INTERNATIONAL STANDARD

ISO/IEC 23001-11

Second edition
2019-03

Information technology — MPEG systems technologies —

Part 11: Energy-efficient media consumption (green metadata)

Technologies de l'information — Technologies des systèmes MPEG —

Partie 11: Consommation des supports éconergétiques (métadonnées vertes)

Reference number
ISO/IEC 23001-11:2019(E)

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

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.

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) or the IEC list of patent declarations received (see http://patents.iec.ch).

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 Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 23001-11:2015), which has been technically revised. It also incorporates the Amendments ISO/IEC 23001-11:2015/Amd 1:2016 and ISO/IEC 23001-11:2015/Amd 2:2018. The main changes compared to the previous edition are as follows:

- specification of an HEVC SEI message carrying green metadata and modification of text specifying the carriage of green Metadata in an AVC SEI message so that the AVC and HEVC SEI messages are consistent;
- inclusion of Annex C which specifies conformance-verification procedures for the power-reduction technologies specified in this document, precises the role of the reference software for each technology and gives the links to reference softwares and test vectors.
- specification of HEVC Complexity metrics and improvement of the existing AVC Complexity metrics.

A list of all parts in the ISO/IEC 23001 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 the metadata (green metadata) that facilitates reduction of energy usage during media consumption as follows:

- the format of the metadata that enables reduced decoder power consumption;
- the format of the metadata that enables reduced display power consumption;
- the format of the metadata that enables media selection for joint decoder and display power reduction;
- the format of the metadata that enables quality recovery after low-power encoding.

This metadata facilitates reduced energy usage during media consumption without any degradation in the quality of experience (QoE). However, it is also possible to use this metadata to get larger energy savings, but at the expense of some QoE degradation.

Information technology — MPEG systems technologies —

Part 11: Energy-efficient media consumption (green metadata)

1 Scope

This document specifies metadata for energy-efficient decoding, encoding, presentation and selection of media.

The metadata for energy-efficient decoding specifies two sets of information: complexity metrics (CM) metadata and decoding operation reduction request (DOR-Req) metadata. A decoder uses CM metadata to vary operating frequency and thus reduce decoder power consumption. In a point-to-point video conferencing application, the remote encoder uses the DOR-Req metadata to modify the decoding complexity of the bitstream and thus reduce local decoder power consumption.

The metadata for energy-efficient encoding specifies a quality metric that is used by a decoder to reduce the quality loss from low-power encoding.

The metadata for energy-efficient presentation specifies RGB-component statistics and quality levels. A presentation subsystem uses this metadata to reduce power by adjusting display parameters, based on the statistics, to provide a desired quality level from those provided in the metadata.

The metadata for energy-efficient media selection specifies decoder operation reduction ratios (DOR-Ratios), RGB-component statistics and quality levels. The client in an adaptive streaming session uses this metadata to determine decoder and display power-saving characteristics of available video representations and to select the representation with the optimal quality for a given power-saving.

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.

ISO/IEC 13818-1, *Information technology — Generic coding of moving pictures and associated audio information — Part 1: Systems*

ISO/IEC 14496-10:—[1], *Information technology — Coding of audio-visual objects — Part 10: Advanced video coding*

ISO/IEC 23001-10, *Information technology — MPEG systems technologies — Part 10: Carriage of timed metadata metrics of media in ISO base media file format*

ISO/IEC 23008-2, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding*

ISO/IEC 23009-1:—[2], *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*

ISO/IEC/TR 23009-3, *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 3: Implementation guidelines*

[1] Under preparation. Stage at the time of publication: ISO/IEC DIS 14496-10:2018.

[2] Under preparation. Stage at the time of publication: ISO/IEC FDIS 23009-1:2019.

3 Terms, definitions, symbols, abbreviated terms and conventions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-10, ISO/IEC 23008-2 and ISO/IEC 23009-1 and the following apply.

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1 Terms and definitions

3.1.1
alpha-point deblocking instance
APDI
single filtering operation that produces either a single, filtered output p'_0 or a single, filtered output q'_0, where p'_0 and q'_0 are filtered samples across a 4x4 block edge

3.1.2
chroma_format_idc
chroma sampling relative to the luma sampling

3.1.3
deblocking filtering instance
single filtering operation that produces either a single, filtered output p' or a single, filtered output q', where p' and q' are filtered samples across a 8x8 block edge

3.1.4
decoding process
process that reads a bitstream and derives decoded pictures from it

Note 1 to entry: This process is specified in ISO/IEC 14496-10 or ISO/IEC 23008-2.

3.1.5
display process
process that takes, as its input, the cropped decoded pictures that are the output of the *decoding process* (3.1.4)

3.1.6
encoder
embodiment of an *encoding process* (3.1.7)

3.1.7
encoding process
process that produces a bitstream

Note 1 to entry: The bitstream produced is conforming to ISO/IEC 14496-10 or ISO/IEC 23008-2.

3.1.8
no-quality-loss operating point
NQLOP
metadata-enabled operating point associated with the largest display-power reduction that can be achieved without any quality loss (infinite PSNR)

3.1.9
non-zero block
block containing at least one non-zero transform coefficient

3.1.10
peak signal
maximum permissible *RGB component* (3.1.16) in a *reconstructed frame* (3.1.14)

Note 1 to entry: For 8-bit video, the peak signal is 255.

3.1.11
period
interval over which complexity-metrics metadata are applicable

3.1.12
PicSizeInMbs
product of the picture width and the picture height in units of macroblocks

3.1.13
pixel
smallest addressable element in an all-points addressable display device

3.1.14
reconstructed frames
frames obtained after applying *RGB colour-space* (3.1.15) conversion and cropping to the specific decoded picture or pictures for which display power-reduction metadata are applicable

3.1.15
RGB colour space
colour space based on the red, green and blue primaries

3.1.16
RGB component
single sample representing one of the three primary colours of the *RGB colour space* (3.1.15)

3.1.17
separate_colour_plane_flag
flag that, when set, specifies that the three colour components of the 4:4:4 chroma format are coded separately

3.1.18
six-tap filtering
STF
single application of the 6-tap filter to generate a single filtered sample for fractional positions using the samples at integer-sample positions

3.2 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in the following apply:

APDI	alpha-point deblocking instance
ASIC	application specific integrated circuit
AVC	advanced video coding — ISO/IEC 14496-10
BMFF	base media file format
CM	complexity metric
CMOS	complementary metal oxide semiconductor
CPU	central processing unit

DASH	dynamic adaptive streaming over HTTP
DOR-Ratio	decoding operation reduction ratio
DOR-Req	decoding operation reduction request
DVFS	dynamic voltage frequency scaling
Fps	frames per second
FS	fresh start
GP	good picture
HEVC	high efficiency video coding — ISO/IEC 23008-2
Mbps	mega bits per second
MPD	media presentation description
MSD	mean square difference
MV	motion vector
NQLOP	no-quality-loss operating point
PSNR	peak signal to noise ratio
QoE	quality of experience
RBLL	remaining battery life level
RGB	red, green, blue
SEI	supplemental enhancement information
SP	start picture
STF	six-tap filtering
XSD	cross-segment decoding

3.3 Conventions

3.3.1 Arithmetic operators

+	Addition
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	Multiplication
x^y	Exponentiation

x/y Division where no truncation or rounding is intended

$\frac{x}{y}$ Division where no truncation or rounding is intended

$\sum_{i=x}^{y} f(i)$ Summation of *f*(*i*) with *i* taking all integer values from *x* up to and including *y*

3.3.2 Mathematical functions

Mathematical functions in this document are defined as follows:

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

$$\text{Clip}(x)=\begin{cases}x, & x<256\\ 255, & \text{otherwise}\end{cases} \tag{2}$$

Floor(*x*) is the greatest integer less than or equal to *x* (3)

Log10(*x*) returns the base-10 logarithm of *x* (4)

Round(*x*) = Sign(*x*) * Floor(Abs(*x*) + 0.5) (5)

$$\text{Sign}(x)=\begin{cases}-1, & x<0\\ 1, & x\leq 0\end{cases} \tag{6}$$

4 Functional architecture

This clause is informative and placed here to provide context.

4.1 Description of the functional architecture

Figure 1 shows the functional architecture utilizing green metadata in this document. The media pre-processor is applied to analyse and to filter the content source and a video encoder is used to encode the content to a bitstream for delivery. The bitstream is delivered to the receiver and decoded by a video decoder with the output rendered on a presentation subsystem that implements a display process.

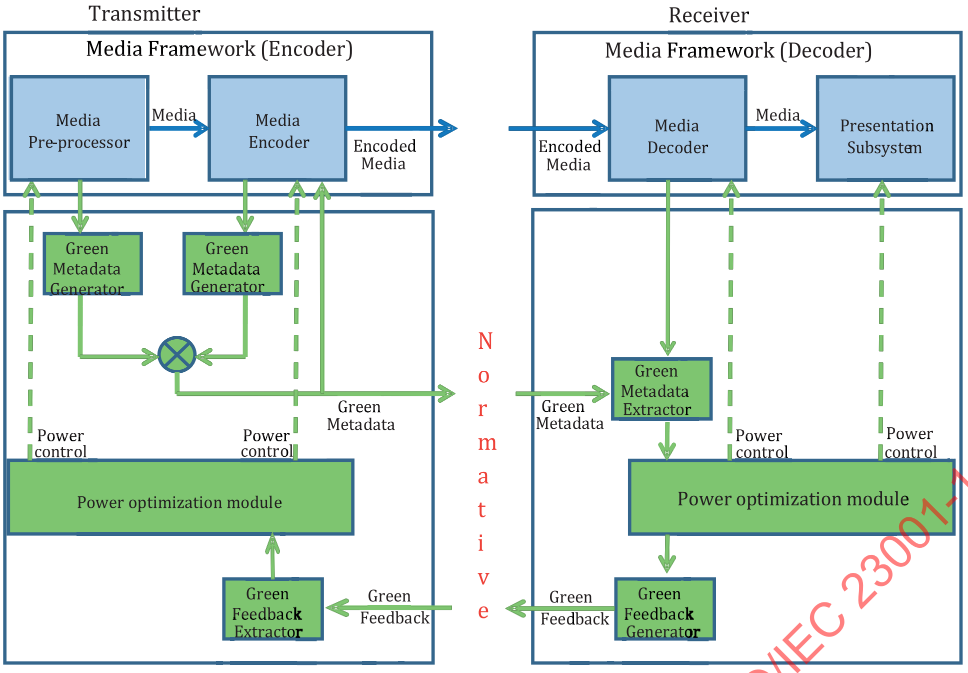

Figure 1 — Functional architecture

The green metadata is extracted from either the media encoder or the media pre-processor. In both cases, the green metadata is multiplexed or encapsulated in the conformant bitstream. Such green metadata is used at the receiver to reduce the power consumption for video decoding and presentation. The bitstream is packetized and delivered to the receiver for decoding and presentation. At the receiver, the metadata extractor processes the packets and sends the green metadata to a power optimization module for efficient power control. For instance, the power optimization module interprets the green metadata and then applies appropriate operations to reduce the video decoder's power consumption when decoding the video and also to reduce the presentation subsystem's power consumption when rendering the video. In addition, the power-optimization module can collect receiver information, such as remaining battery capacity, and send it to the transmitter as green feedback to adapt the encoder operations for power-consumption reduction.

The normative aspect of this document is limited to the green metadata and green feedback in Figure 1.

4.2 Definition of components in the functional architecture

Green metadata generator

— Generates metadata from either the video encoder or the content pre-processor.

Green metadata extractor

— Interprets the bitstream syntax information and sends it to the power optimization module in the receiver.

Green feedback generator

— Generates feedback information for the transmitter.

— Communicates with the transmitter through a feedback channel, if available, for energy-efficient processing.

Green feedback extractor

- Receives the feedback from the receiver and sends it to the power optimization module in the transmitter.

Power optimization module in the transmitter

- Collects platform statistics such as the remaining battery capacity of the device in which the transmitter resides.
- Controls the operation of the green metadata generator, video encoder and content pre-processor.
- Processes green feedback.

Power optimization module in the receiver

- Processes the green-metadata information and applies appropriate operations for power-consumption control.
- Collects platform statistics such as remaining battery capacity of the device in which the receiver resides.
- Sends requests to Green feedback generator.

5 Decoder power reduction

5.1 General

Energy-efficient decoding is achieved with two types of metadata: complexity metrics (CMs) metadata and decoding operation reduction request (DOR-Req) metadata. A decoder may use CMs metadata to vary operating frequency and thus reduce decoder power consumption. In a point-to-point video conferencing application, the remote encoder may use the DOR-Req metadata to modify the decoding complexity of the bitstream and thus reduce local decoder power consumption.

5.2 Complexity metrics for decoder-power reduction

5.2.1 General

With respect to the functional architecture in Figure 1, the green-metadata generator provides CMs that indicate the picture-decoding complexity of an AVC or HEVC bitstream to the decoder.

5.2.2 Syntax

The syntax for the AVC CMs is given in Table 1.

Table 1 — Syntax for the AVC CMs

	Size (bits)	Descriptor
period_type	8	unsigned integer
if (period_type = = 2) || (period_type == 7) {		
num_seconds	16	unsigned integer
}		
else if (period_type = = 3) || (period_type == 8) {		
num_pictures	16	unsigned integer
}		
if (period_type == 8) {		

Table 1 *(continued)*

temporal_map		
for (t=0; t<8; t++) {		
if ((temporal_map>>t)%2 == 1)		
num_pictures_in_temporal_layers[t]		
}		
}		
if (period_type <= 3) {		
portion_non_zero_8x8_blocks	8	unsigned integer
portion_intra_predicted_macroblocks	8	unsigned integer
portion_six_tap_filterings	8	unsigned integer
portion_alpha_point_deblocking_instances	8	unsigned integer
}		
else if (period_type == 4) {		
for (i=0; i<= num_slice_groups_minus1; i++) {		
num_slices_minus1[i]	16	unsigned integer
}		
for (i=0; i<= num_slice_groups_minus1; i++) {		
for (j=0; j<=num_slices_minus1[i]; j++) {		
first_mb_in_slice[i][j]	16	unsigned integer
portion_non_zero_8x8_blocks[i][j]	8	unsigned integer
portion_intra_predicted_macroblocks[i][j]	8	unsigned integer
portion_six_tap_filterings[i][j]	8	unsigned integer
portion_alpha_point_deblocking_instances[i][j]	8	unsigned integer
}		
}		
}		
else if (period_type >= 5) && (period_type <= 8) {		
num_layers_minus1	16	unsigned integer
for (l=0; l<= num_layers_minus1; l++) {		
picture_parameter_set_id[l]	8	unsigned integer
priority_id[l]	6	unsigned integer
dependency_id[l]	3	unsigned integer
quality_id[l]	4	unsigned integer
temporal_id[l]	3	unsigned integer
portion_non_zero_8x8_blocks[l]	8	unsigned integer
portion_intra_predicted_macroblocks[l]	8	unsigned integer
portion_six_tap_filterings[l]	8	unsigned integer
portion_alpha_point_deblocking_instances[l]	8	unsigned integer
}		
}		

The syntax for the HEVC CMs is given in Table 2.

Table 2 — Syntax for the HEVC CMs

	Size (bits)	Descriptor
period_type	8	unsigned integer
if (period_type == 2) {		
num_seconds	16	unsigned integer
}		
else if (period_type == 3) {		
num_pictures	16	unsigned integer
}		
if (period_type <= 3) {		
portion_non_zero_blocks_area	8	unsigned integer
if (portion_non_zero_blocks_area != 0) {		
portion_8x8_blocks_in_non_zero_area	8	unsigned integer
portion_16x16_blocks_in_non_zero_area	8	unsigned integer
portion_32x32_blocks_in_non_zero_area	8	unsigned integer
}		
portion_intra_predicted_blocks_area	8	unsigned integer
if (portion_intra_predicted_blocks_area == 255) {		
portion_planar_blocks_in_intra_area	8	unsigned integer
portion_dc_blocks_in_intra_area	8	unsigned integer
portion_angular_hv_blocks_in_intra_area	8	unsigned integer
}		
else {		
portion_blocks_a_c_d_n_filterings	8	unsigned integer
portion_blocks_h_b_filterings	8	unsigned integer
portion_blocks_f_i_k_q_filterings	8	unsigned integer
portion_blocks_j_filterings	8	unsigned integer
portion_blocks_e_g_p_r_filterings	8	unsigned integer
}		
portion_deblocking_instances	8	unsigned integer
}		
else if (period_type == 4) {		
max_num_slices_tiles_minus1	16	unsigned integer
for (t=0; t<=max_num_slices_tiles_minus1; t++) {		
first_ctb_in_slice_or_tile[t]	16	unsigned integer
portion_non_zero_blocks_area[t]	8	unsigned integer
if (portion_non_zero_blocks_area[t] != 0) {		
portion_8x8_blocks_in_non_zero_area[t]	8	unsigned integer
portion_16x16_blocks_in_non_zero_area[t]	8	unsigned integer
portion_32x32_blocks_in_non_zero_area[t]	8	unsigned integer
}		
portion_intra_predicted_blocks_area[t]	8	unsigned integer
if (portion_intra_predicted_blocks_area[t] == 255) {		
portion_planar_blocks_in_intra_area[t]	8	unsigned integer
portion_dc_blocks_in_intra_area[t]	8	unsigned integer
portion_angular_hv_blocks_in_intra_area[t]	8	unsigned integer

Table 2 *(continued)*

}		
else {		
portion_blocks_a_c_d_n_filterings[t]	8	unsigned integer
portion_blocks_h_b_filterings[t]	8	unsigned integer
portion_blocks_f_i_k_q_filterings[t]	8	unsigned integer
portion_blocks_j_filterings[t]	8	unsigned integer
portion_blocks_e_g_p_r_filterings[t]	8	unsigned integer
}		
portion_deblocking_instances[t]	8	unsigned integer
}		
}		

5.2.3 Signalling

SEI messages can be used to signal complexity metrics metadata in an AVC or HEVC stream. The green metadata SEI message payload type shall be set in accordance with ISO/IEC 14496-10 and ISO/IEC 23008-2. The complete syntax of the green metadata SEI message payload shall be in accordance with Annex A.

The message containing the CMs is transmitted at the start of an upcoming period. The next message containing CMs is transmitted at the start of the next upcoming period. Therefore, when the upcoming period is a picture or the interval up to the next I-slice, a message is transmitted for each picture or interval, respectively. However, when the upcoming period is a specified time interval or a specified number of pictures, the associated message is transmitted with the first picture in the time interval or with the first picture in the specified number of pictures.

5.2.4 Semantics

5.2.4.1 AVC semantics

The semantics of various terms are defined below.

period_type specifies the type of upcoming period over which the four complexity metrics are applicable and is defined in Table 3.

Table 3 — Value and description of period_type

Value	Description
0x00	complexity metrics are applicable to a single picture
0x01	complexity metrics are applicable to all pictures in decoding order, up to (but not including) the picture containing the next I slice
0x02	complexity metrics are applicable over a specified time interval in seconds
0x03	complexity metrics are applicable over a specified number of pictures counted in decoding order
0x04	complexity metrics are applicable to a single picture with slice granularity
0x05	complexity metrics are applicable to a single picture with scalable layer granularity
0x06	complexity metrics are applicable to all pictures in decoding order, up to (but not including) the picture containing the next I slice in the base layer with scalable layer granularity

Table 3 *(continued)*

Value	Description
0x07	complexity metrics are applicable over a specified time interval in seconds with scalable layer granularity
0x08	complexity metrics are applicable over a specified number of pictures counted in decoding order with scalable layer granularity
0x09–0xFF	user-defined

num_seconds indicates the number of seconds over which the complexity metrics are applicable when period_type is 2 or 7.

num_pictures indicates the number of pictures, counted in decoding order, over which the complexity metrics are applicable when period_type is 3 or 8. When period_type is 8, this is a default number of pictures for each temporal layer, which can be overidden using temporal_map flags.

NumPicsInPeriod specifies the number of pictures in the specified period. When period_type is 0 or 4, then NumPicsInPeriod is 1. When period_type is 1, then NumPicsInPeriod is determined by counting the pictures in decoding order up to (but not including) the one containing the next I slice. When period_type is 2, then NumPicsInPeriod is determined from the frame rate. When period_type is 3, then NumPicsInPeriod is equal to num_pictures.

TotalNumMacroblocksInPeriod specifies the total number of macroblocks that are coded in the specified period. It is determined by the following computation:

$$\sum_{n=1}^{\mathrm{NumPicsInPeriod}} \mathrm{TotalNumMacroblocksPic}(n) \tag{7}$$

where TotalNumMacroblocksPic(n) is set to the value of the AVC variable PicSizeInMbs for the nth picture within the specified period, where 1 <= n <= NumPicsInPeriod.

temporal_map indicates which temporal layer has a different number of pictures from num_pictures in the specified period, when period_type is 8.

num_pictures_in_temporal_layer[t] indicates the number of pictures in the specified period for the tth temporal layer when period_type is 8. When not present, it is equal to num_pictures.

NumPicsInPeriodForTemporalLayer[t] specifies the number of pictures in the specified period for the tth temporal layer. When period_type is 5 then NumPicsInPeriodForTemporalLayer[t] is 1. When period_type is 6, then NumPicsInPeriodForTemporalLayer[t] is determined by counting the pictures associated to the tth temporal layer in decoding order up to (but not including) the one containing the next I slice. When period_type is 7, then NumPicsInPeriodForTemporalLayer[t] is determined from the frame rate associated to the tth temporal layer. When period_type is 8, then NumPicsInPeriodForTemporalLayer[t] is equal to num_pictures_in_temporal_layer[t].

portion_non_zero_8x8_blocks indicates the portion of 8x8 blocks with non-zero transform coefficients values in the specified period and is defined as follows:

$$\mathrm{portion\_non\_zero\_8x8\_blocks} = \mathrm{Floor}\left(\frac{\mathrm{NumNonZero8x8Blocks}}{\mathrm{TotalNumMacroblocksInPeriod} * 4} * 255\right) \tag{8}$$

where NumNonZero8x8Blocks is the number of 8x8 blocks with non-zero transform coefficients values in the specified period. NumNonZero8x8Blocks is derived from portion_non_zero_8x8_blocks and TotalNumMacroblocksInPeriod in the decoder.

portion_intra_predicted_macroblocks indicates the portion of intra-predicted macroblocks in the specified period and is defined as follows:

$$\text{portion\_intra\_predicted\_macroblocks} = \text{Floor}\left(\frac{\text{NumIntraPredictedMacroblocks}}{\text{TotalNumMacroblocksInPeriod}} * 255\right) \quad (9)$$

where NumIntraPredictedMacroblocks is the number of intra-predicted macroblocks in the specified period. NumIntraPredictedMacroblocks is derived from portion_intra_predicted_macroblocks and TotalNumMacroblocksInPeriod in the decoder.

portion_six_tap_filterings indicates the portion of 6-tap filterings (STFs) in the specified period and is defined as follows:

$$\text{portion\_six\_tap\_filterings} = \text{Floor}\left(\frac{\text{NumSixTapFilterings}}{\text{MaxNumSixTapFilteringsInPeriod}} * 255\right) \quad (10)$$

where MaxNumSixTapFilteringsInPeriod is the maximum number of STFs that can occur within the specified period and is derived from TotalNumMacroblocksInPeriod variable as

$$\text{MaxNumSixTapFilteringsInPeriod} = (1664 * \text{TotalNumMacroblocksInPeriod}) \quad (11)$$

and NumSixTapFilterings is the number of 6-tap filterings (STFs) within the specified period. Guidance for the counting of NumSixTapFilterings can be found in Annex B. NumSixTapFilterings is derived from portion_six_tap_filterings and MaxNumSixTapFilteringsInPeriod in the decoder.

portion_alpha_point_deblocking_instances indicates the portion of alpha-point deblocking instances (APDIs) in the specified period and is defined as follows:

$$\text{portion\_alpha\_point\_deblocking\_instances} = \text{Floor}\left(\frac{\text{NumAlphaPointDeblockingInstances}}{\text{MaxNumAlphaPointDeblockingInstancesInPeriod}} * 255\right) \quad (12)$$

MaxNumAlphaPointDeblockingInstancesInPeriod is the maximum number of APDIs that can occur within the specified period and is derived from TotalNumMacroblocksInPeriod and ChromaFormatMultiplier variables as

$$\text{MaxNumAlphaPointDeblockingInstancesInPeriod} = 128 * \text{ChromaFormatMultiplier} * \text{TotalNumMacroblocksInPeriod} \quad (13)$$

ChromaFormatMultiplier depends on the AVC variables separate_colour_plane_flag and chroma_format_idc as shown in Table 4.

Table 4 — ChromaFormatMultiplier

ChromaFormatMultiplier	separate_colour_plane_flag	chroma_format_idc	Comment
1	0	0	monochrome
1.5	0	1	4:2:0 sampling
2	0	2	4:2:2 sampling
3	0	3	4:4:4 sampling
3	1	any value	separate colour plane

NumAlphaPointDeblockingInstances is the number of APDIs in the specified period. Using the notation in ISO/IEC 14496-10, this is equivalent to the total number of filtering operations applied to produce filtered samples of the type p'_0 or q'_0, in the specified period. NumAlphaPointDeblockingInstances is derived from portion_alpha_point_deblocking_instances and MaxNumAlphaPointDeblockingInstancesInPeriod in the decoder.

num_slices_minus1 plus 1 indicates the number of slices per slice_group in the picture.

first_mb_in_slice[i][j] indicates the first macroblock number in the slice[i][j].

TotalNumMacroblocksInSlice[i][j] is the total number of macroblocks that are coded in the slice[i][j] and is determined by the following computation:

If num_slice_groups_minus1 is equal to 0:

```
if (j<num_slices_minus1[0])
  TotalNumMacroblocksInSlice[0][j]=first_mb_in_slice[0][j+1] - first_mb_in_slice[0][j]

else
  TotalNumMacroblocksInSlice[0][j]=PicSizeInMbs - first_mb_in_slice[0][j]          (14)
```

Otherwise (num_slice_groups_minus1 is not equal to 0), and after derivation of the macroblock to slice group map (MbToSliceGroupMap) as specified in ISO/IEC 14496-10:—, 8.2.2.8.

```
if (j<num_slices_minus1[i])
    k=0;
 for ( n=first_mb_in_slice[i][j]; n< first_mb_in_slice[i][j+1]; n++ )
            if ( MbToSliceGroupMap[first_mb_in_slice[i][j]] == MbToSliceGroupMap[n] )
              k++;
      TotalNumMacroblocksInSlice[i][j]=k;
else
 k=0;
 for ( n=first_mb_in_slice[i][j]; n< PicSizeInMbs; n++ )
            if ( MbToSliceGroupMap[first_mb_in_slice[i][j]] == MbToSliceGroupMap[n] )
              k++;
TotalNumMacroblocksInSlice[i][j]=k;          (15)
```

portion_non_zero_8x8_blocks[i][j] indicates the portion of 8x8 blocks with non-zero transform coefficients values in the slice[i][j] and is defined as follows:.

$$\text{portion\_non\_zero\_8x8\_blocks}[i][j] = \text{Floor}\left(\frac{\text{NumNonZero8x8Blocks}[i][j]}{\text{TotalNumMacroblocksInSlice}[i][j] * 4} * 255 \right) \qquad (16)$$

where NumNonZero8x8Blocks[i][j] is the number of 8x8 blocks with non-zero transform coefficients values in the slice[i][j]. NumNonZero8x8Blocks[i][j] is derived from portion_non_zero_8x8_blocks[i][j] and TotalNumMacroblocksInSlice[i][j] in the decoder.

portion_intra_predicted_macroblocks[i][j] indicates the portion of macroblocks using Intra prediction modes in the slice[i][j] and is defined as follows:

$$\text{portion\_intra\_coded\_macroblocks[i][j]} = \text{Floor}\left(\frac{\text{NumIntraPredictedMacroblocks[i][j]}}{\text{TotalNumMacroblocksInSlice[i][j]}} * 255\right) \tag{17}$$

where NumIntraPredictedMacroblocks[i][j] is the number of macroblocks using Intra prediction modes in the slice[i][j]. NumIntraPredictedMacroblocks[i][j] is derived from portion_intra_predicted_macroblocks[i][j] and TotalNumMacroblocksInSlice[i][j] in the decoder.

portion_six_tap_filterings[i][j] indicates the portion of 6-tap filterings (STFs) in the specified slice[i][j] and is defined as follows:

$$\text{portion\_six\_tap\_filterings[i][j]} = \text{Floor}\left(\frac{\text{NumSixTapFilterings[i][j]}}{\text{MaxNumSixTapFilteringsInSlice[i][j]}} * 255\right) \tag{18}$$

where MaxNumSixTapFilteringsInSlice[i][j] is the maximum number of STFs that can occur in the slice[i][j] and is derived from TotalNumMacroblocksInSlice[i][j] variable as

$$\text{MaxNumSixTapFilteringsInSlice[i][j]} = 1664 * \text{TotalNumMacroblocksInSlice[i][j]} \tag{19}$$

and NumSixTapFilterings[i][j] is the number of 6-tap filterings (STFs) within the slice[i][j]. Guidance for the counting of NumSixTapFilterings[i][j] can be found in Annex B. NumSixTapFilterings[i][j] is derived from portion_six_tap_filterings[i][j] and MaxNumSixTapFilteringsInSlice[i][j] in the decoder.

portion_alpha_point_deblocking_instances[i][j] indicates the portion of alpha-point deblocking instances (APDIs) in the specified slice[i][j] and is defined as follows:

$$\begin{gathered}\text{portion\_alpha\_point\_deblocking\_instances[i][j]} = \\ \text{Floor}\left(\frac{\text{NumAlphaPointDeblockingInstances[i][j]}}{\text{MaxNumAlphaPointDeblockingInstancesInSlice[i][j]}} * 255\right)\end{gathered} \tag{20}$$

where MaxNumAlphaPointDeblockingInstancesInSlice[i][j] is the maximum number of APDIs that can occur in the slice[i][j] and is derived from TotalNumMacroblocksInSlice[i][j] and ChromaFormatMultiplier variables as

$$\begin{gathered}\text{MaxNumAlphaPointDeblockingInstancesInSlice[i][j]} = \\ 128 * \text{ChromaFormatMultiplier} * \text{TotalNumMacroblocksInSlice[i][j]}\end{gathered} \tag{21}$$

and NumAlphaPointDeblockingInstances[i][j] is the number of alpha-point deblocking instances (APDIs) in slice[i][j]. NumAlphaPointDeblockingInstances[i][j] is derived from portion_alpha_point_deblocking_instances[i][j] and MaxNumAlphaPointDeblockingInstancesInSlice[i][j] in the decoder.

num_layers_minus1 plus 1 indicates the number of scalable layers in the associated picture or in the specified period.

pic_parameter_set_id[l] indicates the picture parameter set in use for the l^{th} scalable layer. The value of pic_parameter_set_id[l] shall be in the range of 0 to 255, inclusive (as specified in ISO/IEC 14496-10:—, G.7.4.3.4).

priority_id[l] indicates a priority identifier for the NAL unit in the l^{th} scalable layer. The value of priority_id[l] shall be in the range of 0 to 63, inclusive (as specified in ISO/IEC 14496-10:—, G.7.4.4.1).

dependency_id[l] indicates a dependency identifier for the NAL unit in the l^{th} scalable layer. The value of dependency_id[l] shall be in the range of 0 to 7, inclusive (as specified in ISO/IEC 14496-10:—, G.7.4.4.1).

quality_id[l] indicates a quality identifier for the NAL unit in the l^{th} scalable layer. The value of quality_id[l] shall be in the range of 0 to 15, inclusive (as specified in ISO/IEC 14496-10:—, G.7.4.4.1).

temporal_id[l] indicates a temporal identifier for the NAL unit in the lth scalable layer. The value of temporal_id[l] shall be in the range of 0 to 7, inclusive (as specified in ISO/IEC 14496-10:—, G.7.4.4.1).

portion_non_zero_8x8_blocks[l] indicates the portion of 8x8 blocks with non-zero transform coefficients values in the lth scalable layer and is defined as follows:.

$$\mathrm{portion\_non\_zero\_8x8\_blocks[l]} = \mathrm{Floor}\left(\frac{\mathrm{NumNonZero8x8Blocks[l]}}{\mathrm{TotalNumMacroblocksInLayerInPeriod[l]}*4}*255\right) \tag{22}$$

TotalNumMacroblocksInLayerInPeriod[l] is the total number of macroblocks in the lth scalable layer in the specified period and is derived from TotalNumMacroblocksInLayer[l] and NumPicsInPeriodForTemporalLayer[temporal[l]]as

$$\mathrm{TotalNumMacroblocksInLayerInPeriod[l]} = \mathrm{TotalNumMacroblocksInLayer[l]} * \mathrm{NumPicsInPeriodForTemporalLayers[temporal\_id[l]]} \tag{23}$$

where TotalNumMacroblocksInLayer[l] is the total number of macroblocks in the lth scalable layer and determined after derivation of the number of macroblock associated with the pic_parameter_set_id[l], as specified in ISO/IEC 14496-10:—, G.7.4.3.4.

NumNonZero8x8Blocks[l] is the number of 8x8 blocks with non-zero transform coefficients values in the lth scalable layer in the specified period. It is derived from portion_non_zero_8x8_blocks[l] and TotalNumMacroblocksInLayerInPeriod[l] in the decoder.

portion_intra_predicted_macroblocks[l] indicates the portion of macroblocks using Intra prediction modes in the lth scalable layer and is defined as follows:

$$\mathrm{portion\_intra\_predicted\_macroblocks[l]} = \mathrm{Floor}\left(\frac{\mathrm{NumIntraPredictedMacroblocks[l]}}{\mathrm{TotalNumMacroblocksInLayerInPeriod[l]}}*255\right) \tag{24}$$

NumIntraPredictedMacroblocks[l] is the number of macroblocks using Intra prediction modes in the lth scalable layer in the specified period. It is derived from portion_intra_predicted_macroblocks[l] and TotalNumMacroblocksInLayerInPeriod[l] in the decoder.

portion_six_tap_filterings[l] indicates the portion of 6-tap filterings (STFs) in the specified lth scalable layer in the specified period and is defined as follows:

$$\mathrm{portion\_six\_tap\_filterings[l]} = \mathrm{Floor}\left(\frac{\mathrm{NumSixTapFilterings[l]}}{\mathrm{MaxNumSixTapFilteringsInLayerInPeriod[l]}}*255\right) \tag{25}$$

MaxNumSixTapFilteringsInLayerInPeriod[l] is the maximum number of STFs that can occur in the lth scalable layer in the specified period and is derived from TotalNumMacroblocksInLayerInPeriod[l] variable as

$$\mathrm{MaxNumSixTapFilteringsInLayerInPeriod[l]} = 1664 * \mathrm{TotalNumMacroblocksInLayerInPeriod[l]} \tag{26}$$

NumSixTapFilterings[l] is the number of 6-tap filterings (STFs) within the lth scalable layer in the specified period. Guidance for the counting of NumSixTapFilterings[l] can be found in Annex B. It is derived from portion_six_tap_filterings[l] and MaxNumSixTapFilteringsInLayerInPeriod[l] in the decoder.

portion_alpha_point_deblocking_instances[l] indicates the portion of alpha-point deblocking instances (APDIs) in the specified l^th scalable layer in the specified period and is defined as follows:

$$\text{portion\_alpha\_point\_deblocking\_instances}[l] = \text{Floor}\left(\frac{\text{NumAlphaPointDeblockingInstances}[l]}{\text{MaxNumAlphaPointDeblockingInstancesInLayerInPeriod}[l]} * 255\right) \quad (27)$$

MaxNumAlphaPointDeblockingInstancesInLayerInPeriod[l] is the maximum number of APDIs that can occur in the l^th scalable layer in the specified period and is derived from TotalNumMacroblocksInLayerInPeriod[l] and ChromaFormatMultiplier variables as

$$\text{MaxNumAlphaPointDeblockingInstancesInLayer}[l] = 128 * \text{ChromaFormatMultiplier} * \text{TotalNumMacroblocksInLayersInPeriod}[l] \quad (28)$$

NumAlphaPointDeblockingInstances[l] is the number of alpha-point deblocking instances (APDIs) in the l^th scalable layer in the specified period. It is derived from portion_alpha_point_deblocking_instances[l] and MaxNumAlphaPointDeblockingInstancesInLayerInPeriod[l] in the decoder.

5.2.4.2 HEVC semantics

The semantics of various terms are defined below.

period_type specifies the type of upcoming period over which the four complexity metrics are applicable and is defined in Table 5.

Table 5 — Value and description of period_type

Value	Description
0x00	complexity metrics are applicable to a single picture
0x01	complexity metrics are applicable to all pictures in decoding order, up to (but not including) the picture containing the next I slice
0x02	complexity metrics are applicable over a specified time interval in seconds
0x03	complexity metrics are applicable over a specified number of pictures counted in decoding order
0x04	complexity metrics are applicable to a single picture with slice or tile granularity
0x05-0xFF	Reserved

num_seconds indicates the number of seconds over which the complexity metrics are applicable when period_type is 2.

num_pictures specifies the number of pictures, counted in decoding order, over which the complexity metrics are applicable when period_type is 3.

NumPicsInPeriod is the number of pictures in the specified period. When period_type is 0, then NumPicsInPeriod is 1. When period_type is 1, then NumPicsInPeriod is determined by counting the pictures in decoding order up to (but not including) the one containing the next I slice. When period_type is 2, then NumPicsInPeriod is determined from the frame rate. When period_type is 3, then NumPicsInPeriod is equal to num_pictures.

TotalNum4x4BlocksInPeriod is the total number of 4x4 blocks that are coded in the specified period.

It is determined by the following computation:

$$\sum_{n=1}^{\mathrm{NumPicsInPeriod}} \mathrm{TotalNum4x4BlocksPic}(\mathrm{n}) \tag{29}$$

where TotalNum4x4BlocksPic(n) is derived from HEVC variables as follows

$$\mathrm{PicSizeInCtbsY} * (1 << (\mathrm{CtbLog2SizeY}-2))^2 \tag{30}$$

for the nth picture within the specified period, where 1 <= n <= NumPicsInPeriod.

portion_non_zero_blocks_area indicates the portion of area covered by blocks with non-zero transform coefficients values, in the pictures of the specified period, using a 4x4 blocks granularity and is defined as follows:

$$\mathrm{portion\_non\_zero\_blocks\_area} = \mathrm{Floor}\left(\frac{\mathrm{NumNonZeroBlocks}}{\mathrm{TotalNum4x4BlocksInPeriod}} * 255\right) \tag{31}$$

where NumNonZeroBlocks is the number of blocks with non-zero transform coefficients values in the specified period using 4x4 granularity. At the encoder side, NumNonZeroBlocks is computed as follows:

$$\begin{gathered}\mathrm{NumNonZeroBlocks} = \mathrm{NumNonZero4x4Blocks} + 4 * \mathrm{NumNonZero8x8Blocks} + \\ 16 * \mathrm{NumNonZero16x16Blocks} + 64 * \mathrm{NumNonZero32x32Blocks}\end{gathered} \tag{32}$$

where NumNonZero4x4Blocks, NumNonZero8x8Blocks, NumNonZero16x16Blocks, NumNonZero32x32Blocks are the number of 4x4, 8x8, 16x16 and 32x32 blocks with non-zero transform coefficients values, respectively, in the specified period.

NumNonZeroBlocks is derived from portion_non_zero_blocks_area and TotalNum4x4BlocksInPeriod in the decoder.

portion_8x8_blocks_in_non_zero_area indicates the portion of 8x8 blocks area in the non-zero area in the specified period and is defined as follows:

$$\mathrm{portion\_8x8\_blocks\_in\_non\_zero\_area} = \mathrm{Floor}\left(\frac{4 * \mathrm{NumNonZero8x8Blocks}}{\mathrm{NumNonZeroBlocks}} * 255\right) \tag{33}$$

When not present, is equal to 0.

NumNonZero8x8Blocks is the number of 8x8 blocks with non-zero transform coefficients values in the specified period. It is derived from portion_8x8_blocks_in_non_zero_area and NumNonZeroBlocks in the decoder.

portion_16x16_blocks_in_non_zero_area indicates the portion of 16x16 blocks area in the non-zero area in the specified period and is defined as follows:

$$\mathrm{portion\_16x16\_blocks\_in\_non\_zero\_area} = \mathrm{Floor}\left(\frac{16 * \mathrm{NumNonZero16x16Blocks}}{\mathrm{NumNonZeroBlocks}} * 255\right) \tag{34}$$

When not present, is equal to 0.

NumNonZero16x16Blocks is the number of 16x16 blocks with non-zero transform coefficients values in the specified period. It is derived from portion_16x16_blocks_in_non_zero_area and NumNonZeroBlocks in the decoder.

portion_32x32_blocks_in_non_zero_area indicates the portion of 32x32 blocks area in the non- zero area in the specified period and is defined as follows:

$$\text{portion\_32x32\_blocks\_in\_non\_zero\_area} = \text{Floor}\left(\frac{64 * \text{NumNonZero32x32Blocks}}{\text{NumNonZeroBlocks}} * 255\right) \tag{35}$$

When not present, is equal to 0.

NumNonZero32x32Blocks is the number of 32x32 blocks with non-zero transform coefficients values in the specified period. It is derived from portion_32x32_blocks_in_non_zero_area and NumNonZeroBlocks in the decoder.

NumNonZero4x4Blocks is the number of 4x4 blocks with non-zero transform coefficients values in the specified period. NumNonZero4x4Blocks is derived from NumNonZeroBlocks, NumNonZero8x8Blocks, NumNonZero16x16Blocks and NumNonZero32x32Blocks as follows in the decoder:

$$\begin{gathered}\text{NumNonZero4x4Blocks} = \text{NumNonZeroBlocks} - 4 * \text{NumNonZero8x8Blocks} - \\ 16 * \text{NumNonZero16x16Blocks} - 64 * \text{NumNonZero32x32Blocks}\end{gathered} \tag{36}$$

portion_intra_predicted_blocks_area indicates the portion of area covered by intra predicted blocks in the pictures of the specified period using 4x4 granularity and is defined as follows:

$$\text{portion\_intra\_predicted\_blocks\_area} = \text{Floor}\left(\frac{4 * \text{NumIntraPredictedBlocks}}{\text{TotalNum4x4BlocksInPeriod}} * 255\right) \tag{37}$$

NumIntraPredictedBlocks is the number of intra predicted blocks in the specified period using 8x8 granularity. At the encoder side, it is computed as follows:

$$\begin{gathered}\text{NumIntraPredictedBlocks} = \text{NumIntraPredicted8x8Blocks} + \\ 4 * \text{NumIntraPredicted16x16Blocks} + 16 * \text{NumIntraPredicted32x32Blocks} + \\ 64 * \text{NumIntraPredicted64x64Blocks}\end{gathered} \tag{38}$$

where NumIntraPredicted8x8Blocks, NumIntraPredicted16x16Blocks, NumIntraPredicted32x32Blocks and NumIntraPredicted64x64Blocks are the number of intra predicted 8x8, 16x16, 32x32 and 64x64 blocks respectively, in the specified period.

NumIntraPredictedBlocks is derived from portion_intra_predicted_blocks_area and TotalNum4x4BlocksInPeriod in the decoder.

portion_planar_blocks_in_intra_area — indicates the portion of planar blocks area in the intra predicted area in the specified period and is defined as follows:

$$\text{portion\_planar\_blocks\_in\_intra\_area} = \text{Floor}\left(\frac{\text{NumPlanarPredictedBlocks}}{4 * \text{NumIntraPredictedBlocks}} * 255\right) \tag{39}$$

When not present, is equal to 0.

NumPlanarPredictedBlocks is the number of intra planar predicted blocks in the specified period using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{gathered}\text{NumPlanarPredictedBlocks} = \text{NumPlanarPredicted4x4Blocks} + \\ 4 * \text{NumPlanarPredicted8x8Blocks} + 16 * \text{NumPlanarPredicted16x16Blocks} + \\ 64 * \text{NumPlanarPredicted32x32Blocks} + 256 * \text{NumPlanarPredicted64x64Blocks}\end{gathered} \tag{40}$$

where NumPlanarPredicted4x4Blocks, NumPlanarPredicted8x8Blocks, NumIntraPredicted16x16Blocks, NumIntraPredicted32x32Blocks and NumIntraPredicted64x64Blocks are the number of intra planar predicted 4x4, 8x8, 16x16, 32x32 and 64x64 blocks respectively, in the specified period.

NumPlanarPredictedBlocks is derived from portion_planar_blocks_in_intra_area and NumIntraPredictedBlocks in the decoder.

portion_dc_blocks_in_intra_area indicates the portion of DC blocks area in the intra predicted area in the specified period and is defined as follows:

$$\text{portion\_dc\_blocks\_in\_intra\_area} = \text{Floor}\left(\frac{\text{NumDCPredictedBlocks}}{4 * \text{NumIntraPredictedBlocks}} * 255\right) \quad (41)$$

When not present, is equal to 0.

NumDCPredictedBlocks is the number of intra DC predicted blocks in the specified period using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{aligned}&\text{NumDCPredictedBlocks} = \text{NumDCPredicted4x4Blocks} +\\&4 * \text{NumDCPredicted8x8Blocks} + 16 * \text{Num\_DCPredicted16x16Blocks} +\\&64 * \text{NumDCPredicted32x32Blocks} + 256 * \text{NumDCPredicted64x64Blocks}\end{aligned} \quad (42)$$

where NumDCPredicted4x4Blocks, NumDCPredicted8x8Blocks, NumDCPredicted16x16Blocks, NumDCPredicted32x32Blocks and NumDCPredicted64x64Blocks are the number of intra DC predicted NumDCPredictedBlocks is derived from portion_dc_blocks_in_intra_area and NumIntraPredictedBlocks in the decoder. 4x4, 8x8, 16x16, 32x32 and 64x64 blocks respectively, in the specified period.

portion_angular_hv_blocks_in_intra_area indicates the portion of angular horizontal or vertical blocks area in the intra predicted area in the specified period and is defined as follows:

$$\text{portion\_angular\_hv\_blocks\_in\_intra\_area} = \text{Floor}\left(\frac{\text{NumAngularHVPredictedBlocks}}{4 * \text{NumIntraPredictedBlocks}} * 255\right) \quad (43)$$

When not present, is equal to 0.

NumAngularHVPredictedBlocks is the number of intra angular horizontally or vertically predicted blocks in the specified period using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{aligned}&\text{NumAngularHVPredictedBlocks} = \text{NumAngularHVPredicted4x4Blocks} +\\&4 * \text{NumAngularHVPredicted8x8Blocks} + 16 * \text{NumAngularHVPredicted16x16Blocks} +\\&64 * \text{NumAngularHVPredicted32x32Blocks} + 256 * \text{NumAngularHVPredicted64x64Blocks}\end{aligned} \quad (44)$$

where NumAngularHVPredicted4x4Blocks, NumAngularHVPredicted8x8Blocks, NumAngularHVPredicted16x16Blocks, NumAngularHVPredicted32x32Blocks and NumAngularHVCPredicted64x64Blocks are the number of intra angular horizontally or vertically predicted 4x4, 8x8, 16x16, 32x32 and 64x64 blocks respectively, in the specified period.

NumAngularHVPredictedBlocks is derived from portion_angular_hv_blocks_in_intra_area and NumIntraPredictedBlocks in the decoder.

portion_blocks_a_c_d_n_filterings indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position a, c, d or n, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the specified period and is defined as follows:

$$\text{portion\_blocks\_a\_c\_d\_n\_filterings} = \text{Floor}\left(\frac{\text{NumBlocksACDNFilterings}}{\text{TotalNum4x4Blocks}} * 255\right) \quad (45)$$

When not present, is equal to 0.

NumBlocksACDNFilterings is the number of prediction blocks whose luma samples positions are located in sub-sample position a, c, d or n in the specified period. It is derived from portion_blocks_a_c_d_n_filterings and TotalNum4x4Blocks in the decoder.

portion_blocks_h_b_filterings indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position h or b, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the specified period and is defined as follows:

$$\text{portion\_blocks\_h\_b\_filterings} = \text{Floor}\left(\frac{\text{NumBlocksHBFilterings}}{\text{TotalNum4x4Blocks}} * 255\right) \quad (46)$$

When not present, is equal to 0.

NumBlocksHBFilterings is the number of prediction blocks whose luma samples positions are located in sub-sample position h or b in the specified period.

It is derived from portion_blocks_h_b_filterings and TotalNum4x4Blocks in the decoder.

portion_blocks_f_i_k_q_filterings indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position f, I, k or q, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the specified period and is defined as follows:

$$\text{portion\_blocks\_f\_i\_k\_q\_filterings} = \text{Floor}\left(\frac{\text{NumBlocksFIKQFilterings}}{\text{TotalNum4x4Blocks}} * 255\right) \quad (47)$$

When not present, is equal to 0.

NumBlocksFIKQFilterings is the number of prediction blocks whose luma samples positions are located in sub-sample position f, I, k or q in the specified period.

It is derived from portion_blocks_f_i_k_q_filterings and TotalNum4x4Blocks in the decoder.

portion_blocks_j_filterings indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position j, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the specified period and is defined as follows:

$$\text{portion\_blocks\_j\_filterings} = \text{Floor}\left(\frac{\text{NumBlocksJFilterings}}{\text{TotalNum4x4Blocks}} * 255\right) \quad (48)$$

When not present, is equal to 0.

NumBlocksJFilterings is the number of prediction blocks whose luma samples positions are located in sub-sample position j in the specified period.

It is derived from portion_blocks_j_filterings and TotalNum4x4Blocks in the decoder.

portion_blocks_e_g_p_r_filterings indicates the portion of prediction blocks whose luma blocks positions are located in sub-sample position e, g, p or r, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the specified period and is defined as follows:

$$\text{portion\_blocks\_e\_g\_p\_r\_filterings} = \text{Floor}\left(\frac{\text{NumBlocksEGPRFilterings}}{\text{TotalNum4x4Blocks}} * 255\right) \quad (49)$$

When not present, is equal to 0.

NumBlocksEGPRFilterings is the number of prediction blocks whose luma samples positions are located in sub-sample position e, g, p or r in the specified period.

It is derived from portion_blocks_e_g_p_r_filterings and TotalNum4x4Blocks in the decoder.

portion_deblocking_instances indicates the portion of deblocking filtering instances in the specified period and is defined as follows:

$$\text{portion\_deblocking\_instances} = \text{Floor}\left(\frac{\text{NumDeblockingInstances}}{4 * \text{ChromaFormatMultiplier} * \text{TotalNum4x4Blocks}} * 255\right) \quad (50)$$

ChromaFormatMultiplier depends on the HEVC variables separate_colour_plane_flag and chroma_format_idc as shown in Table 6.

Table 6 — ChromaFormatMultiplier

ChromaFormatMultiplier	separate_colour_plane_flag	chroma_format_idc	Comment
1	0	0	monochrome
1.5	0	1	4:2:0 sampling
2	0	2	4:2:2 sampling
3	0	3	4:4:4 sampling
3	1	3	separate colour plane

NumDeblockingInstances is the number of deblocking filtering instances in the specified period. It is derived from portion_deblocking_instances, TotalNum4x4Blocks and ChromaFormatMultiplier in the decoder.

max_num_slices_tiles_minus1 specifies the maximum number between the number of slices and the number of tiles in the associated picture.

first_ctb_in_slice_or_tile[t] specifies the first Coding Tree Block (CTB) number in slice[t] or tile[t] in raster scan order.

TotalNum4x4BlocksInSliceOrTile[t] is the total number of 4x4 blocks in the slice[t] or tile[t] and is determined by the following computation after derivation of the Coding tree block raster and tile scanning conversion process (CtbAddrRsToTs) as specified in ISO/IEC 23008-2:—, 6.5.1:

$$\begin{aligned}&\text{TotalNum4x4BlocksInSliceOrTile[t]} = (\text{CtbAddrRsToTs[first\_ctb\_in\_slice\_or\_tile[t+1]]} - \\ &\text{CtbAddrRsToTs[first\_ctb\_in\_slice\_or\_tile[t]]}) * (1 << (\text{CtbLog2SizeY-2}))^2\end{aligned} \quad (51)$$

except for the last slice or tile of the picture (t= num_max_slices_tiles_minus1), where it is determined by the following computation:

$$\begin{aligned}&\text{TotalNum4x4BlocksInSliceOrTile[t]} = (\text{CtbAddrRsToTs[PicSizeInCtbsY]} - \\ &\text{CtbAddrRsToTs[first\_ctb\_in\_slice\_or\_tile[t]]}) * (1 << (\text{CtbLog2SizeY-2}))^2\end{aligned} \quad (52)$$

portion_non_zero_blocks_area[t] indicates the portion of area covered by blocks with non-zero transform coefficients values in the slice[t] or tile[t] using a 4x4 blocks granularity and is defined as follows:

$$\text{portion\_non\_zero\_blocks\_area[t]} = \text{Floor}\left(\frac{\text{NumNonZeroBlocksInSliceOrTile[t]}}{\text{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \quad (53)$$

NumNonZeroBlocksInSliceOrTile[t] is the number of blocks with non-zero transform coefficients values in the slice[t] or tile[t] using 4x4 granularity. At the encoder side, it is computed as follows:

$$\mathrm{NumNonZeroBlocksInSliceOrTile[t]} = \mathrm{NumNonZero4x4BlocksInSliceOrTile[t]} + \\ 4 * \mathrm{NumNonZero8x8BlocksInSliceOrTile[t]} + 16 * \mathrm{NumNonZero16x16BlocksInSliceOrTile[t]} + \\ 64 \times * \mathrm{NumNonZero32x32BlocksInSliceOrTile[t]} \quad (54)$$

where NumNonZero4x4BlocksInSliceOrTile[t], NumNonZero8x8BlocksInSliceOrTile[t], NumNonZero16x16BlocksInSliceOrTile[t], NumNonZero32x32BlocksInSliceOrTiles[t] are the number of non-zero 4x4, 8x8, 16x16, 32x32 blocks in the slice[t] or tile[t] respectively.

NumNonZeroBlocksInSliceOrTile[t] is derived from portion_non_zero_blocks_area[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_8x8_blocks_in_non_zero_area[t] indicates the portion of 8x8 blocks area in the non-zero area in the slice[t] or tile[t] and is defined as follows:

$$\mathrm{portion\_8x8\_blocks\_in\_non\_zero\_area[t]} = \\ \mathrm{Floor}\left(\frac{4 * \mathrm{NumNonZero8x8BlocksInSliceOrTile[t]}}{\mathrm{NumNonZeroBlocksInSliceOrTile[t]}} * 255\right) \quad (55)$$

When not present, is equal to 0.

NumNonZero8x8BlocksInSliceOrTile[t] is the number of 8x8 blocks with non-zero transform coefficients values in the slice[t] or tile[t]. It is derived from portion_8x8_blocks_in_non_zero_area[t] and NumNonZeroBlocksInSliceOrTile[t] in the decoder.

portion_16x16_blocks_in_non_zero_area[t] indicates the portion of 16x16 blocks area in the non-zero area in the slice[t] or tile[t] and is defined as follows:

$$\mathrm{portion\_16x16\_blocks\_in\_non\_zero\_area[t]} = \\ \mathrm{Floor}\left(\frac{16 * \mathrm{NumNonZero16x16BlocksInSliceOrTilez[t]}}{\mathrm{NumNonZeroBlocksInSliceOrTile[t]}} * 255\right) \quad (56)$$

When not present, is equal to 0.

NumNonZero16x16BlocksInSliceOrTile[t] is the number of 16x16 blocks with non-zero transform coefficients values in the slice[t] or tile[t]. It is derived from portion_16x16_blocks_in_non_zero_area[t] and NumNonZeroBlocksInSliceOrTile[t] in the decoder.

portion_32x32_blocks_in_non_zero_area[t] indicates the portion of 32x32 blocks area in the non-zero area in the slice[t] or tile[t] and is defined as follows:

$$\mathrm{portion\_32x32\_blocks\_in\_non\_zero\_area[t]} = \\ \mathrm{Floor}\left(\frac{64 * \mathrm{NumNonZero32x32BlocksInSliceOrTile[t]}}{\mathrm{NumNonZeroBlocksInSliceOrTile[t]}} * 255\right) \quad (57)$$

When not present, is equal to 0.

NumNonZero32x32BlocksInSliceOrTile[t] is the number of 32x32 blocks with non-zero transform coefficients values in the slice[t] or tile[t]. It is derived from portion_32x32_blocks_in_non_zero_area[t] and NumNonZeroBlocksInSliceOrTile[t] in the decoder.

NumNonZero4x4BlocksInSliceOrTile[t] is the number of 4x4 blocks with non-zero transform coefficients values in the slice[t] or tile[t]. It is derived from NumNonZeroBlocksInSliceOrTile[t], NumNonZero8x8BlocksInSliceOrTile[t], NumNonZero16x16BlocksInSliceOrTile[t] and NumNonZero32x32BlocksInSliceOrTile[t] as follows in the decoder:

$$\begin{aligned}&\text{NumNonZero4x4BlocksInSliceOrTile[t]} = \text{NumNonZeroBlocksInSliceOrTile[t]} -\\ &4 * \text{NumNonZero8x8BlocksInSliceOrTile[t]} - 16 * \text{NumNonZero16x16BlocksInSliceOrTile[t]} -\\ &64 * \text{NumNonZero32x32BlocksInSliceOrTile[t]}\end{aligned} \tag{58}$$

portion_intra_predicted_blocks_area[t] indicates the portion of area covered by intra predicted blocks in the slice[t] or tile[t] using 8x8 granularity and is defined as follows:

$$\begin{aligned}&\text{portion\_intra\_predicted\_blocks\_area[t]} =\\ &\text{Floor}\left(\frac{4 * \text{NumIntraPredictedBlocksInSliceOrTile[t]}}{\text{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right)\end{aligned} \tag{59}$$

NumIntraPredictedBlocksInSliceOrTile[t] is the number of intra predicted blocks using 8x8 granularity in the slice[t] or tile[t]. At the encoder side, it is computed as follows:

$$\begin{aligned}&\text{NumIntraPredictedBlocksInSliceOrTile[t]} = \text{NumIntraPredicted8x8BlocksInSliceOrTile[t]} +\\ &4 * \text{NumIntraPredicted16x16BlocksInSliceOrTile[t]} +\\ &16 * \text{NumIntraPredicted32x32BlocksInSliceOrTile[t]} +\\ &64 * \text{NumIntraPredicted64x64BlocksInSliceOrTile[t]}\end{aligned} \tag{60}$$

where NumIntraPredicted8x8BlocksInSliceOrTile[t], NumIntraPredicted16x16BlocksInSliceOrTile[t], NumIntraPredicted32x32BlocksInSliceOrTiles[t] and NumIntraPredicted64x64BlocksInSliceOrTile[t] are the number of intra predicted 8x8, 16x16, 32x32 and 64x64 blocks in the slice[t] or tile[t] respectively.

NumIntraPredictedBlocksInSliceOrTile[t] is derived from portion_intra_predicted_blocks_area[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_planar_blocks_in_intra_area[t] indicates the portion of planar blocks in the intra predicted area the slice[t] or tile[t] and is defined as follows:

$$\begin{aligned}&\text{portion\_planar\_blocks\_in\_intra\_area[t]} =\\ &\text{Floor}\left(\frac{\text{NumPlanarBlocksInSliceOrTile[t]}}{4 * \text{NumIntraPredictedBlocksInSliceOrTile[t]}} * 255\right)\end{aligned} \tag{61}$$

When not present, is equal to 0.

NumPlanarBlocksInSliceOrTile[t] is the number of intra planar predicted blocks in the slice[t] or tile[t] using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{aligned}&\text{NumPlanarBlocksInSliceOrTile[t]} = \text{NumPlanar4x4Blocks[t]} +\\ &4 * \text{NumPlanar8x8Blocks[t]} + 16 * \text{NumPlanar16x16Blocks[t]} +\\ &64 * \text{NumPlanar32x32Blocks[t]} + 256 * \text{NumPlanar64x64Blocks[t]}\end{aligned} \tag{62}$$

where NumPlanar4x4Blocks[t], NumPlanar8x8Blocks[t], NumPlanar16x16Blocks[t], NumPlanar32x32Blocks[t] and NumPlanar64x64Blocks[t] are the number of intra planar predicted 4x4, 8x8, 16x16, 32x32 and 64x64 blocks in the slice[t] or tile[t] respectively.

NumPlanarBlocksInSliceOrTile[t] is derived from portion_planar_blocks_in_intra_area[t] and NumIntraPredictedBlocksInSliceOrTile[t] in the decoder.

portion_dc_blocks_in_intra_area[t] indicates the portion of DC blocks in the intra predicted area in the slice[t] or tile[t] and is defined as follows:

$$\text{portion\_dc\_blocks\_in\_intra\_area}[t] = \text{Floor}\left(\frac{\text{NumDCBlocksInSliceOrTile}[t]}{4 * \text{NumIntraPredictedBlocksInSliceOrTile}[t]} * 255\right) \tag{63}$$

When not present, is equal to 0.

NumDCBlocksInSliceOrTile [t] is the number of intra DC predicted blocks in the slice[t] or tile[t] using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{gathered}\text{NumDCBlocksInSliceOrTile}[t] = \text{NumDC4x4Blocks}[t] + \\ 4 * \text{NumDC8x8Blocks}[t] + 16 * \text{NumDC16x16Blocks}[t] + \\ 64 * \text{NumDC32x32Blocks}[t] + 256 * \text{NumDC64x64Blocks}[t]\end{gathered} \tag{64}$$

where NumDC4x4Blocks[t], NumDC8x8Blocks[t], NumDC16x16Blocks[t], NumDC32x32Blocks[t] and NumDC64x64Blocks[t] are the number of intra DC predicted 4x4, 8x8, 16x16, 32x32 and 64x64 blocks in the slice[t] or tile[t] respectively.

NumDCBlocksInSliceOrTile[t] is derived from portion_dc_blocks_in_intra_area[t] and NumIntraPredictedBlocksInSliceOrTile[t] in the decoder.

portion_angular_hv_blocks_in_intra_area[t] indicates the portion of angular horizontal or vertical blocks in the intra predicted area in the slice[t] or tile[t] and is defined as follows:

$$\text{portion\_angular\_hv\_blocks\_in\_intra\_area}[t] = \text{Floor}\left(\frac{\text{NumAngularHVBlocksInSliceOrTile}[t]}{4 * \text{NumIntraPredictedBlocksInSliceOrTile}[t]} * 255\right) \tag{65}$$

When not present, is equal to 0.

NumAngularHVBlocksInSliceOrTile[t] is the number of intra angular horizontally or vertically predicted blocks in the slice[t] or tile[t] using 4x4 granularity. At the encoder side, it is computed as follows:

$$\begin{gathered}\text{NumAngularHVInSliceOrTile}[t] = \text{NumAngularHV4x4Blocks}[t] + \\ 4 * \text{NumAngularHV8x8Blocks}[t] + 16 * \text{NumAngularHV16x16Blocks}[t] + \\ 64 * \text{NumAngularHV32x32Blocks}[t] + 256 * \text{NumAngularHV64x64Blocks}[t]\end{gathered} \tag{66}$$

where NumAngularHV4x4Blocks[t], NumAngularHV8x8Blocks[t], NumAngularHV16x16Blocks[t], NumAngularHV32x32Blocks[t] and NumAngularHV64x64Blocks[t] are the number of intra angular horizontally or vertically predicted 4x4, 8x8, 16x16, 32x32 and 64x64 blocks in the slice[t] or tile[t] respectively.

NumAngularHVBlocksInSliceOrTile[t] is derived from portion_angular_hv_blocks_in_intra_area[t] and NumIntraPredictedBlocksInSliceOrTile[t] in the decoder.

portion_blocks_a_c_d_n_filterings[t] indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position a, c, d or n, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the slice[t] or tile[t]. When not present, is equal to 0.

$$\text{portion\_blocks\_a\_c\_d\_n\_filterings}[t] = \text{Floor}\left(\frac{\text{NumBlocksACDNFilterings}[t]}{\text{TotalNum4x4BlocksInSliceOrTile}[t]} * 255\right) \tag{67}$$

NumBlocksACDNFilterings[t] is the number of prediction blocks whose luma samples positions are located in sub-sample position a, c, d or n in the slice[t] or tile[t]. It is derived from portion_blocks_a_c_d_n_filterings[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_blocks_h_b_filterings[t] indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position h or b, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the slice[t] or tile[t]. When not present, is equal to 0.

$$\mathrm{portion\_blocks\_h\_b\_filterings[t]} = \mathrm{Floor}\left(\frac{\mathrm{NumBlocksHBFilterings[t]}}{\mathrm{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \tag{68}$$

NumBlocksHBFilterings[t] is the number of prediction blocks whose luma samples positions are located in sub-sample position h or b in the slice[t] or tile[t]. It is derived from portion_blocks_h_b_filterings[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_blocks_f_i_k_q_filterings[t] indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position f, i, k or q, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the slice[t] or tile[t]. When not present, is equal to 0.

$$\mathrm{portion\_blocks\_f\_i\_k\_q\_filterings[t]} = \mathrm{Floor}\left(\frac{\mathrm{NumBlocksFIKQFilterings[t]}}{\mathrm{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \tag{69}$$

NumBlocksFIKQFilterings[t] is the number of prediction blocks whose luma samples positions are located in sub-sample position f, i, k or q in the slice[t] or tile[t]. It is derived from portion_blocks_f_i_k_q_filterings[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_blocks_j_filterings[t] indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position j, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the slice[t] or tile[t]. When not present, is equal to 0.

$$\mathrm{portion\_blocks\_j\_filterings[t]} = \mathrm{Floor}\left(\frac{\mathrm{NumBlocksJFilterings[t]}}{\mathrm{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \tag{70}$$

NumBlocksJFilterings[t] is the number of prediction blocks whose luma samples positions are located in sub-sample position j in the slice[t] or tile[t]. It is derived from portion_blocks_j_filterings[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_blocks_e_g_p_r_filterings[t] indicates the portion of prediction blocks whose luma samples positions are located in sub-sample position e, g, p or r, as defined in ISO/IEC 23008-2 and illustrated in Annex B, in the slice[t] or tile[t]. When not present, is equal to 0.

$$\mathrm{portion\_blocks\_e\_g\_p\_r\_filterings[t]} = \mathrm{Floor}\left(\frac{\mathrm{NumBlocksEGPRFilterings[t]}}{\mathrm{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \tag{71}$$

NumBlocksEGPRFilterings[t] is the number of prediction blocks whose luma samples positions are located in sub-sample position e, g, p or r in the slice[t] or tile[t]. It is derived from portion_blocks_e_g_p_r_filterings[t] and TotalNum4x4BlocksInSliceOrTile[t] in the decoder.

portion_deblocking_instances[t] indicates the portion of deblocking filtering instances in the slice[t] or tile[t].

$$\mathrm{portion\_deblocking\_instances[t]} = \mathrm{Floor}\left(\frac{\mathrm{NumDeblockingInstances[t]}}{4 * \mathrm{ChromaFormatMultiplier} * \mathrm{TotalNum4x4BlocksInSliceOrTile[t]}} * 255\right) \tag{72}$$

NumDeblockingInstances[t] is the number of deblocking filtering instances in the slice[t] or tile[t]. It is derived from portion_deblocking_instances[t], TotalNum4x4BlocksInSliceOrTile[t] and ChromaFormatMultiplier in the decoder.

5.3 Interactive signalling for remote decoder-power reduction

5.3.1 General

For point-to-point video conferencing, each device contains a transmitter and a receiver. A local device sends metadata that instructs the remote device to modify the decoding complexity of the bitstream and thus reduce local decoder-power consumption.

5.3.2 Syntax

The syntax is as given in Table 7.

Table 7 — Syntax of interactive signalling for remote decoder-power reduction

	Size (bits)	Descriptor
dec_ops_reduction_req	8	signed integer

5.3.3 Signalling

The transmitter in each device sends a dec_ops_reduction_req (DOR-Req) message to the attention of the remote encoder. This message requests the remote encoder to adjust its encoding parameters so that ideally, when the local decoder decodes the bitstream, the power saving of the local decoder matches the power saving implied by the DOR-Req message.

5.3.4 Semantics

dec_ops_reduction_req indicates the requested percentage reduction of local decoding operations relative to the local decoding operations since the last dec_ops_reduction_req was sent to the transmitter, or since the start of the video session, if no earlier dec_ops_reduction_req was sent. The percentage is expressed as a signed integer. A negative percentage means an increase of decoding operations. dec_ops_reduction_req is an integer in the interval [–100, 100].

6 Display power reduction using display adaptation

6.1 General

With respect to the functional architecture, Display Adaptation (DA) provides green metadata comprised of RGB-component statistics and quality indicators. The statistics are used to set display controls in the presentation subsystem so that desired quality levels and corresponding display power reductions are attained.

6.2 Syntax

6.2.1 Systems without a signalling mechanism from the receiver to the transmitter

The message format given in Table 8 is used to send metadata from the transmitter to the receiver.

Table 8 — Message format for sending metadata from the transmiter to the receiver

	Size (bits)	Descriptor
num_constant_backlight_voltage_time_intervals	2	unsigned integer
num_max_variations	2	unsigned integer
num_quality_levels	4	unsigned integer
for (j = 0; j < num_max_variations; j++) {		

Table 8 *(continued)*

max_variation[j]	8	unsigned integer
}		
for (k = 0; k < num_constant_backlight_voltage_time_intervals;k++) {		
constant_backlight_voltage_time_interval[k]	16	unsigned integer
for (j = 0; j < num_max_variations; j++) {		
lower_bound[k][j]	8	unsigned integer
if (**lower_bound[k][j]** > 0) {		
upper_bound[k][j]	8	unsigned integer
}		
rgb_component_for_infinite_psnr[k][j]	8	unsigned integer
for (i = 1; i < = num_quality_levels; i++) {		
max_rgb_component[k][j][i]	8	unsigned integer
scaled_psnr_rgb[k][j][i]	8	unsigned integer
}		
}		
}		

6.2.2 Systems with a signalling mechanism from the receiver to the transmitter

The receiver first uses the message format given in Table 9 to signal information to the transmitter.

Table 9 — Message format for signalling information to the transmitter

	Size (bits)	Descriptor
constant_backlight_voltage_time_interval	16	unsigned integer
max_variation	8	unsigned integer

The transmitter then uses the message format shown in Table 10 to then signal metadata to the receiver.

Table 10 — Message format for signalling metadata to the receiver

	Size (bits)	Descriptor
num_quality_levels	4	unsigned integer
lower_bound	8	unsigned integer
if (lower_bound > 0)		
upper_bound	8	unsigned integer
rgb_component_for_infinite_psnr	8	unsigned integer
for (i = 1; i < = num_quality_levels; i++) {		
max_rgb_component[i]	8	unsigned integer
scaled_psnr_rgb[i]	8	unsigned integer
}		

6.3 Signalling

6.3.1 General

Display Adaptation Metadata can be carried in a transport stream as specified in ISO/IEC 13818-1 or it can be carried in metadata tracks within the ISO base media file format (ISO/IEC 14496-12). When carried in a transport stream, Display Adaptation Metadata shall be signalled in accordance with ISO/IEC 13818-1. When carried in metadata tracks within ISO base media file format, Display Adaptation Metadata shall be signalled in accordance with ISO/IEC 23001-10.

6.3.2 Systems without a signalling mechanism from the receiver to the transmitter

Using the format in 6.2.1, the transmitter sends a message to the receiver. The DA metadata is applicable to the presentation subsystem until the next message containing DA metadata arrives.

6.3.3 Systems with a signalling mechanism from the receiver to the transmitter

Using the first message format described in 6.2.2, the receiver first signals constant_backlight_voltage_time_interval and max_variation to the transmitter. The transmitter then uses the second message format in 6.2.2 to send a message to the receiver. The DA metadata is applicable to the presentation subsystem until the next message containing DA metadata arrives.

6.4 Semantics

num_constant_backlight_voltage_time_intervals indicates the number of constant backlight/voltage time intervals for which metadata is provided in the bitstream.

num_max_variations indicates the number of maximum variations for which metadata is provided in the bitstream.

num_quality_levels indicates the number of quality levels that are enabled by the metadata, excluding the NQLOP.

max_variation[j] indicates the maximal change between backlight values of two successive frames relative to the backlight value of the earlier frame. The backlight value for a frame is the value of BacklightScalingFactor[k][j][i] for that frame. BacklightScalingFactor[k][j][i] is derived from max_rgb_component[k][j][i] and PeakSignal, as max_rgb_component[k][j][i]/PeakSignal, for the kth constant_backlight_voltage_time_interval, jth max_variation and ith quality level. PeakSignal variable is the peak signal.

max_variation is in the range [0,001, 0,1] and is normalized to one byte by rounding after multiplying by 2 048. This is the jth maximal backlight change for which metadata is provided in the bitstream, where 0 <= j < num_max_variations.

constant_backlight_voltage_time_interval[k] indicates the minimum time interval, in milliseconds, that shall elapse before the backlight can be updated after the last backlight update. This is the kth minimum time interval for which metadata is provided in the bitstream, where 0 <= k < num_constant_backlight_voltage_time_intervals.

lower_bound[k][j] indicates if lower_bound[k][j] is greater than zero, then metadata for contrast enhancement is available at the lowest quality level, for the kth constant_backlight_voltage_time_interval and jth max_variation. If lower_bound[k][j] = 0, then contrast-enhancement metadata is unavailable.

upper_bound[k][j] indicates for the kth constant_backlight_voltage_time_interval and jth max_variation, if lower_bound[k][j] is greater than zero, then contrast enhancement is performed as follows: All RGB components of reconstructed frames that are less than or equal to lower_bound[k][j] are set to zero and all RGB components that are greater than or equal to upper_bound[k][j] are saturated to PeakSignal. The RGB components in the range (lower_bound[k][j], upper_bound[k][j]) are mapped linearly onto the range (0, PeakSignal).

rgb_component_for_infinite_psnr[k][j] indicates for the kth constant_backlight_voltage_time_interval and jth max_variation, the largest RGB component (as defined in 3.1) in the reconstructed frames. Therefore, ScaledFrames[k][j][0] are identical to the reconstructed frames. The rgb_component_for_infinite_psnr[k][j] defines a no-quality-loss operating point (NQLOP) and consequently ScaledFrames[k][j][0] have a PSNR of infinity relative to the reconstructed frames.

max_rgb_component[k][j][i] indicates for the kth constant_backlight_voltage_time_interval, jth max_variation and ith quality level, the maximum RGB component (as defined in 3.1) that is retained in the frames, where 1 <= i <= num_quality_levels.

NOTE 1 max_rgb_component[k][j][0] = rgb_component_for_infinite_psnr[k][j].

scaled_psnr_rgb[k][j][i] indicates the PSNR of ScaledFrames[k][j][i] relative to the reconstructed frames. ScaledFrames[k][j][i] are for the kth constant_backlight_voltage_time_interval, jth max_variation and ith quality level, the frames obtained from the reconstructed frames by saturating to max_rgb_component[k][j][i] all RGB components that are greater than max_rgb_component[k][j][i], where 0 <= i <= num_quality_levels.

The PSNR is defined as follows:

$$\text{scaled\_psnr\_rgb[k][j][i]}=$$

$$\text{Clip}\left(\text{Round}\left(10\log_{10}\left(\frac{\text{PeakSignal}^2*\text{width}*\text{height}*N_{\text{colour}}*N_{\text{frames}}}{\sum_{n=1}^{N_{\text{frames}}}\sum_{c=1}^{N_{\text{colour}}}\sum_{l=\text{X}_\text{s}+1}^{\text{PeakSignal}}N_{c,n}(l)*\left(l-\text{X}_\text{s}\right)^2}\right)\right)\right) \quad (73)$$

for 0 < i <= num_quality_levels,

where

width is the width of a video frame;

height is the height of a video frame;

N_{colour} is the number of colour channels. For RGB colourspace, $N_{\text{colour}} = 3$;

N_{frames} is the number of frames in the reconstructed frames;

$N_{\text{c,n}}(l)$ is the number of RGB components that are set to l in the nth frame of colour-channel c in reconstructed frames;

X_s is max_rgb_component[k][j][i].

NOTE 2 scaled_psnr_rgb[k][j][0] is associated with the NQLOP. It is not transmitted, but understood to be mathematically infinite.

7 Energy-efficient media selection

7.1 General

The green metadata specified in this clause can enable a client in an adaptive streaming session, such as DASH, to determine decoder and display power-saving characteristics of available video Representations and to select the Representation with the optimal quality for a given power-saving.

Two types of green metadata are defined as follows:

- decoder-power indication metadata gives the potential decoder power saving of each available Representation of a video Segment;
- display-power indication metadata gives the maximum potential display power saving of a video Segment for a specified number of quality levels. This metadata is computed without any constraint on the maximal backlight change between two successive frames and with no practical restriction on the minimum time interval between backlight updates. Therefore, using the semantics of 6.4, the metadata is produced with the assumptions that max_variation is mathematically infinite and

that constant_backlight_voltage_time_interval is less than or equal to the interval between two successive frames.

7.2 Syntax

The decoder-power indication metadata is a pair of decoder operations reduction ratios, as shown in Table 11.

Table 11 — Pair of decoder operations reduction ratios forming the decoder-power indication metadata

	Size (bits)	**Descriptor**
dec_ops_reduction_ratio_from_max	8	unsigned integer
dec_ops_reduction_ratio_from_prev	16	signed integer

The display-power indication metadata contains a list of ms_num_quality_levels pairs, as shown in Table 12.

Table 12 — List of ms_num_quality_levels pairs contained in the display-power indication metadata

	Size (bits)	**Descriptor**
ms_num_quality_levels	4	unsigned integer
ms_rgb_component_for_infinite_psnr	8	unsigned integer
for (i = 1; i < = ms_num_quality_levels; i++) {		
ms_max_rgb_component[i]	8	unsigned integer
ms_scaled_psnr_rgb[i]	8	unsigned integer
}		

7.3 Signalling

Decoder and Display Power Indication Metadata can be carried in metadata tracks within the ISO base media file format (ISO/IEC 14496-12). Metadata shall be signalled in accordance with ISO/IEC 23001-10.

In the context of DASH delivery, a specific Adaptation Set within the MPD can define the available green metadata Representations and their association to the available media Representations. Metadata Representations shall be signalled in MPD in accordance with ISO/IEC 23009-1 and ISO/IEC 23009-3 and illustrated in Annex B.

7.4 Semantics

7.4.1 Decoder-power indication metadata semantics

dec_ops_reduction_ratio_from_max(i) indicates the percentage by which decoding operations are reduced in the i$^{\text{th}}$ Representation compared to the most demanding Representation of the current video Segment:

$$\text{dec\_ops\_reduction\_ratio\_from\_max}(\text{i}) = \text{Floor}\left(\frac{\text{MaxNumDecOps} - \text{NumDecOps}(\text{i})}{\text{MaxNumDecOps}} * 100\right) \qquad (74)$$

MaxNumDecOps is the estimated number of decoding operations required for the most demanding Representation of the current video Segment.

NumDecOps(i) is the estimated number of decoding operations required for the i$^{\text{th}}$ Representation of the current video Segment.

dec_ops_reduction_ratio_from_prev(i) indicates the percentage by which decoding operations are reduced in the current video Segment compared to the previous video Segment for the i^{th} Representation in a given DASH Period. A negative value means an increase in decoding operations:

$$\mathrm{dec\_ops\_reduction\_ratio\_from\_prev}(i) = \mathrm{Floor}\left(\frac{\mathrm{NumPrevDecOps}(i) - \mathrm{NumDecOps}(i)}{\mathrm{NumDecOps}(i)} * 100\right) \quad (75)$$

If the current video Segment is the first Segment of a DASH Period, then dec_ops_reduction_ratio_from_prev(i) = 0.

NumPrevDecOps(i) is the estimated number of decoding operations required for the i^{th} Representation of the previous video Segment in a given DASH Period. If the current video Segment is the first segment of a DASH Period, then NumPrevDecOps(i) = NumDecOps(i).

7.4.2 Display-power indication metadata semantics

ms_num_quality_levels indicates the number of quality levels that are enabled by the metadata.

ms_rgb_component_for_infinite_psnr indicates the average, over the N reconstructed frames of the video Segment, of the largest RGB component (as defined in 3.1) in each of the reconstructed frames.

ms_max_rgb_component[i] indicates, for the i^{th} quality level (1 <= i <= num_quality_levels), the average, over the N reconstructed frames of the video Segment, of the maximum RGB component that is retained in each of the reconstructed frames.

NOTE 1 ms_max_rgb_component[0] = ms_rgb_component_for_infinite_psnr.

ms_scaled_psnr_rgb[i] indicates, for the i^{th} quality level (1 <= i <= num_quality_levels), the average, over the N reconstructed frames in the video segment, of scaled_psnr_rgb[i] computed for each frame as defined in 6.4, with $N_{\mathrm{frames}} = 1$.

NOTE 2 ms_scaled_psnr_rgb[0] is associated with the NQLOP. It is not transmitted, but understood to be mathematically infinite.

8 Metrics for quality recovery after low-power encoding

8.1 General

An encoder can achieve power reduction by encoding alternating high-quality and low-quality Segments, in a segmented delivery mechanism such as DASH. The power reduction occurs because low-complexity encoding mechanisms are used to produce the low-quality Segments. A metric describing the quality of the last picture of each Segment is delivered as metadata to the decoder. The metric is utilized, by the decoder, in conjunction with the last frame of the prior high-quality Segment to enhance the quality of the low-quality Segment and, thereby, ameliorate any negative visual impact. Annex B describes in detail how cross-segment decoding may be used to improve the quality of the low-quality Segments.

8.2 Syntax

The encoder embeds the following message in the last picture of each Segment using the syntax given in Table 13.

Table 13 — Syntax

	Size (bits)	Descriptor
xsd_metric_type	8	unsigned integer
xsd_metric_value	16	unsigned integer

8.3 Signalling

SEI messages can be used to signal quality recovery Metadata in an AVC stream. The green metadata SEI message payload type for AVC shall be set in accordance with ISO/IEC 14496-10. The complete syntax of the green metadata SEI message payload shall be in accordance with Annex A.

The SEI message for green metadata can be used to signal the preceding message as explained in Annex A.

8.4 Semantics

xsd_metric_type indicates the type of the objective quality metric as shown in Table 14. PSNR, as defined in ISO/IEC 23001-10, is the only type currently supported.

Table 14 — Value and description of xsd_metric_type

Value	Description
0x00	PSNR
0x01–0xFF	User-defined

xsd_metric_value contains the metric value of the last picture of the Segment. When xsd_metric_type is 0, then the stored 16-bit unsigned integer xsd_metric_value is interpreted as a floating-point PSNR value (in dB) as follows:

$$\text{PSNR} = \frac{\text{xsd\_metric\_value}}{100} \qquad (76)$$

9 Conformance and reference software

Conformance files and reference software for green metadata shall be used in accordance with Annex C.

Annex A
(normative)

Supplemental enhancement information (SEI) syntax

A.1 Syntax and semantics of green metadata SEI message carried in AVC NAL units

A.1.1 General

This clause describes the payload syntax and semantics if payloadType 56 appears in an AVC NAL unit with nal_unit_type set to 6.

A.1.2 Syntax

Table A.1 — Syntax — AVC NAL unit

	Descriptor
green_metadata(payload_size)	
green_metadata_type	u(8)
switch (**green_metadata_type**) {	
case 0:	
period_type	u(8)
if (period_type == 2) || (period_type == 7) {	
num_seconds	u(16)
}	
else if (period_type == 3) || (period_type == 8) {	
num_pictures	u(16)
}	
if (period_type == 8) {	
temporal_map	
for (t=0; t<8; t++) {	
if ((temporal_map>>t)%2 == 1)	
num_pictures_in_temporal_layers[t]	
}	
}	
if (period_type <= 3) {	
portion_non_zero_8x8_blocks	u(8)
portion_intra_predicted_macroblocks	u(8)
portion_six_tap_filterings	u(8)
portion_alpha_point_deblocking_instances	u(8)
}	
else if (period_type == 4) {	
for (i=0; i<= num_slice_groups_minus1; i++) {	
num_slices_minus1[i]	u(16)
}	

Table A.1 *(continued)*

for (i=0; i<= num_slice_groups_minus1; i++) {	
for (j=0; j<=num_slices_minus1[i]; j++) {	
first_mb_in_slice[i][j]	u(16)
portion_non_zero_8x8_blocks[i][j]	u(8)
portion_intra_predicted_macroblocks[i][j]	u(8)
portion_six_tap_filterings[i][j]	u(8)
portion_alpha_point_deblocking_instances[i][j]	u(8)
}	
}	
}	
else if (period_type >= 5) && (period_type <= 8) {	
num_layers_minus1	u(16)
for (l=0; l<= num_layers_minus1; l++) {	
picture_parameter_set_id[l]	u(8)
priority_id[l]	u(6)
dependency_id[l]	u(3)
quality_id[l]	u(4)
temporal_id[l]	u(3)
portion_non_zero_8x8_blocks[l]	u(8)
portion_intra_predicted_macroblocks[l]	u(8)
portion_six_tap_filterings[l]	u(8)
portion_alpha_point_deblocking_instances[l]	u(8)
}	
}	
break;	
case 1:	
xsd_metric_type	u(8)
xsd_metric_value	u(16)
break;	
default:	
}	

A.1.3 Semantics

green_metadata_type specifies the type of metadata that is present in the SEI message. If green_metadata_type is 0, then complexity metrics are present. Otherwise, if green_metadata_type is 1, then metadata enabling quality recovery after low-power encoding is present. Other values of green_metadata_type are reserved for future use by ISO/IEC.

A.2 Syntax and semantics of green metadata SEI message carried in HEVC NAL units

A.2.1 General

This clause describes the payload syntax and semantics if payloadType 56 appears in an HEVC NAL unit with nal_unit_type set to PREFIX_SEI_NUT.

A.2.2 Syntax

Table A.2 — Syntax — HEVC NAL unit

	Descriptor
green_metadata(payload_size)	
green_metadata_type	u(8)
switch (**green_metadata_type**) {	
case 0:	
period_type	u(8)
if (period_type == 2) {	
num_seconds	u(16)
}	
else if (period_type == 3) {	
num_pictures	u(16)
}	
if (period_type <= 3) {	
portion_non_zero_blocks_area	u(8)
if (portion_non_zero_blocks_area != 0) {	
portion_8x8_blocks_in_non_zero_area	u(8)
portion_16x16_blocks_in_non_zero_area	u(8)
portion_32x32_blocks_in_non_zero_area	u(8)
}	
portion_intra_predicted_blocks_area	u(8)
if (portion_intra_predicted_blocks_area == 255) {	
portion_planar_blocks_in_intra_area	u(8)
portion_dc_blocks_in_intra_area	u(8)
portion_angular_hv_blocks_in_intra_area	u(8)
}	
else {	
portion_blocks_a_c_d_n_filterings	u(8)
portion_blocks_h_b_filterings	u(8)
portion_blocks_f_i_k_q_filterings	u(8)
portion_blocks_j_filterings	u(8)
portion_blocks_e_g_p_r_filterings	u(8)
}	
portion_deblocking_instances	u(8)
}	
else if(period_type == 4) {	
max_num_slices_tiles_minus1	u(16)
for (t=0; t<=max_num_slices_tiles_minus1; t++) {	
first_ctb_in_slice_or_tile[t]	u(16)
portion_non_zero_blocks_area[t]	u(8)
if (portion_non_zero_blocks_area[t] != 0) {	
portion_8x8_blocks_in_non_zero_area[t]	u(8)
portion_16x16_blocks_in_non_zero_area[t]	u(8)
portion_32x32_blocks_in_non_zero_area[t]	u(8)

Table A.2 *(continued)*

}	
portion_intra_predicted_blocks_area[t]	u(8)
if (portion_intra_predicted_blocks_area[t] == 255) {	
portion_planar_blocks_in_intra_area[t]	u(8)
portion_dc_blocks_in_intra_area[t]	u(8)
portion_angular_hv_blocks_in_intra_area[t]	u(8)
}	
else {	
portion_blocks_a_c_d_n_filterings[t]	u(8)
portion_blocks_h_b_filterings[t]	u(8)
portion_blocks_f_i_k_q_filterings[t]	u(8)
portion_blocks_j_filterings[t]	u(8)
portion_blocks_e_g_p_r_filterings[t]	u(8)
}	
portion_deblocking_instances[t]	u(8)
}	
}	
break;	
case 1:	
xsd_metric_type	u(8)
xsd_metric_value	u(16)
break;	
default:	
}	

A.2.3 Semantics

green_metadata_type specifies the type of metadata that is present in the SEI message. If green_metadata_type is 0, then complexity metrics are present. Otherwise, if green_metadata_type is 1, then metadata enabling quality recovery after low-power encoding is present. Other values of green_metadata_type are reserved for future use by ISO/IEC.

Annex B
(informative)

Implementation guidelines for the usage of green metadata

B.1 Codec dynamic voltage frequency scaling for decoder-power reduction

B.1.1 General

Codec dynamic voltage frequency scaling (C-DVFS) uses the DVFS technique to scale the voltage and operating frequency of the CPU to achieve power savings while decoding a bitstream. Typically, the dynamic power consumption of a CMOS circuit increases monotonically with the operating frequency. The power-optimization module at the receiver extracts the complexity metrics (CMs) metadata that indicates picture-decoding complexity. It uses these CMs to determine and set the optimum operating voltage and frequency of the CPU so that video pictures are correctly decoded with minimal power consumption. By embedding these CMs as metadata into the bitstream at the encoder, C-DVFS enabled receivers achieve power reduction.

B.1.2 Derivation of the complexity metrics

5.2.2 specifies these four CMs associated to AVC: portion_non_zero_8x8_blocks, portion_intra_predicted_macroblocks, portion_six_tap_filterings and portion_alpha_point_deblocking_instances. The computation of the first two CMs, as explained in 5.2.4, is straightforward. However, computation of portion_six_tap_filterings and portion_alpha_point_deblocking_instances is more involved. To provide a better understanding of these two CMs, the next two subclauses describe how MaxNumSixTapFilteringsPic(i) and MaxNumAlphaPointDeblockingInstancesPic(i) are derived.

5.2.2 specifies these five CMs associated to HEVC: portion_blocks_a_c_d_n_filterings, portion_blocks_h_b_filterings, portion_blocks_f_i_k_q_filterings, portion_blocks_j_filterings and portion_blocks_e_g_p_r_filterings. To provide a better understanding of these five CMs, the different sub-sample position a, b, c, d, e, f, g, h, i, j, k, n, p, q and r, are represented in Figure B.1, where upper-case letters represent integer samples and lower-case letters represent sub-sample positions derived.

B.1.2.1 Deriving the worst-case, largest value for MaxNumSixTapFilteringsPic(i)

To determine MaxNumSixTapFilteringsPic(i), the following terms, as defined in ISO/IEC 14496-10, are referenced: motion vector, PicSizeInMbs, reference picture list. At the decoder, the worst-case, largest number of 6-tap filterings (STFs) occurs in a picture when all partitions consist of 4x4 blocks that are interpolated. The 4x4 blocks produce the largest number of STFs because the overhead from interpolating samples that are outside the block is larger for 4x4 blocks than for 8x8 blocks as explained below.

In Figure B.1, upper-case letters represent integer samples and lower-case letters represent fractional sample positions. Subscripts are used to indicate the integer sample that is associated with a fractional sample position. The subsequent analysis is for the worst-case largest number of STFs for the interpolation of the 4x4-block consisting of samples G, H, I, J, M, N, P, Q, R, S, V, W, T, U, X, Y. This interpolation shall be performed when a motion vector (MV) points to one of the following fractional-sample positions: a_G, b_G, c_G, d_G, e_G, f_G, g_G, h_G, i_G, j_G, k_G, n_G, p_G, q_G, r_G. If the MV points to a_G, then the decoder shall compute a_G and the 15 points (a_H, a_I, ...) that have the same respective relative locations to H, I, J, M, N, P, Q, R, S, V, W, T, U, X, Y that a_G has to G. Similarly, the decoder would need to compute 16 points for each of the other fractional-sample positions (b_G, c_G, ..., r_G) that the MV could point to. To determine the worst-case largest number of STFs for the interpolation of the 4x4 block, here is a count of the STFs required for each fractional-sample position that the MV could point to.

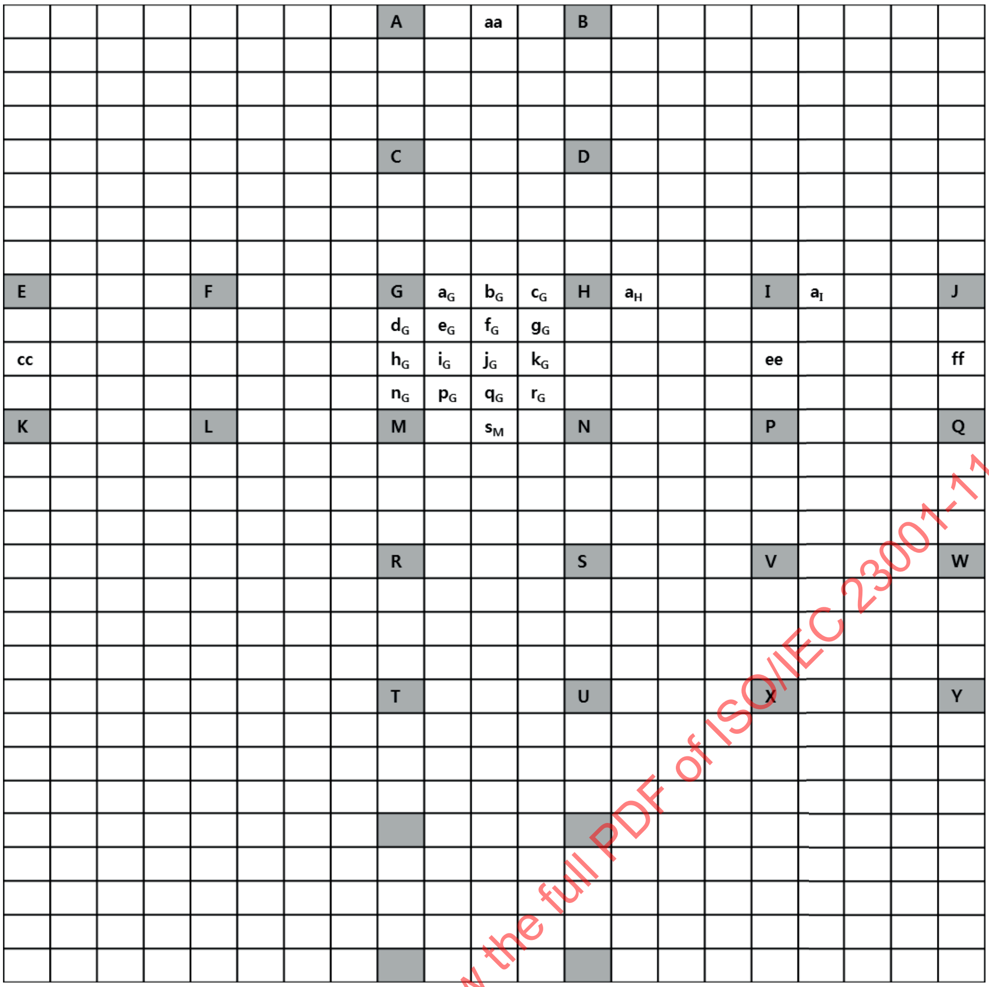

Figure B.1 — Quarter-sample interpolation of the 4x4-block consisting of samples G, H, I, J, M, N, P, Q, R, S, V, W, T, U, X, Y

1) If the MV points to b_G, then to interpolate b_G, the decoder shall apply 1 STF to E, F, G, H, I, J which are already available as integer samples. So 16 STFs are needed to compute b_G, ..., b_Y for the 4x4 block.

2) If the MV points to h_G, then to interpolate h_G, the decoder shall apply 1 STF to A, C, G, M, R, T which are already available as integer samples. So 16 STFs are needed to compute h_G, ..., h_Y for the 4x4 block.

3) If the MV points to j_G, then to interpolate j_G, the decoder shall apply 6 STFs to compute aa, bb, b_G, s_M, gg, hh because these are unavailable. Next, 1 STF is needed to compute j_G from aa, bb, b_G, s_M, gg, hh. So 7 STFs are required for j_G.

   a) To get j_M, the decoder needs bb, b_G, s_M, gg, hh, ii. Only ii is unavailable. So 2 STFs are needed for j_M (one for ii and one for j_M).

   b) To get j_R, the decoder needs 2 STFs (one for jj and one for j_R).

   c) To get j_T, the decoder needs 2 STFs (one for kk and one for j_T).

d) Therefore, for j_G, j_M, j_R and j_T, the decoder needs 7 + 2 + 2 + 2 = 13 STFs. Since the computation is identical for each of the four columns GMRT, HNSU, IPVX and JQWY, the decoder needs 13 * 4 = 52 STFs to compute j_G, ... j_Y for the 4x4 block.

4) If the MV points to a_G, then to interpolate a_G, the decoder needs 1 STF to get b_G (from (1)) and therefore 16 STFs to compute a_G, ..., a_Y for the 4x4 block.

5) If the MV points to c_G, then to interpolate c_G, the decoder needs 1 STF to get b_G (from (1)) and therefore 16 STFs to compute c_G, ..., c_Y for the 4x4 block.

6) If the MV points to d_G, then to interpolate d_G, the decoder needs 1 STF to get h_G (from (2)) and therefore 16 STFs to compute d_G, ..., d_Y for the 4x4 block.

7) If the MV points to n_G, then to interpolate n_G, the decoder needs 1 STF to get h_G (from (2)) and therefore 16 STFs to compute n_G, ..., n_Y for the 4x4 block.

8) If the MV points to f_G, then to interpolate f_G, the decoder needs 7 STFs to get j_G (from (3)). b_G is included in these 7 STFs. Therefore, from (3), 52 STFs are required to compute f_G, ... f_Y for the 4x4 block.

9) If the MV points to i_G, then to interpolate i_G, the decoder needs 7 STFs to get j_G. h_G is computed by one of these 7 STFs. Therefore, 52 STFs are required to compute i_G, ... i_Y for the 4x4 block. For this analysis, the row j_G, j_H, j_I, j_J is computed first (to obtain h_G) and then this process is repeated for the other 3 rows (MNPQ, RSVW, TUXY) in the 4x4 block. Previously, in (3), Column GMRT was analysed first and the analysis was then repeated for the other 3 columns (HNSU, IPVX, JQWY).

10) If the MV points to k_G, then to interpolate k_G, the decoder needs 7 STFs to get j_G. m_G is computed by one of these 7 STFs. Therefore, 52 STFs are required to compute k_G, ... k_Y for the 4x4 block.

11) If the MV points to q_G, then to interpolate q_G, the decoder needs 7 STFs to get j_G. s_G is computed by one of these 7 STFs. Therefore, 52 STFs are required to compute q_G, ... q_Y for the 4x4 block.

12) If the MV points to e_G, then to interpolate e_G, the decoder needs 2 STFs to get b_G and h_G (from (1), (2)). Therefore 32 STFs are needed to compute e_G, ..., e_Y for the 4x4 block.

13) If the MV points to g_G, then to interpolate g_G, the decoder needs 2 STFs to get b_G and m_H. Therefore, 32 STFs are needed to compute g_G, ..., g_Y for the 4x4 block.

14) If the MV points to p_G, then to interpolate p_G, the decoder needs 2 STFs to get h_G and s_G. Therefore, 32 STFs are needed to compute p_G, ..., p_Y for the 4x4 block.

15) If the MV points to r_G, then to interpolate r_G, the decoder needs 2 STFs to get m_G and s_G. Therefore, 32 STFs are needed to compute r_G, ..., r_Y for the 4x4 block.

From (1),...,(15), the worst-case, largest number of STFs is 52, when the MV points to j_G, f_G, i_G, k_G or q_G. Since the overhead of filtering samples outside the block is smaller for larger block sizes, the worst case STFs is when all partitions are 4x4 blocks and two MVs are used for each block (one from each reference picture list). In this case, the worst-case, the largest number of STFs in a picture is

$$\begin{aligned}&\mathrm{MaxNumSixTapFilteringsPic}(\mathrm{i})=(\text{worst-case number of STFs in a 4x4 block})*\\&(\text{worst-case number of reference picture lists})*(\mathrm{PicSizeInMbs})*\\&(\text{number of 4x4 luma blocks in a macroblock})=52*2*\mathrm{PicSizeInMbs}*16=\\&1664*\mathrm{PicSizeInMbs}\end{aligned} \quad (B.1)$$

B.1.2.2 Deriving the worst-case, largest value for MaxNumAlphaPointDeblockingInstancesPic(i)

To determine MaxNumAlphaPointDeblockingInstancesPic(i), the following analysis determines the worst-case, largest number of alpha-point deblocking instances (APDIs) that can occur when deblocking a picture at the decoder. The following terms, as defined in ISO/IEC 14496-10, are referenced: raster scan, PicSizeInMbs.

Consider a macroblock containing a 16x16 luma block in which the samples have been numbered in raster-scan order as shown in Figure B.2. Upper-case roman numerals are used to reference columns of samples and lower-case roman numerals are used to reference rows of samples. For example, Column IV refers to the column of Samples 4, 20, ... 244 and Row xiii refers to the row of samples 193, 194, ..., 208. Edges are indicated by an ordered pair that specifies the columns or rows on either side of the edge. For example, Edge (IV, V) refers to the vertical edge between Columns IV and V. Similarly, Edge (xii, xiii) indicates the horizontal edge between Rows xii and xiii. The leftmost vertical edge and the topmost horizontal edge are denoted by (0, I) and (0, i) respectively.

The maximum number of APDIs occurs when the 4x4 transform is used on each block and a single APDI occurs in every set of eight samples across a 4x4 block horizontal or vertical edge denoted as p_i and q_i with i = 0,...,3 as shown in ISO/IEC 14496-10:—, Figure 8-11.

For the macroblock in Figure B.2, the Vertical Edges (0, I), (IV, V), (VIII, IX) and (XII, XIII) are filtered first. Then the Horizontal Edges (0,i), (iv, v), (viii, ix) and (xii, xiii) are filtered. Now, when Vertical Edge (0, I) is filtered, in the worst-case, an APDI occurs on each row of the edge because the q_0 Samples 1, 17, ... 241 are all APDIs. Therefore, 16 APDIs occur in Vertical Edge (0, I). Similarly, when Vertical Edge (IV, V) is filtered, there are also 16 APDIs corresponding to the 16 (p_0, q_0) sample pairs (20, 21), (36, 37), ... (244, 245). Thus, there are 16 * 4 = 64 APDIs from vertical-edge filtering. After horizontal-edge filtering, there is an additional 64 APDIs because each horizontal edge contributes 16 APDIs. For example, Horizontal Edge (viii, ix) contributes the 16 APDIs corresponding to the (p_0, q_0) sample pairs (113, 129), (114, 130), ..., (128, 144). Hence, in the worst-case, deblocking the luma block in a macroblock produces 128 APDIs.

Next, consider the two chroma blocks corresponding to the luma block in the macroblock. The worst-case number of APDIs is determined by the chroma sampling relative to the luma sampling.

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV	XV	XVI
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ii	17			20	21			24	25			28	29			
iii	33			36	37			40	41			44	45			
iv	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
v	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
vi	81			84	85			88	89			92	93			
vii	97			100	101			104	105			108	109			
viii	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
ix	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
x	145			148	149			152	153			156	157			
xi	161			164	165			168	169			172	173			
xii	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
xiii	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
xiv	209			212	213			216	217			220	221			
xv	225			228	229			232	233			236	237			
xvi	241			244	245			248	249			252	253			256

NOTE Upper-case roman numerals reference columns of samples and lower-case roman numerals reference rows of samples.

Figure B.2 — 16x16 luma block

1) For each chroma block in 4:2:0 format, two vertical edges and two horizontal edges are filtered. Each edge contributes 8 APDIs, in the worst-case. So, 8 * 4 * 2 = 64 APDIs are produced by worst-case deblocking of the two chroma blocks.

2) For 4:2:2 format, two vertical edges and four horizontal edges are filtered. Each vertical edge contributes 16 APDIs and each horizontal edge contributes 8 APDIs. So, 2 * (2 * 16 + 4 * 8) = 128 APDIs are produced by worst-case deblocking of the two chroma blocks.

3) For 4:4:4 format, the worst-case analysis for each chroma block is identical to that of the 16x16 luma block. Therefore, 256 APDIs are produced by worst-case deblocking of the two chroma blocks.

4) Finally, for separate colour planes, the worst-case analysis of a 16x16 block is identical to that a 16x16 luma block.

To conclude, since each picture has PicSizeInMbs macroblocks, the worst-case number of APDIs per picture, is as follows:

$$\begin{aligned}
&\mathrm{MaxNumAlphaPointDeblockingInstancesPic}(i)\\
&\quad = \mathrm{PicSizeInMbs} * (128+64) = 192 * \mathrm{PicSizeInMbs}, \text{ for 4:2:0},\\
&\quad = \mathrm{PicSizeInMbs} * (128+128) = 256 * \mathrm{PicSizeInMbs}, \text{ for 4:2:2}, \qquad \text{(B.2)}\\
&\quad = \mathrm{PicSizeInMbs} * (128+256) = 356 * \mathrm{PicSizeInMbs}, \text{ for 4:4:4},\\
&\quad = 128 * \mathrm{PicSizeInMbs}, \text{ for a single colour plane}
\end{aligned}$$

B.1.3 Example usage of C-DVFS metadata

C-DVFS metadata may be signalled at a slice, layer, picture, group of pictures, or scene level and can therefore be adapted to application requirements. Signalling may be done with SEI messages. With SEI-message signalling, each time the SEI message is encountered by the decoder, a new upcoming period begins. The value period_type indicates whether the new upcoming period is a single picture, a single group of pictures, or a time interval (specified in seconds or number of pictures). Figure 1 shows an example process for metadata extraction, complexity prediction, DVFS control-parameter determination and decoding under DVFS control. As an example, assume that the upcoming period is a single picture. Then, the SEI message is parsed to obtain portion_non_zero_8x8_blocks, portion_intra_predicted_macroblocks, portion_six_tap_filterings and portion_num_alpha_point_deblocking_instances. From these portion values and the corresponding worst-case instances the four CMs are derived: num_non_zero_8x8_blocks (n_{nz}), num_intra_predicted_macroblocks (n_{intra}), num_six_tap_filterings (n_{six}), and num_alpha_point_deblocking_instances (n_{α}). Once the complexity parameters are derived, the total picture complexity ($C_{picture}$) is estimated or predicted according to Formula (B-3):

$$C_{picture} = K_{init} * n_{MB} + k_{bit} * n_{bit} + k_{nz} * n_{nz} + k_{intra} * n_{intra} + k_{six} * n_{six} + k_{\alpha} * n_{\alpha} \qquad \text{(B.3)}$$

where $C_{picture}$ is the total picture complexity. The total number of macroblocks per picture (n_{MB}) and the number of bits per picture (n_{bit}) can be easily obtained after de-packetizing the encapsulated packets and parsing the sequence parameter set. Constants k_{init}, k_{bit}, k_{nz}, k_{intra}, k_{six}, and k_{α} are unit-complexity constants for performing macroblock initialization (including parsed data filling and prefetching), single-bit parsing, non-zero block transform and quantization, intra-block prediction, inter-block six-tap filtering, and deblocking alpha-points filtering, respectively. k_{nz}, k_{intra}, and k_{six} are fixed constants for a typical platform, while k_{init}, k_{bit}, and k_{α} can be accurately estimated using a linear predictor from a previous decoded picture.

Once the picture complexity is determined, the decoder applies DVFS to determine a suitable clock frequency and supply voltage for the decoder. Then, the decoder can decode the video picture at the appropriate clock frequency and supply voltage.

The DVFS-enabling SEI message can be inserted into the bitstream on a slice-by-slice, layer-by-layer, picture-by-picture, scene-by-scene, or even time-interval-by-time-interval basis, depending on the underlying application. Therefore, the SEI message can be inserted once at the start of each picture, scene, or time interval. A scene-interval or time-interval inserted message requires less overhead than a picture-level inserted message. For processors that do not support high-frequency DVFS (e.g. adapting at 33 ms for 30 Hz video playback), setting period_type to an interval is preferable to setting period_type to a picture. Once all complexity metrics are obtained from the SEI message, the decoder estimates the complexity for the next slice, layer, picture, group of pictures, or time interval as indicated by period_type. This complexity is then used to adjust the voltage and frequency for the upcoming period.

In a hardware (ASIC) implementation, instead of deriving decoding complexity and using it directly to control a single clock frequency in a DVFS scheme, the ASIC can be designed so that it includes several distinct clock domains, each of which corresponds to one of the terms in Formula (B-3). Greater power reduction can be obtained by using such a flexible ASIC with distinct clock domains. For example, six clock domains in the ASIC can control the following six sections of the ASIC: macroblock initialization, bit parsing, transform and quantization, intra-block prediction, interpolation, and deblocking. To achieve fine-grained DVFS adjustments, the clock frequencies in each domain may be varied in proportion to the corresponding term in Formula B-3. Accordingly, the preceding clock domains can have instantaneous clock frequencies that are respectively proportional to the following terms: $k_{init} * n_{MB}$, $k_{bit} * n_{bit}$, $k_{nz} * n_{nz}$, $k_{intra} * n_{intra}$, $k_{six} * n_{six}$, and $k_{\alpha} * n_{\alpha}$.

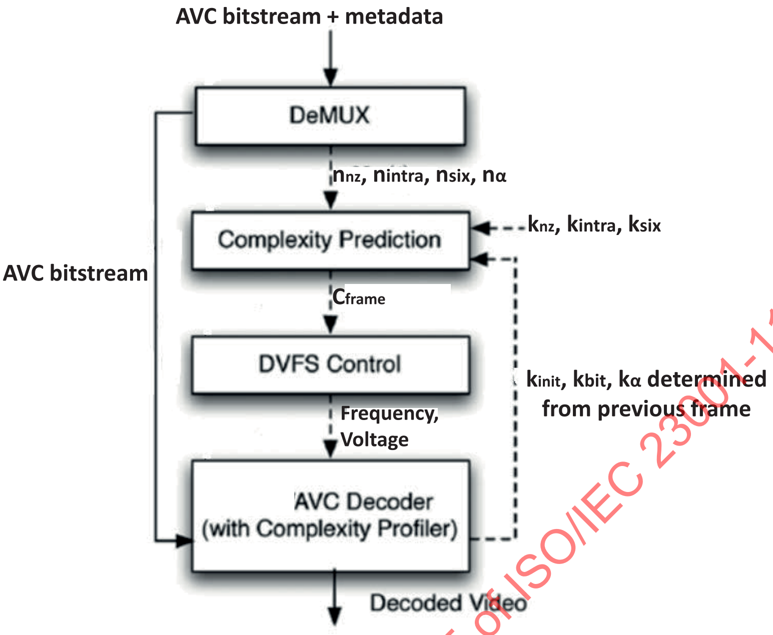

Figure B.3 — Example of parsing, complexity prediction and DVFS control

B.2 Display adaptation

B.2.1 General

Display Adaptation (DA) achieves power savings by scaling up the RGB components in the reconstructed frames while reducing the backlight or voltage proportionally. The decreased backlight or voltage reduces display power consumption while still producing the same perceived display. The metadata in 6.2.1 can be stored using the file format specified in ISO/IEC 23001-10 or the metadata can be carried in a transport stream as specified in ISO/IEC 13818-1.

B.2.2 Example usage of display-adaptation metadata

B.2.2.1 General

The metadata scaled_psnr_rgb[i] indicates the PSNR for the ith quality level. At the transmitter, reconstructed frames are available within the encoder and ScaledFrames[i] is estimated by saturating all RGB components of reconstructed frames to max_rgb_component[i]. The ScaledFrames[i] thus obtained are what would be perceived at the display after the receiver scales the RGB components of reconstructed frames by (PeakSignal/max_rgb_component[i]), PeakSignal variable being the peak signal, and then applies the backlight scaling factor, b = (max_rgb_component[i]/PeakSignal) to the LCD backlight. scaled_psnr_rgb[i] is computed at the transmitter using PeakSignal and by assuming that the noise is the difference between ScaledFrames[i] and reconstructed frames accumulated over R, G and B components, as explained in 6.4.

The receiver examines the (num_quality_levels + 1) pairs of metadata and selects the pair (max_rgb_component[iSelected], scaled_psnr_rgb[iSelected]) for which scaled_psnr_rgb[iSelected] is an acceptable quality level. Then, the receiver derives DA scaling factors from max_rgb_component[iSelected]. Finally, the display scales the RGB components of reconstructed frames by PeakSignal/max_rgb_component[iSelected] and it scales the backlight or voltage level by max_rgb_component[iSelected]/

PeakSignal. After backlight scaling, the displayed pixels are perceived as ScaledFrames[iSelected]. The metadata clearly enables a trade-off between quality (PSNR) and power reduction (backlight scaling factor).

The following power-saving protocol can be implemented in a mobile device. The user specifies a list of N acceptable PSNR quality levels Q[1], ..., Q[N], where Q[1] > Q[2] > ...> Q[N] and a list of remaining battery life levels (RBLLs) RBLL[1], ..., RBLL[N] so that RBLL[1] > RBLL[2] > ... > RBLL[N]. For example, consider N = 3 and Q[1] = 40, Q[2] = 35, Q[3] = 25 with RBLL[1] = 70 %, RBLL[2] = 40 % and RBLL[3] = 0 %. When the user watches a video, the device monitors the actual RBLL, denoted RBLLactual, of the device and selects RBLL[iSelected] so that RBLL[iSelected-1] > RBLLactual > RBLL[iSelected], where RBLL[0] = 100 %. For each frame to be displayed, the device examines the display-adaptation metadata and selects the pair indexed by jSelected for which Q[iSelected-1] > scaled_psnr_rgb[jSelected] > Q[iSelected], where Q[0] = infinity. The metadata max_rgb_component[jSelected] is then used to determine display-adaptation scaling parameters. Thus, the device implements a protocol that strikes a balance between perceived quality and power-saving. The balance is tilted toward quality when the RBLL is high but shifts toward power saving as the battery is depleted.

B.2.2.2 Example usage of display-adaptation metadata for contrast enhancement

At low quality levels, contrast enhancement significantly improves perceived visual quality, especially for bright content. To enhance contrast at the lowest quality level associated with the backlight scaling factor b = (max_rgb_component[num_quality_levels]/PeakSignal), the receiver first examines lower_bound. If it is greater than zero, then contrast enhancement metadata is available and the receiver stores upper_bound. The presentation subsystem performs contrast enhancement by setting the backlight scaling factor to b = (max_rgb_component[num_quality_levels]/PeakSignal), and for each RGB component, x, of reconstructed frames, the following scaling to S(x) is performed:

S(x) = 0, for x in [0, lower_bound],

= PeakSignal * (x-lower_bound)/(upper_bound – lower_bound),

for x in (lower_bound, upper_bound),

= PeakSignal, for x in [upper_bound, PeakSignal]

The interval (lower_bound, upper_bound) is mapped to the interval (0, PeakSignal). Then, after applying the backlight scaling factor, b, to the display, the interval (lower_bound, upper_bound) is perceived visually as the interval (0, b * PeakSignal). Therefore, for RGB components within the interval (lower_bound, upper_bound), the perceived contrast enhancement is proportional to b * PeakSignal/(upper_bound – lower_bound). This expression simplifies to b/(upper_bound – lower_bound), because PeakSignal is a constant. For RGB components within the intervals [0, lower_bound] and [upper_bound, PeakSignal], all contrast is lost because these intervals are mapped to 0 and PeakSignal, respectively.

From the preceding observation, it is clear that the contrast is maximized by determining lower_bound and upper_bound so that the majority of RGB components lie within the interval (lower_bound, upper_bound). Therefore, the optimal contrast-enhancement metadata is computed by the following process, at the transmitter. First, determine the BacklightScalingFactor corresponding to the lowest quality level as b = max_rgb_component[num_quality_levels]/PeakSignal. Then, invoke the following pseudocode function get_contrast_metadata() to determine lower_bound and upper_bound.

```
// Given RGB components, x, of reconstructed frames with
// cumulative distribution function, C(x), the function get_contrast_metadata() returns
// lower_bound and upper_bound.
[lower_bound, upper_bound] = get_contrast_metadata(C(x)) {
// C(x): Cumulative distribution function of RGB components of reconstructed frames.
max_enhancement = 0;
for (lower_bound = 0; lower_bound < PeakSignal; lower_bound++){
  for (upper_bound = lower_bound; upper_bound < PeakSignal; upper_bound++){
    enhancement = (C(upper_bound) – C(lower_bound)) / (upper_bound – lower_bound)
    if (enhancement > max_enhancement) {
      max_enhancement = enhancement;
      best_lower_bound = lower_bound;
      best_upper_bound = upper_bound;
    }
  }
 }
return (best_lower_bound, best_upper_bound);
}
```

Although the metadata computed by get_contrast_metadata() is optimal for each frame, flicker artefacts may occur when the video is viewed due to large differences between lower_bound (or upper_bound) settings on successive video frames. To avoid such flicker, the lower_bound and upper_bound metadata should be smoothed temporally using the pseudo-code function smooth_contrast_metadata() shown below.

```
// Given a video sequence with frameNum in [1,...,N], first smooth the lower bounds by
// applying the function recursively to all frames by issuing
// smooth_contrast_metadata(LowerBounds,1),
// ...
// smooth_contrast_metadata(LowerBounds,N)
// Then smooth the upper bounds by issuing
// smooth_contrast_metadata(UpperBounds,1),
// ...
// smooth_contrast_metadata(UpperBounds,N)
// where
// LowerBounds: vector of lower_bound metadata for the N frames
// UpperBounds: vector of upper_bound metadata for the N frames
void smooth_contrast_metadata(Vector, frameNum) {
// Vector: vector of metadata to be smoothed
// frameNum: current frame number
cur = Vector[frameNum]
prev = Vector[frameNum - 1]
if Abs((cur - prev) / prev) > Threshold { // Check whether the metadata variation between
                                          // successive frames exceeds the threshold.
 if (cur < prev) { // if the current frame's metadata are lower than the previous frame's metadata,
                   // then increase the current frame's metadata so that it reaches the acceptable
                   // threshold.
   Vector[frameNum] = prev * (1 - Threshold)
} else { // increase the previous frame's metadata so that it reaches the acceptable
         // threshold. Then adjust the metadata for all preceding frames.
   Vector[frameNum - 1] = cur / (1 + Threshold)
   smooth_contrast_metadata(Vector, frameNum - 1)
 }
}
```

The value of Threshold is display independent and can be set to 0,015, which corresponds to a 1,5 % metadata variation between successive frames.

B.2.2.3 Preventing flicker arising from control latency

If DA metadata were unavailable, then to implement DA, the display would have to estimate max_rgb_component[i] and immediately adjust the backlight (or voltage). This is impossible in most practical implementations because there is a significant latency of D milliseconds between the instant when the backlight scaling control is applied and the instant when the backlight actually changes, in response to the control. If D is sufficiently large, then the backlight values are not synchronized with the displayed frames and flickering is visible. Fortunately, DA metadata eliminates this flickering. Because the receiver obtains the metadata in advance, the backlight scaling factor can be applied D milliseconds ahead of the video frame with which that scaling factor is associated. Therefore, by transmitting metadata, the latency issue is solved and the backlight scaling factor will be set appropriately for each frame. This avoids flicker from backlight changes during video display.

B.2.2.4 Metadata for DA on displays with control-frequency limitations

Besides eliminating flicker arising from backlight-control latency, DA metadata can also enable DA to be applied to displays in which the backlight (or voltage) cannot be changed frequently. For such displays, once the backlight has been updated, it shall retain its value for a time interval that spans the duration of some number of successive frames. After the time interval has elapsed, the backlight may be updated again. DA metadata allows the backlight to be set appropriately for the specified time interval so that maximal power reduction and minimal RGB-component saturation occurs. This appropriate backlight value is determined by aggregating the RGB component histograms in all successive frames in each time interval over which the backlight shall remain constant. The aggregated histograms are then used to derive DA metadata, as explained in preceding subclauses. To enable this mode of operation, the receiver shall signal to the transmitter, constant_backlight_voltage_time_interval, the time interval over which the backlight (or voltage) shall remain constant. Alternatively, the transmitter may assume a reasonable value for constant backlight voltage time interval.

On currently available displays, setting constant_backlight_voltage_time_interval to 100 milliseconds is sufficient to prevent flicker. Therefore, setting num_constant_backlight_voltage_time_intervals = 1 and constant_backlight_voltage_time_interval[0] = 100 is sufficient to prevent flicker arising from control-frequency limitations. However, in the future, a new display technology with constant_backlight_voltage_time_interval significantly different from 100 milliseconds may be invented. During the transition period from the current display technology to the new display technology, two types of displays will be widely used and it will be necessary to set num_constant_backlight_voltage_time_intervals = 2, to support both display types. The preceding mode of operation assumes that a signalling mechanism from the receiver to the transmitter does not exist.

However, if such a signalling mechanism does exist, then the receiver can explicitly signal constant_backlight_voltage_time_interval to the transmitter as explained in 6.2.2 and 6.3.3. If the transmitter is additionally capable of re-computing the display adaptation metadata to be consistent with the signalled constant_backlight_voltage_time_interval, then the re-computed metadata can subsequently be provided to the receiver.

B.2.2.5 DA metadata to prevent flicker from large variations

On some platforms, besides the flicker that arises from control latency and control-frequency limitations, flicker can also occur due to a large difference between the backlight (or voltage) settings (defined as BacklightScalingFactor in 6.4) of successive video frames. To avoid such flicker, a transmitter can use the function below to adjust the backlight setting of each frame. Specifically, if the relative backlight variation between a frame and its predecessor is larger than a threshold, then the backlight values of all preceding frames shall be adjusted. This adjustment is done at the transmitter after metadata has been computed using one of the methods described in the preceding subclauses.

For example, for a targeted quality level, the transmitter would estimate max_rgb_component and the corresponding BacklightScalingFactor for each of N frames. Given max_variation (normalized to 255), the transmitter applies adjust_backlight() with the specified max_variation threshold computed as the floating-point number (max_variation/2 048). This function adjusts the vector of BacklightScalingFactor values for the *N* frames so that the relative backlight variation between successive frames is less than max_variation. After the backlight values have been adjusted, the DA metadata is modified, if necessary, to be consistent with the adjusted backlight values.

```
// Given a video sequence with frameNum in [1,...,N], apply the function
// recursively to all frames by issuing
// adjust_backlight(Backlights,1,max_variation),
// ...
// adjust_backlight(Backlights,N,max_variation)
void adjust_backlight(Backlights, frameNum, max_variation) {
// Backlights: vector of BacklightScalingFactor values
// frameNum: current frame number
// max_variation: maximum permissible backlight variation between two
// consecutive backlight values
cur = Backlights[frameNum]
prev = Backlights[frameNum - 1]
if Abs((cur - prev) / prev) > max_variation { // Check whether the backlight variation between
                                             // successive frames exceeds the threshold.
if (cur < prev) { // if the current frame's backlight is lower than the previous frame's backlight,
                  // then increase the current frame's backlight so that it reaches the acceptable threshold.
  Backlights[frameNum] = prev * (1 - max_variation)
}else { // increase the previous frame's backlight so that it reaches the acceptable
        // threshold.Then adjust the backlights for all preceding frames.
  Backlights[frameNum - 1] = cur / (1 + max_variation)
  adjust_backlight(Backlights, frameNum - 1, max_variation)
}
}
```

For a given display, large values of max_variation induce more flicker but also save more power. Therefore, the selected value of max_variation is a compromise between flicker reduction and power saving. The max_variation metadata guarantees that the receiver will not experience flicker because the backlights are adjusted specifically for the receiver's display.

On currently available displays, setting max_variation = 0,015 * 2 048 is sufficient to prevent flicker. Therefore, setting num_max_variations = 1 and max_variation = 0,015 * 2 048 is sufficient to prevent flicker arising from control-frequency limitations. However, in the future, a new display technology with max_variation significantly different from 0,015 * 2 048 may be invented. During the transition period from the current display technology to the new display technology, two types of displays will be widely used and it will be necessary to set num_max_variations = 2, to support both display types. The preceding mode of operation assumes that a signalling mechanism from the receiver to the transmitter does not exist.

However, if such a signalling mechanism does exist, then the receiver can explicitly signal max_variation to the transmitter as explained in 6.3.3. If the transmitter is additionally capable of re-computing the display adaptation metadata to be consistent with the signalled max_variation, then the re-computed metadata can subsequently be provided to the receiver.

B.3 Energy-efficient media selection in adaptive streaming

B.3.1 General

This clause explains how the green metadata for adaptive streaming can be computed at the server and how such metadata can be used at the client.

B.3.2 Green metadata production and transmission at the server side

Given *N* video Representations, the decoder-power indication metadata dec_ops_reduction_ratio_from_max(i) (DOR-Ratio-Max(i)) and dec_ops_reduction_ratio_from_prev(i) (DOR-Ratio-Prev(i)) are computed by the encoding system and provided by the server for i = 0 to N-1, as shown in Figure B.4. The display-power indication metadata is computed from one Representation.

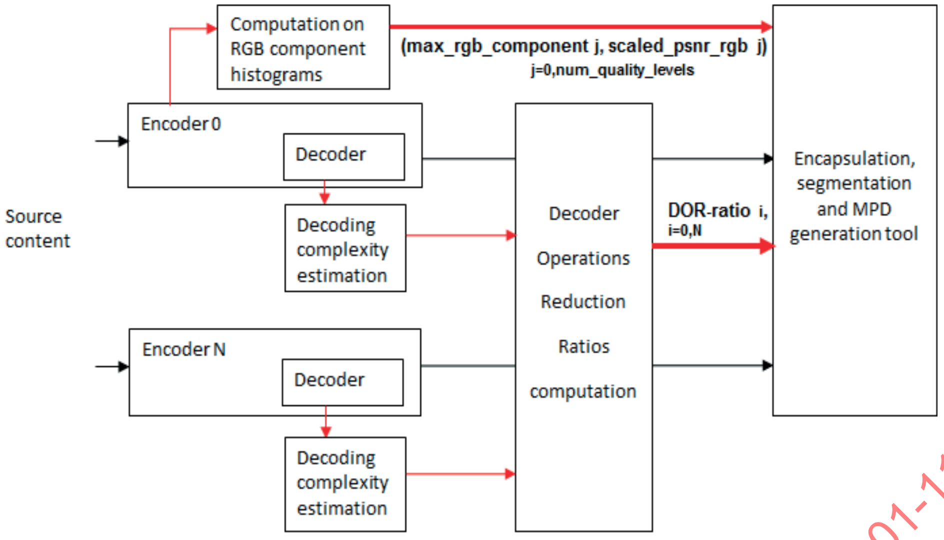
Computation on RGB component histograms
(max_rgb_component j, scaled_psnr_rgb j)
j=0,num_quality_levels
Source content
Encoder 0
Decoder
Decoding complexity estimation
Encoder N
Decoder
Decoding complexity estimation
Decoder Operations Reduction Ratios computation
DOR-ratio i, i=0,N
Encapsulation, segmentation and MPD generation tool

Figure B.4 — Green metadata computation and insertion

The DOR-Ratio-Max(i) associated with each video Representation i of a Segment is computed as the power-saving ratio from the most demanding video Representation produced for the Segment, as defined in 7.4.1.

The DOR-Ratio-Prev(i) associated with each video Representation i of a Segment is computed as the power-saving ratio from the previous Segment of the same Representation, as defined in 7.4.1.

To produce the normative green metadata DOR-Ratio-Max(i) and DOR-Ratio-Prev(i) for a given Segment, the encoding system needs to estimate the decoding complexity of each video Representation, as a number of processing cycles.

Each sample which contains the DOR-Ratio values is then stored in a specific metadata file "id/$Time$.mp4m" (one for each Segment) using the format specified in ISO/IEC 23001-10. In the DASH context, the metadata files created for one or multiple video Representations are considered as metadata Representations. The available metadata Representations are signalled in a specific Adaptation Set within the MPD. The association of a metadata Representation with a media Representation is signalled in the MPD through the @associationId and @associationType attributes. A metadata Segment and its associated media Segment(s) are time aligned on Segment boundaries.

The decoder-power indication metadata Representation is associated with a single media Representation as shown in Figure B.5.

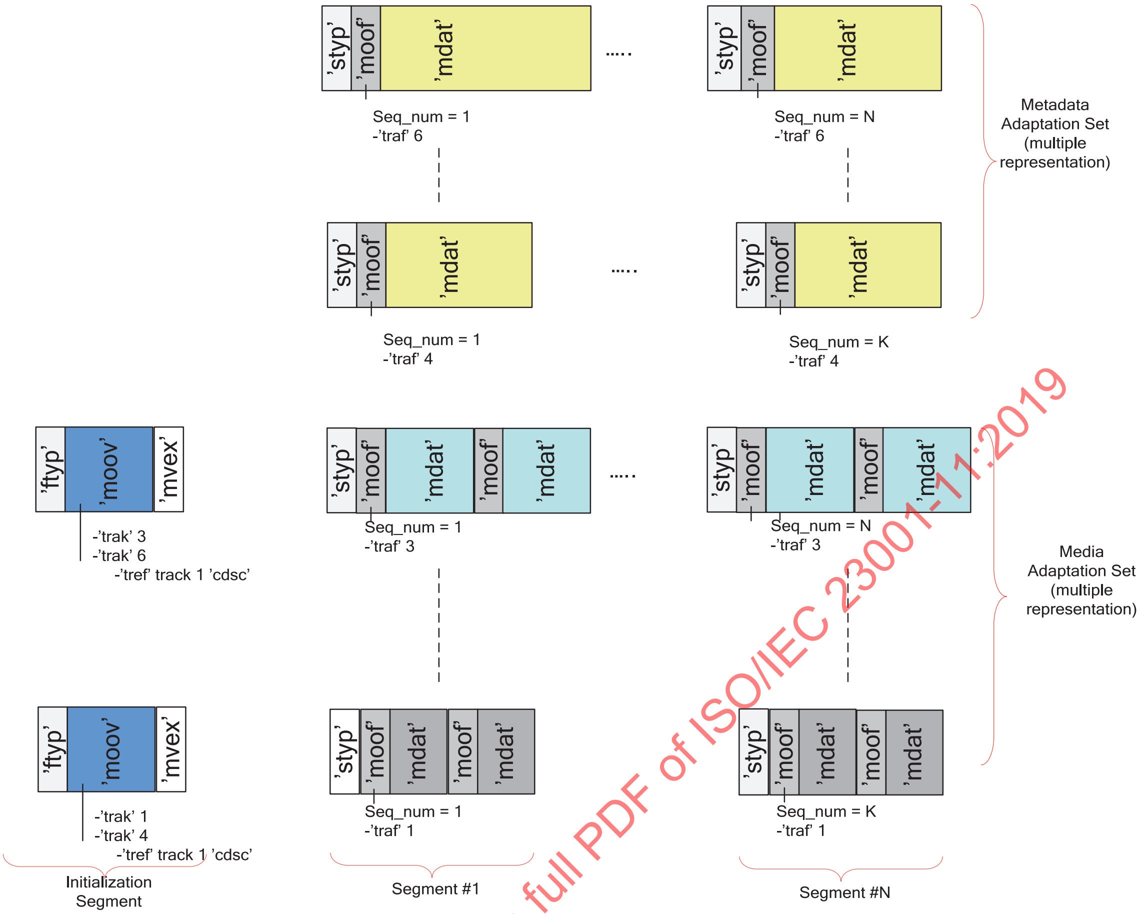

Figure B.5 — One metadata Representation for one media Representation

The following XML file provides an example of an MPD for decoder-power indication metadata:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<MPD
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:DASH:schema:MPD:XXXX"
  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:xxxx"
  type="dynamic"
  minimumUpdatePeriod="PT2S"
  timeShiftBufferDepth="PT30M"
  availabilityStartTime="2011-12-25T12:30:00"
  minBufferTime="PT4S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011">

  <BaseURL>http://cdn1.example.com/</BaseURL>
  <BaseURL>http://cdn2.example.com/</BaseURL>

  <Period>
    <!-- Video -->
    <AdaptationSet
      id="video"
      mimeType="video/mp4"
      codecs="avc1.4D401F"
      frameRate="30000/1001"
      segmentAlignment="true"
      startWithSAP="1">
      <BaseURL>video/</BaseURL>
      <SegmentTemplate timescale="90000" media="$Bandwidth$/$Time$.mp4v">
        <SegmentTimeline>
          <S t="0" d="180180" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="v0" width="320" height="240" bandwidth="250000"/>
      <Representation id="v1" width="640" height="480" bandwidth="500000"/>
      <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
    </AdaptationSet>
    <!-- English Audio -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.0x40" lang="en" segmentAlignment="0">
      <SegmentTemplate timescale="48000" media="audio/en/$Time$.mp4a">
        <SegmentTimeline>
          <S t="0" d="96000" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="a0" bandwidth="64000" />
    </AdaptationSet>
    <!-- French Audio -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.0x40" lang="fr" segmentAlignment="0">
      <SegmentTemplate timescale="48000" media="audio/fr/$Time$.mp4a">
        <SegmentTimeline>
          <S t="0" d="96000" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="a0" bandwidth="64000" />
    </AdaptationSet>
<!--AdaptationSet carrying Green Video Information for Video  -->
   <AdaptationSet id="green_video" codecs="depi"/>
      <BaseURL>video_green_depi/</BaseURL>
      <SegmentTemplate timescale="90000" media="$id$/$Time$.mp4m">
        <SegmentTimeline>
          <S t="0" d="180180" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="gv0" bandwidth="1000" associationId="v0" associationType="cdsc"/>
      <Representation id="gv1" bandwidth="1000" associationId="v1" associationType="cdsc"/>
      <Representation id="gv2" bandwidth="1000" associationId="v2" associationType="cdsc"/>
    </AdaptationSet>
  </Period>

</MPD>
```

The display-power indication metadata is a list of (ms_num_quality_levels + 1) pairs of the form (ms_max_rgb_component[i], ms_scaled_psnr_rgb[i]) as defined in 7.4.1. This metadata is produced without considering any constraint on max_variation, the maximal backlight variation between two successive frames. It is also assumed that the backlight can be updated on each frame so that constant_backlight_voltage_time_interval is the inter-frame interval. Therefore, the display power-indication metadata provides the maximum power saving for a given quality level.

The display-power indication metadata is stored in a specific metadata file "id/$Time$.mp4m" (one for each Segment) using the format specified in ISO/IEC 23001-10. The display-power indication metadata Representation is associated with all the available media Representations as shown in Figure B.6.

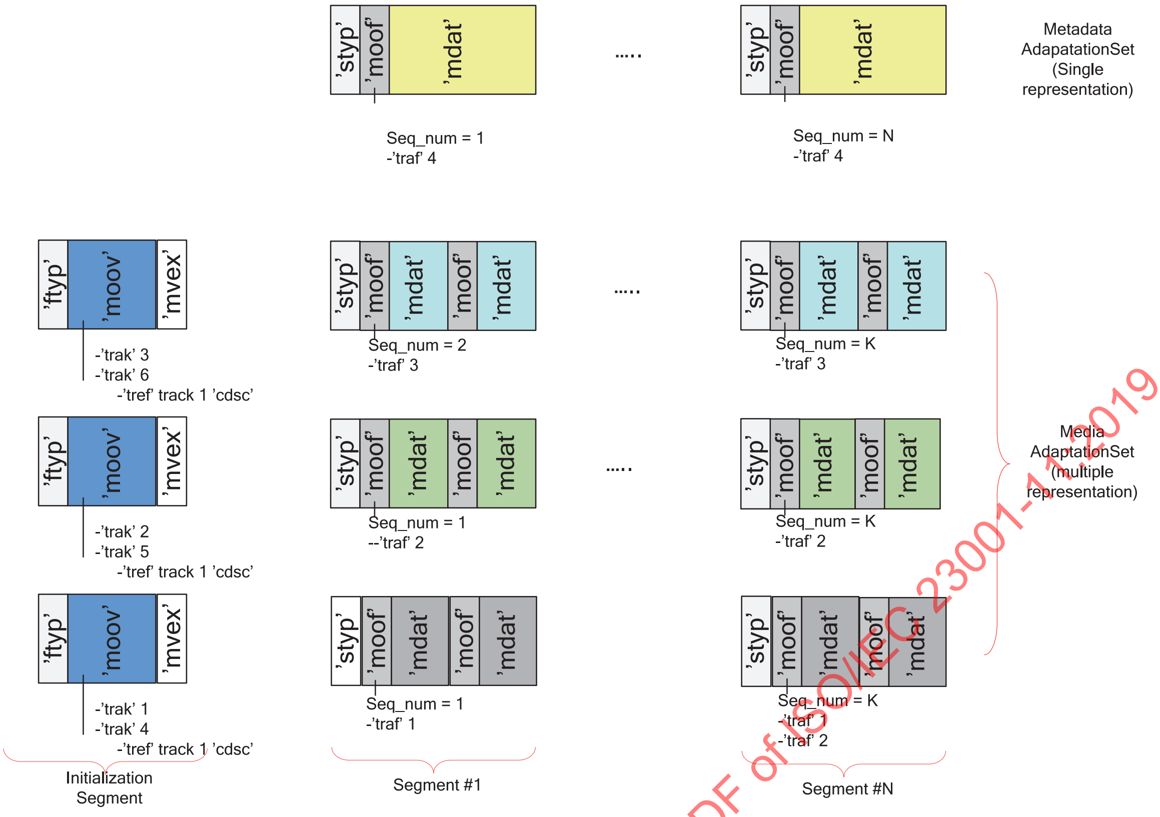

Figure B.6 — One metadata Representation for all media Representations

The following XML file provides an example of an MPD for display-power indication metadata:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<MPD
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:DASH:schema:MPD:XXXX"
  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:xxxx"
  type="dynamic"
  minimumUpdatePeriod="PT2S"
  timeShiftBufferDepth="PT30M"
  availabilityStartTime="2011-12-25T12:30:00"
  minBufferTime="PT4S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011">

  <BaseURL>http://cdn1.example.com/</BaseURL>
  <BaseURL>http://cdn2.example.com/</BaseURL>

  <Period>
    <!-- Video -->
    <AdaptationSet
      id="video"
      mimeType="video/mp4"
      codecs="avc1.4D401F"
      frameRate="30000/1001"
      segmentAlignment="true"
      startWithSAP="1">
      <BaseURL>video/</BaseURL>
      <SegmentTemplate timescale="90000" media="$Bandwidth$/$Time$.mp4v">
        <SegmentTimeline>
          <S t="0" d="180180" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="v0" width="320" height="240" bandwidth="250000"/>
      <Representation id="v1" width="640" height="480" bandwidth="500000"/>
      <Representation id="v2" width="960" height="720" bandwidth="1000000"/>
    </AdaptationSet>
    <!-- English Audio -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.0x40" lang="en" segmentAlignment="0">
      <SegmentTemplate timescale="48000" media="audio/en/$Time$.mp4a">
        <SegmentTimeline>
          <S t="0" d="96000" r="432"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="a0" bandwidth="64000" />
    </AdaptationSet>
    <!-- French Audio -->
```