* ([3.7](#)), including one or more consecutive *coded frames* ([3.13](#))

3.59

skipped macroblock

macroblock ([3.38](#)) for which no syntax elements are present in the *bitstream* ([3.7](#)) except for the indication that the *macroblock* is a *skipped macroblock* ([3.59](#))

3.60

slice

integer number of consecutive *macroblock* ([3.38](#)) rows in the *raster scan* ([3.51](#)) order that is associated with the same header data

3.61

slice header

part of a coded *slice* ([3.60](#)) containing the data elements pertaining to the first or all *macroblocks* ([3.58](#)) represented in a *slice*

3.62

source

video material or some of its attributes before operation of an *encoding process* ([3.23](#))

3.63

start code

32-bit codeword pattern which is unique in the whole *bitstream* ([3.7](#))

Note 1 to entry: Start codes can be used to identify the starting point of a syntax structure in the *bitstream* (e.g. to enable *random access* ([3.50](#))).

3.64

stuffing bits

bit string having a prescribed pattern of fixed values at a particular position in the *bitstream* ([3.7](#))

3.65

syntax element

element of data represented in the *bitstream* ([3.7](#))

3.66

transform coefficient

scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* ([3.27](#)) in an *inverse transform* ([3.34](#)) part of the *decoding process* ([3.20](#))

3.67**transform coefficient level**

integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* (3.20) prior to *scaling* (3.57) for computation of a *transform coefficient* value (3.66)

3.68**video buffering verifier**

hypothetical reference *decoder* (3.18) that operates on the *bitstream* (3.7) to perform the *decoding process* (3.20) with a specified timing and with a specified limited capacity for buffering the coded data and *decoded frames* (3.17)

Note 1 to entry: Its purpose is to provide a constraint on the variability of the data rate that an *encoder* (3.22) or editing process may produce.

4 Abbreviations

LSB	least significant bit
MB	macroblock
MSB	most significant bit
VBV	video buffering verifier

5 Conventions

NOTE The mathematical operators and their precedence rules used in this document are similar to those used in the C programming language. However, operators of integer divisions with truncation and of rounding are specifically defined. If not specifically explained, numbering and counting begin from zero.

5.1 Arithmetic operators

+	Addition
–	Subtraction (as a binary operator) or negation (as a unary prefix operator)
*	Multiplication
a^b	Exponential operation: a is raised to power of b. (May alternatively represent a super-script.)
/	Integer division with truncation of the result toward zero. For example, 7/4 and (–7)/(–4) are truncated to 1 and (–7)/4 and 7/(–4) are truncated to –1.
÷	Division in mathematical formulae where no truncation or rounding is intended.
$\frac{a}{b}$	Division in mathematical formulae where no truncation or rounding is intended.
$\sum_{i=a}^b f(i)$	The summation of $f(i)$ with i taking integral values from a up to and including b.
$a \% b$	Remainder of a divided by b, defined only for $a \geq 0$ and $b > 0$.

5.2 Logical operators

- a && b Logical AND operation between a and b
- a || b Logical OR operation between a and b
- ! Logical NOT operation
- a ? b : c If a is TRUE or not equal to 0, evaluates to b; otherwise, evaluates to c.

5.3 Relational operators

- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- == equal to
- != not equal to

5.4 Bitwise operators

- & AND operation
- | OR operation
- ~ Negation operation
- a >> b Shift a in 2's complement binary integer representation format to the right by b bit positions. This operator is only defined with b, a positive integer.
- a << b Shift a in 2's complement binary integer representation format to the left by b bit positions. This operator is only defined with b, a positive integer.

5.5 Assignment

- = Assignment operator
- ++ Increment, x++ is equivalent to x = x + 1. When this operator is used for an array index, the variable value is obtained before the increment operation.
- Decrement, i.e. x-- is equivalent to x = x - 1. When this operator is used for an array index, the variable value is obtained before the decrement operation.
- += Addition assignment operator, for example, x += 3 corresponds to x = x + 3, x += (-3) is equivalent to x = x + (-3).
- = Subtraction assignment operator, for example, x -= 3 corresponds to x = x - 3, x -= (-3) is equivalent to x = x - (-3).

5.6 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

- operations of a higher precedence are evaluated before any operation of a lower precedence;
- operations of the same precedence are evaluated sequentially from left to right.

[Table 1](#) specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE For those operators that are also used in the C programming language, the order of precedence used in this document is the same as used in the C programming language.

Table 1 — Operation precedence from highest (at top of table) to lowest (at bottom of table)

operations (with operands x, y, and z)
"x++", "x--"
"!x", "-x" (as a unary prefix operator)
"xy"
"x * y", "x / y", "x ÷ y", " $\frac{x}{y}$ ", "x % y"
"x + y", "x - y" (as a two-argument operator), " $\sum_{i=a}^b f(i)$ "
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x == y", "x != y"
"x & y"
"x y"
"x && y"
"x y"
"x ? y : z"
"x = y", "x += y", "x -= y"

5.7 Mathematical functions

$$\text{abs}(x) = \begin{cases} x & ; x \geq 0 \\ -x & ; x < 0 \end{cases}$$

$\text{ceil}(x)$ Takes the smallest integer not smaller than x

$\text{clip1}(x) = \text{clip3}(0, 255, x)$

$$\text{clip3}(a, b, c) = \begin{cases} a & ; c < a \\ b & ; c > b \\ c & ; \text{else} \end{cases}$$

$\text{floor}(x)$ Takes the biggest integer not bigger than x

$\log_2(x)$	Logarithm number of x with base 2
$\log_{10}(x)$	Logarithm number of x with base 10
$\text{median}(x, y, z) =$	$x + y + z - \min(x, \min(y, z)) - \max(x, \max(y, z))$
$\min(x, y) =$	$\begin{cases} x & ; x \leq y \\ y & ; x > y \end{cases}$
$\max(x, y) =$	$\begin{cases} x & ; x \geq y \\ y & ; x < y \end{cases}$
$\text{round}(x) =$	$\text{sign}(x) * \text{floor}(\text{abs}(x) + 0.5)$
$\text{sign}(x) =$	$\begin{cases} 0 & ; x \geq 0 \\ 1 & ; x < 0 \end{cases}$
InverseRasterScan $(a, b, c, d, e) =$	$\begin{cases} (a \% (d/b)) * b & ; e == 0 \\ (a / (d/b)) * b & ; e == 1 \end{cases}$

5.8 Variables, syntax elements and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lowercase letters with underscore characters), its one or two syntax categories, and one or two descriptors for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases the syntax tables may use the values of other variables derived from syntax element values. Such variables appear in the syntax tables, or text, named by a mixture of lowercase and uppercase letter and without any underscore characters. Variables starting with an uppercase letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an uppercase letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lowercase letter are only used within the subclause in which they are derived.

In some cases, “mnemonic” names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes “mnemonic” names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an uppercase letter and may contain more uppercase letters.

NOTE The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 5.11.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.7) are described by their names, which start with an uppercase letter, contain a mixture of lower and uppercase letters without any underscore character, and end with left and right parentheses including

zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix *s* at horizontal position *x* and vertical position *y* may be denoted either as *s*[*x*, *y*] or as *s*_{*y**x*}.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

5.9 Text description of logical operations

In the text, a statement of logical operations as would be described in pseudo-code as:

```
if( condition 0 )
    statement 0
else if( condition 1 )
    statement 1
...
else /* informative remark on remaining condition */
    statement n
```

may be described in the following manner:

```
... as follows / ... the following applies.
— If condition 0, statement 0
— Otherwise, if condition 1, statement 1
— ...
— Otherwise (informative remark on remaining condition), statement n
```

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ...". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described in pseudo-code as:

```
if( condition 0a && condition 0b )
    statement 0
else if( condition 1a || condition 1b )
    statement 1
...
else
    statement n
```

may be described in the following manner:

... as follows / ... the following applies.

- If all of the following conditions are true, statement 0
 - condition 0a
 - condition 0b
- Otherwise, if any of the following conditions are true, statement 1
 - condition 1a
 - condition 1b
- ...
- Otherwise, statement n

In the text, a statement of logical operations as would be described in pseudo-code as:

```
if( condition 0 )  
    statement 0  
if( condition 1 )  
    statement 1
```

may be described in the following manner:

```
When condition 0, statement 0  
When condition 1, statement 1
```

5.10 Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and uppercase variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lowercase variable explicitly specified as the input. Each process specification has explicitly specified an output. The output is a variable that can either be an uppercase variable or a lowercase variable.

When invoking a process, the assignment of variables is specified as follows.

- If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lowercase input or output variables of the process specification.
- Otherwise (the variables at the invoking and the process specification have the same name), assignment is implied.

In the specification of a process, a specific macroblock may be referred to by the variable name having a value equal to the address of the specific macroblock.

5.11 Description of bitstream syntax parsing process and decoding process

5.11.1 Method of describing bitstream syntax

The description style of the syntax is similar to C programming language.

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses.

NOTE An actual decoder would implement means for identifying entry points into the bitstream and means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified here.

[Table 2](#) lists examples of pseudo code used to describe the syntax. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

Table 2 — Examples of pseudo code

	Descriptor
/* A statement can be a syntax element with associated descriptor or can be an expression used to specify its existence, type, and value, as in the following examples */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A “while” structure specifies that the statement is to be evaluated repeatedly while the condition remains true. */	
while (condition)	
statement	
/* A “do ... while” structure executes the statement once, and then tests the condition. It repeatedly evaluates the statement while the condition remains true. */	
do	
statement	
while (condition)	
/* An “if ... else” structure tests the condition first. If it is true, the primary statement is evaluated. Otherwise, the alternative statement is evaluated. If the alternative statement is unnecessary to be evaluated, the “else” and corresponding alternative statement can be omitted. */	
if(condition)	
primary statement	
else	
alternative statement	
/* A “for” structure evaluates the initial statement at the beginning then tests the condition. If it is true, the primary and subsequent statements are evaluated until the condition becomes false. */	
for (initial statement; condition; subsequent statement)	
primary statement	

Parsing and decoding process are described using text and C-like pseudo language.

5.11.2 Syntax functions

Functions used for syntax description are explained in this section. It is assumed that the decoder has a bitstream position indicator. This bitstream position indicator locates the position of the bit that is going to be read right next. A function consists of its name and a sequence of parameters inside of parentheses. A function may not have any parameters.

`byte_aligned()`

The function `byte_aligned()` returns TRUE if the current position is on a byte boundary. Otherwise, it returns FALSE.

`next_bits( n )`

The function returns the next *n* bits from the bitstream, MSB first. The current bitstream position indicator is not changed. If the remaining number of bits to be read are less than *n*, it returns 0.

`byte_aligned_next_bits( n )`

If the current position of the bitstream is not byte-aligned, returns *n* bits beginning from the next byte-aligned position, MSB first. The current bitstream position indicator is not changed. If the current position of the bitstream is byte-aligned, returns *n* bits from the current position, MSB first. The current bitstream position is not changed. If the remaining number of bits to be read is less than *n*, it returns 0.

`next_start_code()`

The `next_start_code()` function locates the next start code. It is defined in [Table 3](#).

Table 3 — next_start_code() function

<code>next_start_code() {</code>	Descriptor
stuffing_bit	'1'
stuffing_bit	'1'
<code>while (! byte_aligned())</code>	
stuffing_bit	'0'
<code>while (next_bits(24) != '0000 0000 0000 0000 0000 0001')</code>	
stuffing_byte	'0000 0000'
<code>}</code>	

`is_end_of_slice()`

This function tests if the current position is at the end of the slice. The function's definition is shown in [Table 4](#).

Table 4 — Function's definition of the end of the slice

<code>is_end_of_slice () {</code>	Descriptor
<code>if(byte_aligned ()) {</code>	
<code>if(next_bits(32) == 0xc0000001)</code>	
<code>return TRUE /* end of slice */</code>	
<code>}</code>	
<code>else {</code>	
<code>if(((byte_aligned_next_bits(24) == 0x0000001) </code> <code>(byte_aligned_next_bits(32) == 0x80000001)) &&</code> <code>is_stuffing_pattern())</code>	
<code>return TRUE /* end of slice */</code>	

Table 4 (continued)

}	
return FALSE	
}	

is_stuffing_pattern()

This function tests whether the remaining bits of the current byte or the next byte (in case the current position is byte-aligned), are stuffing bits. The function's definition is shown in [Table 5](#).

Table 5 — Function's definition of stuffing bits

is_stuffing_pattern () {	descriptor
if((n == 7) && (next_bits(1) == 1)) /* n, in the range 0..7, is the bitstream position indicator in the current byte, when n is 0, the bitstream position indicator indicates the MSB of the current byte. */	
return TRUE	
else if((n < 7) && (next_bits(8-n) == ((1<< (7-n)) + (1<< (6-n))))	
return TRUE	
else	
return FALSE	
}	

read_bits(n)

This function returns n bits of the bitstream from the current position, MSB first. The bitstream position indicator advances n bits. If n is equal to 0, the function returns 0, and the bitstream position indicator does not move.

Syntax functions can be also used for describing parsing process and decoding process.

5.11.3 Syntax descriptors

The descriptors below specify the parsing process of syntax elements.

b(8)

A byte with arbitrary value (8 bits). The parsing process for this descriptor is specified by the return value of read_bits(8).

f(n)

A bit string with n bits. The parsing process is specified by the returned value of read_bits(n).

i(n)

Signed integer with n bits. In syntax table, if n is 'v', the number of bits is determined by other syntax elements. The parsing process is specified by the return value of read_bits(n), interpreted as two's complement representation with MSB first.

r(n)

A bit string with n bits equal to '0'. The parsing process is specified by the returned value of read_bits(n).

u(n)

Unsigned integer with n bits. In syntax table, if n is 'v', the number of bits is determined by other syntax elements. The parsing process is specified by the returned value of `read_bits(n)`, interpreted as two's complement representation with MSB first.

`ue(v)`

Unsigned integer Exp-Golomb(Exponential Golomb) coded syntax element with the first bit on left. The parsing process is specified in subclause [9.2](#).

5.11.4 Reserved, forbidden and marker bit

In the bitstream syntax defined by this document, the value of some syntax elements is marked as 'reserved' or 'forbidden'.

The term 'reserved', when used in the clauses specifying some values of a particular syntax element, is for future uses. These values shall not be used in the bitstreams conforming to this document, but may be used in future extensions or revisions of this document.

The term 'forbidden' specifies some values of syntax elements that shall not be used in the bitstreams conforming to this document. `marker_bit` specifies a bit with value '1'. `reserved_bits` specifies that some particular syntax elements are used for future extension of this document. The decoding process shall ignore these bits.

6 Source, coded, decoded and output data formats

6.1 Source

This document only deals with coding of progressive-scanned video sequences, and each picture in the video sequence is a frame. The sequence, at the output of the decoding process, consists of a series of reconstructed frames that are separated in time by a frame period which is the inverse of a specified frame rate.

A frame consists of three matrices of integer samples: a luma sample matrix (Y), and two chroma sample matrices (Cb and Cr).

Each element of each colour component matrix has an integer value.

6.2 Colour format

This document only deals with 4:2:0 colour format, in which the two chroma colour component matrices have half the number of samples of the corresponding luma colour component matrix both horizontally and vertically. The luma and chroma samples are positioned as shown in [Figure 1](#).

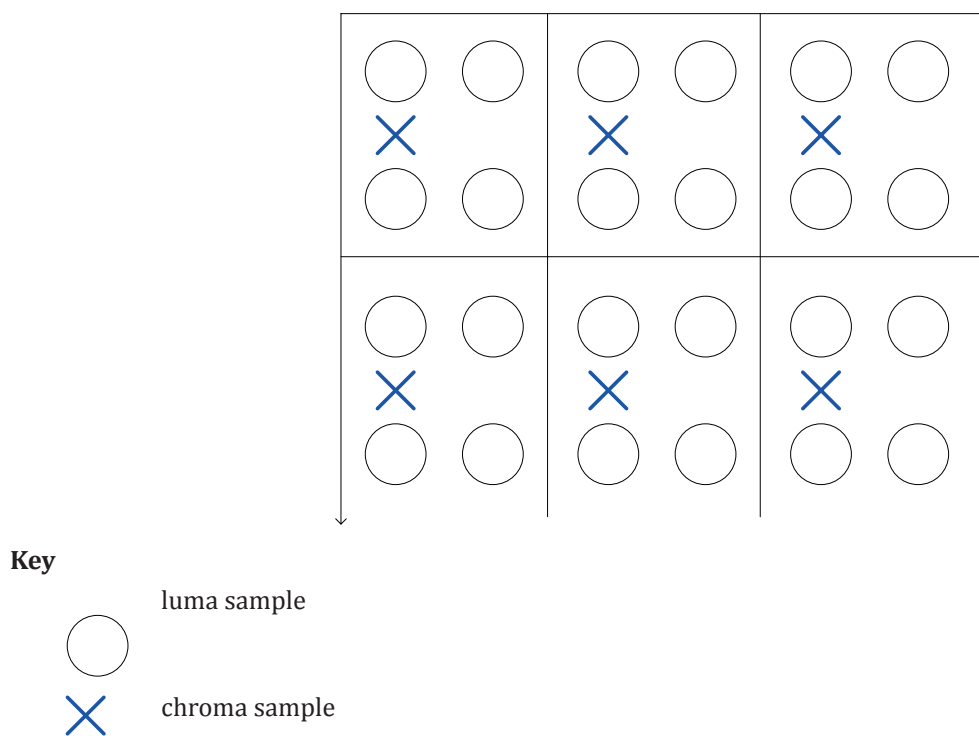


Figure 1 — Position of luma and chroma samples in 4:2:0 format

6.3 Coded bitstream format

The highest syntactic structure of the coded video bitstream is the video sequence. A video sequence commences with a sequence header which is followed by one or more coded frames. In front of each frame, a frame header is present. The order of the coded frames in the coded bitstream is the bitstream order. The bitstream order is same as the decoding order. The decoding order is not necessarily same as the output order. The video sequence is terminated by a sequence_end_code.

6.4 Sequence header

A video sequence header commences with sequence header start code and is followed by a series of coded frame data. A sequence header is allowed to be repeatedly present in the bitstream. This sequence header is called repeat sequence header. The main purpose of repeat sequence header is providing with random access functionality. The first coded frame after a sequence header shall be an I frame. The P frames after a sequence header only refer to frames appeared after the sequence header. It is a requirement of bitstream conformance that if the bitstream is modified by removing all of the data preceding any of the repeat sequence headers, then the resulting bitstream shall be a legal bitstream that conforms to this document.

6.5 Frame

A reconstructed frame is obtained by decoding a coded frame, i.e. a frame header, the optional extensions immediately following it, and the frame data.

6.6 Frame types

There are three types of frames that use different coding methods:

- an **Intra-coded (I) frame** is coded using information only from itself;

- a **Predictive-coded (P) frame** is a frame which is coded using motion compensated prediction from past reference frames;
- a **Bidirectionally predictive-coded (B) frame** is a frame which is coded using motion compensated prediction from past or future reference frames.

This document defines three sub-types of P frames, which can be used for P frame coding in low delay cases as shown in the [Table 6](#). A non-reference P frame is not used as a reference frame for motion compensated inter-frame prediction. A non-reference P frame with reference frame buffer (RPB) swapping is referred as a non-reference P frame accompanied with the operation of RPB swapping. After decoding a non-reference P frame with RPB swapping, the last two decoded frames placed in RPB shall exchange their positions in the buffer.

Table 6 — P frame sub-types

Name	Value
P frame	1
Non-reference P frame	2
Non-reference P frame with RPB swapping	3

6.7 Slice

A slice is a series of an arbitrary number of consecutive macroblocks. The first and last macroblocks of a slice shall not be skipped macroblocks, it is a requirement of bitstream conformance that mb_part_type is not equal to zero when the macroblock is at the first or last position in the slice. Every slice contains at least one macroblock. Slices shall not overlap. The position of slices may change from frame to frame. Slices shall occur in the bitstream in the order in which they are encountered, starting at the upper-left of the frame and proceeding by raster-scan order from left to right and top to bottom.

6.8 Macroblock

A slice is partitioned into macroblocks. A macroblock contains a section of the luma component and the spatially corresponding chroma components. The term macroblock can either refer to source and decoded data or to the corresponding coded data elements. A macroblock consists of 6 8x8 blocks. This structure holds 4 Y, 1 Cb and 1 Cr blocks and the block order is depicted in [Figure 2](#).

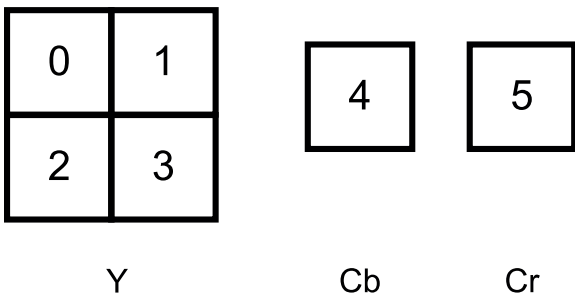


Figure 2 — Partitioning of a macroblock into 8x8 blocks (4:2:0 format)

6.9 Block

The term “block” can refer either to source and reconstructed data or to the transform coefficients or to the corresponding coded data elements.

When “block” refers to source and reconstructed data, it refers to an orthogonal section of a luma or chroma component with the same number of lines and samples. The size of a block can be either 4x4, 8x8 or 16x16.

6.10 Frame re-ordering

When the sequence contains coded B frames, the number of consecutive coded B frames is variable and shall be less than 127. The first coded frame after a sequence header shall not be a B frame.

The order of the coded frames in the bitstream, also called coded order, is the order in which a decoder reconstructs them. The order of the reconstructed frames at the output of the decoding process, also called the output order, is not always the same as the coded order and this subclause defines the rules of frame re-ordering between the decoder input and decoder output.

When the sequence contains no coded B frames, the coded order is the same as the output order. This is true in particular always when `low_delay` is one. When B frames are present in the sequence, re-ordering shall be performed to produce the output order according to the following rules:

- If the current frame in coded order is a B frame, the output frame is the frame reconstructed from that B frame;
- If the current frame in coded order is an I frame or P frame, the output frame is the frame reconstructed from the previous I frame or P frame if one exists. If none exists, at the start of the sequence, no frame is output.

The frame reconstructed from the final I frame or P frame is output immediately after the frame reconstructed when the last coded frame in the sequence was removed from the VBV buffer.

The following [Figure 3](#) is an example for explaining re-ordering: there are two coded B frames between successive coded P frames. The P frame with only intra coded blocks is marked as “I”. Frame ‘1I’ is used to form a prediction for frame ‘4P’. Frames ‘4P’ and ‘1I’ are both used to form predictions for frames ‘2B’ and ‘3B’. Therefore the order of coded frames in the coded sequence is ‘1I’, ‘4P’, ‘2B’, ‘3B’. However, the decoder outputs them in the order ‘1I’, ‘2B’, ‘3B’, ‘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

At the encoder output, in the coded bitstream, and at the decoder input:

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

At the decoder output:

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

Figure 3 — Frame re-ordering example

6.11 Reference frames

P frame can use up to eight (at maximum) past frames as reference. B frame can refer up to one forward reference frame or one backward reference frame.

6.12 Inverse scanning processes and derivation processes for neighbours

6.12.1 General

This subclause specifies inverse scanning processes, i.e., the mapping of indices to locations, and derivation processes for neighbours.

6.12.2 Inverse macroblock scanning process

Input to this process is a macroblock address mbAddr.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock with address mbAddr relative to the upper-left sample of the frame.

The inverse macroblock scanning process is specified as follows:

- x = InverseRasterScan(mbAddr, 16, 16, PicWidth, 0);
- y = InverseRasterScan(mbAddr, 16, 16, PicWidth, 1).

6.12.3 Inverse macroblock partition scanning process

Macroblocks may be partitioned, and the partitions are scanned for inter prediction as shown in Figure 4. The outer rectangles refer to the samples in a macroblock. The rectangles refer to the partitions. The number in each rectangle specifies the index of the inverse macroblock partition scan.

The functions MbPartWidth(), and MbPartHeight() describing the width and height of macroblock partitions are specified in Table 16, and Table 17. MbPartWidth() and MbPartHeight() are set to appropriate values for each macroblock partition, depending on the macroblock partition type (denoted by mb_part_type).

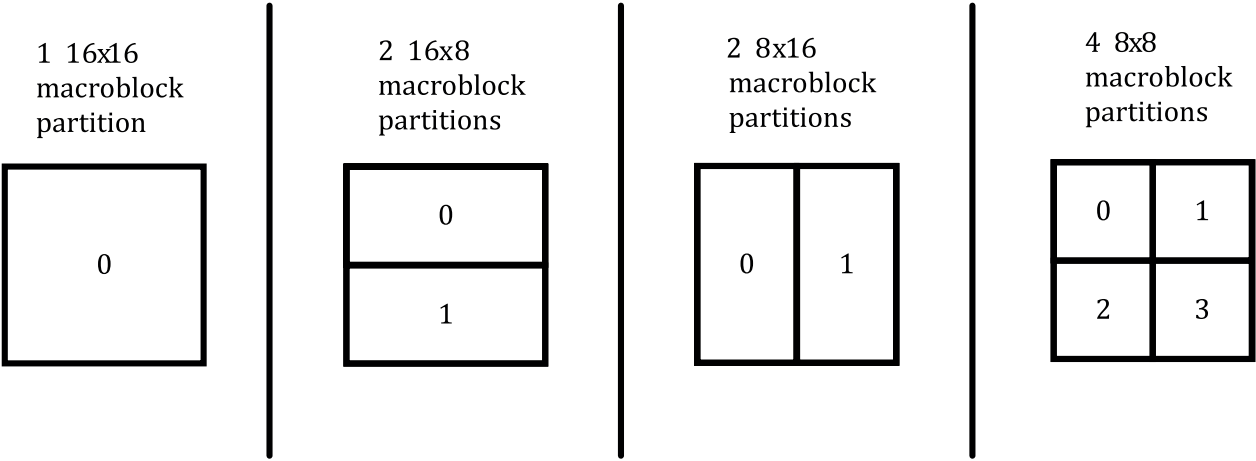


Figure 4 — Macroblock partitions

Input to this process is the index of a macroblock partition mbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock partition mbPartIdx relative to the upper-left sample of the macroblock.

The inverse macroblock partition scanning process is specified by:

- x = InverseRasterScan(mbPartIdx, MbPartWidth(mb_part_type), MbPartHeight(mb_part_type), 16, 0);

- $y = \text{InverseRasterScan}(\text{mbPartIdx}, \text{MbPartWidth}(\text{mb_part_type}), \text{MbPartHeight}(\text{mb_part_type}), 16, 1)$.

6.12.4 Inverse 8x8 luma block scanning process

Input to this process is the index of an 8x8 luma block luma8x8BlkIdx within a 16x16 luma block.

Output of this process is the location (x, y) of the upper-left luma sample for the 8x8 luma block with index luma8x8BlkIdx relative to the upper-left luma sample of the 16x16 luma block.

[Figure 5](#) shows the scan order for the 8x8 luma blocks.

0	1
2	3

Figure 5 — Scan order for 8x8 luma blocks

The inverse 8x8 luma block scanning process is specified as follows.

- $x = \text{InverseRasterScan}(\text{luma8x8BlkIdx}, 8, 8, 16, 0)$;
- $y = \text{InverseRasterScan}(\text{luma8x8BlkIdx}, 8, 8, 16, 1)$.

6.12.5 Inverse 4x4 luma block scanning process

Input to this process is the index of a 4x4 luma block luma4x4BlkIdx within an 8x8 luma block.

Output of this process is the location (x, y) of the upper-left luma sample for the 4x4 luma block with index luma4x4BlkIdx relative to the upper-left luma sample of the 8x8 block.

[Figure 6](#) shows the scan order for the 4x4 luma blocks.

0	1
2	3

Figure 6 — Scan order for 4x4 luma blocks

The inverse 4x4 luma block scanning process is specified as follows.

- $x = \text{InverseRasterScan}(\text{luma4x4BlkIdx}, 4, 4, 8, 0)$
- $y = \text{InverseRasterScan}(\text{luma4x4BlkIdx}, 4, 4, 8, 1)$

6.12.6 Derivation process of the availability for macroblock addresses

Input to this process is a macroblock address mbAddr .

Output of this process is the availability of the macroblock mbAddr.

NOTE The meaning of availability is determined when this process is invoked.

The macroblock is marked as available, unless one of the following conditions is true in which case the macroblock is marked as not available:

- $\text{mbAddr} < 0$;
- $\text{mbAddr} > \text{CurrMbAddr}$;
- the macroblock with address mbAddr belongs to a different slice than the current slice.

6.12.7 Derivation process for neighbouring macroblock addresses and their availability

The outputs of this process are:

- mbAddrA: the address and availability status of the macroblock to the left of the current macroblock;
- mbAddrB: the address and availability status of the macroblock above the current macroblock;
- mbAddrC: the address and availability status of the macroblock above-right of the current macroblock;
- mbAddrD: the address and availability status of the macroblock above-left of the current macroblock.

Figure 7 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

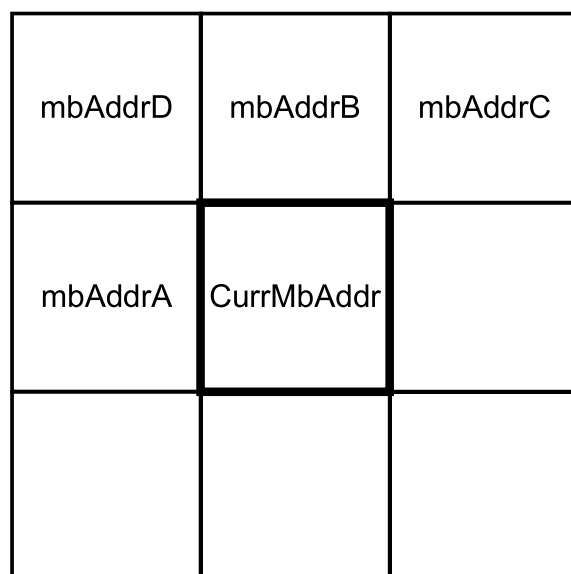


Figure 7 — Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.12.6 is $\text{mbAddrA} = \text{CurrMbAddr} - 1$ and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when $\text{CurrMbAddr} \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.12.6 is $\text{mbAddrB} = \text{CurrMbAddr} - \text{PicWidthInMbs}$ and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.12.6 is $\text{mbAddrC} = \text{CurrMbAddr} - \text{PicWidthInMbs} + 1$ and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when $(\text{CurrMbAddr} + 1) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.12.6 is $\text{mbAddrD} = \text{CurrMbAddr} - \text{PicWidthInMbs} - 1$ and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when $\text{CurrMbAddr} \% \text{PicWidthInMbs}$ is equal to 0.

6.12.8 Derivation processes for neighbouring macroblocks, blocks, and partitions

6.12.8.1 General

Subclause 6.12.8.2 specifies the derivation process for neighbouring macroblocks.

Subclause 6.12.8.3 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.12.8.4 specifies the derivation process for neighbouring partitions.

Table 8 specifies the values for the difference of luma location (x_D, y_D) for the input and the replacement for N in mbAddrN , mbPartIdxN , and luma8x8BlkIdxN for the output.

These input and output assignments are used in subclauses 6.12.8.2 to 6.12.8.4. The variable predPartWidth is specified when Table 7 is referred to.

Table 7 — Specification of input and output assignments for subclauses 6.12.8.2 to 6.12.8.4

N	x_D	y_D
A	-1	0
B	0	-1
C	predPartWidth	-1
D	-1	-1

Figure 8 illustrates the relative location of the neighbouring macroblocks, blocks, or partitions A, B, C, and D to the current macroblock, partition, or block.

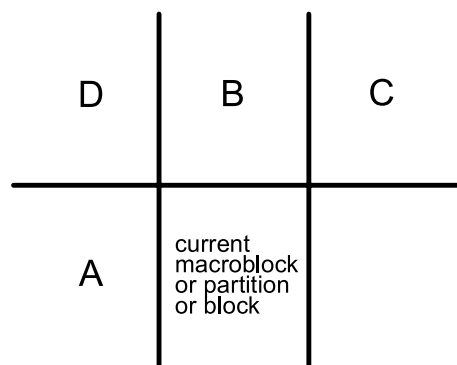


Figure 8 — Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.12.8.2 Derivation process for neighbouring macroblocks

Outputs of this process are:

- mbAddrA : the address of the macroblock to the left of the current macroblock and its availability status; and

- mbAddrB : the address of the macroblock above the current macroblock and its availability status.

mbAddrN (with N being A or B) is derived as follows.

- The difference of luma location (x_D, y_D) is set according to Table 7.

- The derivation process for neighbouring locations as specified in subclause 6.12.9 is invoked for luma locations with (x_N, y_N) equal to (x_D, y_D), and the output is assigned to mbAddrN.

6.12.8.3 Derivation process for neighbouring 8x8 luma block

Input to this process is an 8x8 luma block index luma8x8BlkIdx.

The luma8x8BlkIdx specifies the 8x8 luma blocks of a macroblock in a raster scan.

Outputs of this process are:

- mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status;
- luma8x8BlkIdxA: the index of the 8x8 luma block to the left of the 8x8 block with index luma8x8BlkIdx and its availability status;
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status;
- luma8x8BlkIdxB: the index of the 8x8 luma block above the 8x8 block with index luma8x8BlkIdx and its availability status.

mbAddrN and luma8x8BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (x_D, y_D) is set according to Table 7.
- The luma location (x_N, y_N) is specified by:
 - $x_N = (\text{luma8x8BlkIdx} \% 2) * 8 + x_D$;
 - $y_N = (\text{luma8x8BlkIdx} / 2) * 8 + y_D$.
- The derivation process for neighbouring locations as specified in subclause 6.12.9 is invoked for luma locations with (x_N, y_N) as the input and the output is assigned to mbAddrN and (x_W, y_W).
- The variable luma8x8BlkIdxN is derived as follows.
 - If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.
 - Otherwise (mbAddrN is available), the 8x8 luma block in the macroblock mbAddrN covering the luma location (x_W, y_W) is assigned to luma8x8BlkIdxN.

6.12.8.4 Derivation process for neighbouring partitions

Inputs to this process are

- a macroblock partition index mbPartIdx.

Outputs of this process are:

- mbAddrA\mbPartIdxA: specifying the macroblock partition to the left of the current macroblock and its availability status;
- mbAddrB\mbPartIdxB: specifying the macroblock partition above the current macroblock and its availability status;
- mbAddrC\mbPartIdxC: specifying the macroblock partition to the right-above of the current macroblock and its availability status;
- mbAddrD\mbPartIdxD: specifying the macroblock partition to the left-above of the current macroblock and its availability status.

$mbAddrN$ and $mbPartIdxN$ (with N being A, B, C, or D) are derived as follows.

- The inverse macroblock partition scanning process as described in subclause 6.12.3 is invoked with $mbPartIdx$ as the input and (x, y) as the output.
- The variable $predPartWidth$ in Table 7 is specified as follows.
 $predPartWidth = MbPartWidth(mb_part_type)$.
- The difference of luma location (xD, yD) is set according to Table 7.
- The neighbouring luma location (xN, yN) is specified by:
 - $xN = x + xD$;
 - $yN = y + yD$.
- The derivation process for neighbouring locations as specified in subclause 6.12.9 is invoked for luma locations with (xN, yN) as the input and the output is assigned to $mbAddrN$ and (xW, yW) .
- Depending on $mbAddrN$, the following applies.
 - If $mbAddrN$ is not available, the macroblock partition $mbAddrN \backslash mbPartIdxN$ is marked as not available.
 - Otherwise ($mbAddrN$ is available), the following applies.
 - The macroblock partition in the macroblock $mbAddrN$ covering the luma location (xW, yW) is assigned to $mbPartIdxN$.
 - When the partition given by $mbPartIdxN$ is not yet decoded, the macroblock partition $mbPartIdxN$ is marked as not available.

6.12.9 Derivation process for neighbouring locations

Input to this process is a luma or chroma location (xN, yN) expressed relative to the upper left corner of the current macroblock.

Outputs of this process are:

- $mbAddrN$: either equal to $CurrMbAddr$ or to the address of neighbouring macroblock that contains (xN, yN) and its availability status;
- (xW, yW) : the location (xN, yN) expressed relative to the upper-left corner of the macroblock $mbAddrN$ (rather than relative to the upper-left corner of the current macroblock).

Let $maxWH$ be a variable specifying a maximum value of the location components xN, yN, xW , and yW . $maxWH$ is derived as follows.

- If this process is invoked for neighbouring luma locations:
 $maxWH = 16$
- Otherwise (this process is invoked for neighbouring chroma locations):
 $maxWH = 8$

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.12.7 is invoked with $mbAddrA, mbAddrB, mbAddrC$, and $mbAddrD$ as well as their availability status as the output.

Table 8 specifies $mbAddrN$ depending on (xN, yN) .