INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 N8242

Klagenfurt, Austria, July 2006

Title: **Joint Draft 7 of SVC Amendment (with proposed changes)**

Status: Output Document of JVT

Purpose: Text

Editor(s) / Contact(s):

Thomas Wiegand
Heinrich Hertz Institute (FhG),
Einsteinufer 37,
D-10587 Berlin, Germany
Tel: +49 (30) 31002-617
Fax: +49 (30) 39272-00
Email: wiegand@hhi.de

Gary Sullivan
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052 USA
Tel: +1 (425) 703-5308
Fax: +1 (425) 706-7329
Email: garysull@microsoft.com

Julien Reichel
GE Security (VisioWave),
Rte. de la Pierre 22,
CH-1024 Ecublens, Switzerland
Tel: +41 (21) 695-0041
Fax: +41 (21) 695-0001
Email: reichel@ieee.org

Heiko Schwarz
Heinrich Hertz Institute (FhG),
Einsteinufer 37,
D-10587 Berlin, Germany
Tel: +49 (30) 31002-226
Fax: +49 (30) 39272-00
Email: heiko.schwarz@hhi.fhg.de

Mathias Wien
Institut für Nachrichtentechnik,
RWTH Aachen University,
D-52056 Aachen, Germany
Tel: +49 (241) 80-27681
Fax: +49 (241) 80-22196
Email: wien@ient.rwth-aachen.de

Source: JVT

CONTENTS

Page

Foreword ..... **xvi**

0 Introduction **1**

0.1 Prologue *1*

0.2 Purpose *1*

0.3 Applications *1*

0.4 Publication and versions of this specification *1*

0.5 Profiles and levels *2*

0.6 Overview of the design characteristics *2*

0.6.1 Predictive coding 3

0.6.2 Coding of progressive and interlaced video 3

0.6.3 Picture partitioning into macroblocks and smaller partitions 3

0.6.4 Spatial redundancy reduction 3

0.7 How to read this specification *4*

1 Scope **4**

2 Normative references **4**

3 Definitions **4**

4 Abbreviations **12**

5 Conventions **13**

5.1 Arithmetic operators *13*

5.2 Logical operators *14*

5.3 Relational operators *14*

5.4 Bit-wise operators *14*

5.5 Assignment operators *14*

5.6 Range notation *15*

5.7 Mathematical functions *15*

5.8 Variables, syntax elements, and tables *16*

5.9 Text description of logical operations *16*

5.10 Processes *17*

6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships **18**

6.1 Bitstream formats *18*

6.2 Source, decoded, and output picture formats *18*

6.3 Spatial subdivision of pictures and slices *23*

6.4 Inverse scanning processes and derivation processes for neighbours *24*

6.4.1 Inverse macroblock scanning process 24

6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process 25

6.4.2.1 Inverse macroblock partition scanning process 25

6.4.2.2 Inverse sub-macroblock partition scanning process 26

6.4.3 Inverse 4x4 luma block scanning process 26

6.4.4 Inverse 8x8 luma block scanning process 26

6.4.5 Derivation process of the availability for macroblock addresses 27

6.4.6 Derivation process for neighbouring macroblock addresses and their availability 27

6.4.7 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames 28

6.4.8 Derivation processes for neighbouring macroblocks, blocks, and partitions 29

6.4.8.1 Derivation process for neighbouring macroblocks 30

6.4.8.2 Derivation process for neighbouring 8x8 luma block 30

6.4.8.3 Derivation process for neighbouring 4x4 luma blocks 31

6.4.8.4 Derivation process for neighbouring 4x4 chroma blocks 31

6.4.8.5 Derivation process for neighbouring partitions 32

6.4.9 Derivation process for neighbouring locations 33

6.4.9.1 Specification for neighbouring locations in fields and non-MBAFF frames 34

6.4.9.2 Specification for neighbouring locations in MBAFF frames 35

7 Syntax and semantics **37**

7.1 Method of specifying syntax in tabular form *37*

7.2 Specification of syntax functions, categories, and descriptors *38*

7.3 Syntax in tabular form *39*

7.3.1 NAL unit syntax 39

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax 40

7.3.2.1 Sequence parameter set RBSP syntax 40

7.3.2.1.1 Scaling list syntax ... 41
7.3.2.1.2 Sequence parameter set extension RBSP syntax ... 41
7.3.2.2 Picture parameter set RBSP syntax ... 42
7.3.2.3 Supplemental enhancement information RBSP syntax ... 43
7.3.2.3.1 Supplemental enhancement information message syntax ... 44
7.3.2.4 Access unit delimiter RBSP syntax ... 44
7.3.2.5 End of sequence RBSP syntax ... 44
7.3.2.6 End of stream RBSP syntax ... 44
7.3.2.7 Filler data RBSP syntax ... 45
7.3.2.8 Slice layer without partitioning RBSP syntax ... 45
7.3.2.9 Slice data partition RBSP syntax ... 45
7.3.2.9.1 Slice data partition A RBSP syntax ... 45
7.3.2.9.2 Slice data partition B RBSP syntax ... 45
7.3.2.9.3 Slice data partition C RBSP syntax ... 46
7.3.2.10 RBSP slice trailing bits syntax ... 46
7.3.2.11 RBSP trailing bits syntax ... 46
7.3.3 Slice header syntax ... 47
7.3.3.1 Reference picture list reordering syntax ... 48
7.3.3.2 Prediction weight table syntax ... 49
7.3.3.3 Decoded reference picture marking syntax ... 50
7.3.4 Slice data syntax ... 51
7.3.5 Macroblock layer syntax ... 52
7.3.5.1 Macroblock prediction syntax ... 53
7.3.5.2 Sub-macroblock prediction syntax ... 54
7.3.5.3 Residual data syntax ... 54
7.3.5.3.1 Residual block CAVLC syntax ... 56
7.3.5.3.2 Residual block CABAC syntax ... 57
7.4 Semantics ... 58
7.4.1 NAL unit semantics ... 58
7.4.1.1 Encapsulation of an SODB within an RBSP (informative) ... 61
7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences ... 61
7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation ... 61
7.4.1.2.2 Order of access units and association to coded video sequences ... 62
7.4.1.2.3 Order of NAL units and coded pictures and association to access units ... 63
7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture ... 64
7.4.1.2.5 Order of VCL NAL units and association to coded pictures ... 65
7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics ... 66
7.4.2.1 Sequence parameter set RBSP semantics ... 66
7.4.2.1.1 Scaling list semantics ... 71
7.4.2.1.2 Sequence parameter set extension RBSP semantics ... 71
7.4.2.2 Picture parameter set RBSP semantics ... 73
7.4.2.3 Supplemental enhancement information RBSP semantics ... 76
7.4.2.3.1 Supplemental enhancement information message semantics ... 76
7.4.2.4 Access unit delimiter RBSP semantics ... 76
7.4.2.5 End of sequence RBSP semantics ... 76
7.4.2.6 End of stream RBSP semantics ... 76
7.4.2.7 Filler data RBSP semantics ... 76
7.4.2.8 Slice layer without partitioning RBSP semantics ... 77
7.4.2.9 Slice data partition RBSP semantics ... 77
7.4.2.9.1 Slice data partition A RBSP semantics ... 77
7.4.2.9.2 Slice data partition B RBSP semantics ... 77
7.4.2.9.3 Slice data partition C RBSP semantics ... 77
7.4.2.10 RBSP slice trailing bits semantics ... 78
7.4.2.11 RBSP trailing bits semantics ... 78
7.4.3 Slice header semantics ... 78
7.4.3.1 Reference picture list reordering semantics ... 84
7.4.3.2 Prediction weight table semantics ... 85
7.4.3.3 Decoded reference picture marking semantics ... 85
7.4.4 Slice data semantics ... 88
7.4.5 Macroblock layer semantics ... 89
7.4.5.1 Macroblock prediction semantics ... 96
7.4.5.2 Sub-macroblock prediction semantics ... 97
7.4.5.3 Residual data semantics ... 99

7.4.5.3.1 Residual block CAVLC semantics 100
7.4.5.3.2 Residual block CABAC semantics 100
8 Decoding process 101
8.1 NAL unit decoding process 102
8.2 Slice decoding process 102
8.2.1 Decoding process for picture order count 102
8.2.1.1 Decoding process for picture order count type 0 104
8.2.1.2 Decoding process for picture order count type 1 105
8.2.1.3 Decoding process for picture order count type 2 106
8.2.2 Decoding process for macroblock to slice group map 107
8.2.2.1 Specification for interleaved slice group map type 108
8.2.2.2 Specification for dispersed slice group map type 108
8.2.2.3 Specification for foreground with left-over slice group map type 108
8.2.2.4 Specification for box-out slice group map types 109
8.2.2.5 Specification for raster scan slice group map types 109
8.2.2.6 Specification for wipe slice group map types 110
8.2.2.7 Specification for explicit slice group map type 110
8.2.2.8 Specification for conversion of map unit to slice group map to macroblock to slice group map 110
8.2.3 Decoding process for slice data partitioning 110
8.2.4 Decoding process for reference picture lists construction 111
8.2.4.1 Decoding process for picture numbers 111
8.2.4.2 Initialisation process for reference picture lists 112
8.2.4.2.1 Initialisation process for the reference picture list for P and SP slices in frames 113
8.2.4.2.2 Initialisation process for the reference picture list for P and SP slices in fields 113
8.2.4.2.3 Initialisation process for reference picture lists for B slices in frames 114
8.2.4.2.4 Initialisation process for reference picture lists for B slices in fields 114
8.2.4.2.5 Initialisation process for reference picture lists in fields 115
8.2.4.3 Reordering process for reference picture lists 116
8.2.4.3.1 Reordering process of reference picture lists for short-term reference pictures 116
8.2.4.3.2 Reordering process of reference picture lists for long-term reference pictures 117
8.2.5 Decoded reference picture marking process 118
8.2.5.1 Sequence of operations for decoded reference picture marking process 118
8.2.5.2 Decoding process for gaps in frame_num 119
8.2.5.3 Sliding window decoded reference picture marking process 119
8.2.5.4 Adaptive memory control decoded reference picture marking process 120
8.2.5.4.1 Marking process of a short-term reference picture as “unused for reference” 120
8.2.5.4.2 Marking process of a long-term reference picture as “unused for reference” 120
8.2.5.4.3 Assignment process of a LongTermFrameIdx to a short-term reference picture 120
8.2.5.4.4 Decoding process for MaxLongTermFrameIdx 121
8.2.5.4.4.1 Marking process of all reference pictures as “unused for reference” and setting MaxLongTermFrameIdx to “no long-term frame indices” 121
8.2.5.4.5 Process for assigning a long-term frame index to the current picture 121
8.3 Intra prediction process 122
8.3.1 Intra_4x4 prediction process for luma samples 122
8.3.1.1 Derivation process for the Intra4x4PredMode 123
8.3.1.1.1 Derivation process for Intra_4x4 prediction modes 124
8.3.1.2 Intra_4x4 sample prediction 125
8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode 126
8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode 126
8.3.1.2.3 Specification of Intra_4x4_DC prediction mode 126
8.3.1.2.4 Specification of Intra_4x4_Diagonal_Down_Left prediction mode 127
8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode 127
8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode 127
8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode 128
8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode 128
8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode 129
8.3.2 Intra_8x8 prediction process for luma samples 129
8.3.2.1 Derivation process for Intra8x8PredMode 130
8.3.2.1.1 Derivation process for Intra_8x8 prediction modes 130
8.3.2.2 Intra_8x8 sample prediction 132
8.3.2.2.1 Reference sample filtering process for Intra_8x8 sample prediction 133
8.3.2.2.2 Specification of Intra_8x8_Vertical prediction mode 134
8.3.2.2.3 Specification of Intra_8x8_Horizontal prediction mode 134

8.3.2.2.4 Specification of Intra_8x8_DC prediction mode 134
8.3.2.2.5 Specification of Intra_8x8_Diagonal_Down_Left prediction mode 135
8.3.2.2.6 Specification of Intra_8x8_Diagonal_Down_Right prediction mode 135
8.3.2.2.7 Specification of Intra_8x8_Vertical_Right prediction mode 135
8.3.2.2.8 Specification of Intra_8x8_Horizontal_Down prediction mode 136
8.3.2.2.9 Specification of Intra_8x8_Vertical_Left prediction mode 136
8.3.2.2.10 Specification of Intra_8x8_Horizontal_Up prediction mode 137
8.3.3 Intra_16x16 prediction process for luma samples 137
8.3.3.1 Specification of Intra_16x16_Vertical prediction mode 138
8.3.3.2 Specification of Intra_16x16_Horizontal prediction mode 138
8.3.3.3 Specification of Intra_16x16_DC prediction mode 138
8.3.3.4 Specification of Intra_16x16_Plane prediction mode 139
8.3.4 Intra prediction process for chroma samples 139
8.3.4.1 Specification of Intra_Chroma_DC prediction mode 141
8.3.4.2 Specification of Intra_Chroma_Horizontal prediction mode 142
8.3.4.3 Specification of Intra_Chroma_Vertical prediction mode 142
8.3.4.4 Specification of Intra_Chroma_Plane prediction mode 143
8.3.5 Sample construction process for I_PCM macroblocks 143
8.4 Inter prediction process 144
8.4.1 Derivation process for motion vector components and reference indices 146
8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices 147
8.4.1.2 Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 148
8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions 149
8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode . 152
8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode 153
8.4.1.3 Derivation process for luma motion vector prediction 156
8.4.1.3.1 Derivation process for median luma motion vector prediction 157
8.4.1.3.2 Derivation process for motion data of neighbouring partitions 158
8.4.1.4 Derivation process for chroma motion vectors 159
8.4.2 Decoding process for Inter prediction samples 159
8.4.2.1 Reference picture selection process 160
8.4.2.2 Fractional sample interpolation process 161
8.4.2.2.1 Luma sample interpolation process 162
8.4.2.2.2 Chroma sample interpolation process 165
8.4.2.3 Weighted sample prediction process 166
8.4.2.3.1 Default weighted sample prediction process 167
8.4.2.3.2 Weighted sample prediction process 167
8.4.3 Derivation process for prediction weights 168
8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process . 170
8.5.1 Specification of transform decoding process for 4x4 luma residual blocks 171
8.5.2 Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode 171
8.5.3 Specification of transform decoding process for 8x8 luma residual blocks 172
8.5.4 Specification of transform decoding process for chroma samples 173
8.5.5 Inverse scanning process for transform coefficients 175
8.5.6 Inverse scanning process for 8x8 luma transform coefficients 175
8.5.7 Derivation process for the chroma quantisation parameters and scaling function 177
8.5.8 Scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type 179
8.5.8.1 Transformation process for luma DC transform coefficients for Intra_16x16 macroblock type 180
8.5.8.2 Scaling process for luma DC transform coefficients for Intra_16x16 macroblock type 180
8.5.9 Scaling and transformation process for chroma DC transform coefficients 181
8.5.9.1 Transformation process for chroma DC transform coefficients 181
8.5.9.2 Scaling process for chroma DC transform coefficients 182
8.5.10 Scaling and transformation process for residual 4x4 blocks 183
8.5.10.1 Scaling process for residual 4x4 blocks 183
8.5.10.2 Transformation process for residual 4x4 blocks 184
8.5.11 Scaling and transformation process for residual 8x8 luma blocks 185
8.5.11.1 Scaling process for residual 8x8 luma blocks 186
8.5.11.2 Transformation process for residual 8x8 luma blocks 186
8.5.12 Picture construction process prior to deblocking filter process 189
8.5.13 Residual colour transform process 190

8.6 Decoding process for P macroblocks in SP slices or SI macroblocks ... 190
8.6.1 SP decoding process for non-switching pictures ... 191
8.6.1.1 Luma transform coefficient decoding process ... 191
8.6.1.2 Chroma transform coefficient decoding process ... 192
8.6.2 SP and SI slice decoding process for switching pictures ... 194
8.6.2.1 Luma transform coefficient decoding process ... 194
8.6.2.2 Chroma transform coefficient decoding process ... 194
8.7 Deblocking filter process ... 195
8.7.1 Filtering process for block edges ... 199
8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge ... 201
8.7.2.1 Derivation process for the luma content dependent boundary filtering strength ... 201
8.7.2.2 Derivation process for the thresholds for each block edge ... 203
8.7.2.3 Filtering process for edges with bS less than 4 ... 204
8.7.2.4 Filtering process for edges for bS equal to 4 ... 206
9 Parsing process ... 207
9.1 Parsing process for Exp-Golomb codes ... 207
9.1.1 Mapping process for signed Exp-Golomb codes ... 209
9.1.2 Mapping process for coded block pattern ... 209
9.2 CAVLC parsing process for transform coefficient levels ... 212
9.2.1 Parsing process for total number of transform coefficient levels and trailing ones ... 213
9.2.2 Parsing process for level information ... 216
9.2.2.1 Parsing process for level_prefix ... 217
9.2.3 Parsing process for run information ... 218
9.2.4 Combining level and run information ... 221
9.3 CABAC parsing process for slice data ... 221
9.3.1 Initialisation process ... 222
9.3.1.1 Initialisation process for context variables ... 223
9.3.1.2 Initialisation process for the arithmetic decoding engine ... 233
9.3.2 Binarization process ... 234
9.3.2.1 Unary (U) binarization process ... 236
9.3.2.2 Truncated unary (TU) binarization process ... 236
9.3.2.3 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process ... 237
9.3.2.4 Fixed-length (FL) binarization process ... 237
9.3.2.5 Binarization process for macroblock type and sub-macroblock type ... 238
9.3.2.6 Binarization process for coded block pattern ... 241
9.3.2.7 Binarization process for mb_qp_delta ... 241
9.3.3 Decoding process flow ... 241
9.3.3.1 Derivation process for ctxIdx ... 242
9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements ... 244
9.3.3.1.1.1 Derivation process of ctxIdxInc for the syntax element mb_skip_flag ... 244
9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag ... 244
9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax element mb_type ... 245
9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element coded_block_pattern ... 245
9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta ... 246
9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1 ... 246
9.3.3.1.1.7 Derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1 ... 247
9.3.3.1.1.8 Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode ... 249
9.3.3.1.1.9 Derivation process of ctxIdxInc for the syntax element coded_block_flag ... 249
9.3.3.1.1.10 Derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag ... 250
9.3.3.1.2 Assignment process of ctxIdxInc using prior decoded bin values ... 251
9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 ... 251
9.3.3.2 Arithmetic decoding process ... 254
9.3.3.2.1 Arithmetic decoding process for a binary decision ... 255
9.3.3.2.1.1 State transition process ... 255
9.3.3.2.2 Renormalization process in the arithmetic decoding engine ... 258
9.3.3.2.3 Bypass decoding process for binary decisions ... 259
9.3.3.2.4 Decoding process for binary decisions before termination ... 259
9.3.4 Arithmetic encoding process (informative) ... 260
9.3.4.1 Initialisation process for the arithmetic encoding engine (informative) ... 260
9.3.4.2 Encoding process for a binary decision (informative) ... 260
9.3.4.3 Renormalization process in the arithmetic encoding engine (informative) ... 261
9.3.4.4 Bypass encoding process for binary decisions (informative) ... 263

9.3.4.5 Encoding process for a binary decision before termination (informative) ... 264
9.3.4.6 Byte stuffing process (informative) ... 265

Annex A Profiles and levels ... 267
A.1 Requirements on video decoder capability ... 267
A.2 Profiles ... 267
A.2.1 Baseline profile ... 267
A.2.2 Main profile ... 268
A.2.3 Extended profile ... 268
A.2.4 High profile ... 268
A.2.5 High 10 profile ... 269
A.2.6 High 4:2:2 profile ... 269
A.2.7 High 4:4:4 profile ... 270
A.3 Levels ... 270
A.1.1 Level limits common to the Baseline, Main, and Extended profiles ... 270
A.3.1 Level limits common to the High, High 10, High 4:2:2, and High 4:4:4 profiles ... 272
A.1.2 Profile-specific level limits ... 273
A.3.1.1 Baseline profile limits ... 274
A.3.1.2 Main, High, High 10, High 4:2:2, or High 4:4:4 profile limits ... 275
A.3.1.3 Extended Profile Limits ... 276
A.3.2 Effect of level limits on frame rate (informative) ... 278

Annex B Byte stream format ... 281
B.1 Byte stream NAL unit syntax and semantics ... 281
B.1.1 Byte stream NAL unit syntax ... 281
B.1.2 Byte stream NAL unit semantics ... 281
B.2 Byte stream NAL unit decoding process ... 282
B.3 Decoder byte-alignment recovery (informative) ... 282

Annex C Hypothetical reference decoder ... 283
C.1 Operation of coded picture buffer (CPB) ... 286
C.1.1 Timing of bitstream arrival ... 286
C.1.2 Timing of coded picture removal ... 287
C.2 Operation of the decoded picture buffer (DPB) ... 288
C.2.1 Decoding of gaps in frame_num and storage of "non-existing" frames ... 288
C.2.2 Picture decoding and output ... 288
C.2.3 Removal of pictures from the DPB before possible insertion of the current picture ... 288
C.2.4 Current decoded picture marking and storage ... 289
C.2.4.1 Marking and storage of a reference decoded picture into the DPB ... 289
C.2.4.2 Storage of a non-reference picture into the DPB ... 289
C.3 Bitstream conformance ... 289
C.4 Decoder conformance ... 291
C.4.1 Operation of the output order DPB ... 292
C.4.2 Decoding of gaps in frame_num and storage of "non-existing" pictures ... 292
C.4.3 Picture decoding ... 292
C.4.4 Removal of pictures from the DPB before possible insertion of the current picture ... 292
C.4.5 Current decoded picture marking and storage ... 292
C.4.5.1 Storage and marking of a reference decoded picture into the DPB ... 292
C.4.5.2 Storage and marking of a non-reference decoded picture into the DPB ... 293
C.4.5.3 "Bumping" process ... 293

Annex D Supplemental enhancement information ... 295
D.1 SEI payload syntax ... 296
D.1.1 Buffering period SEI message syntax ... 297
D.1.2 Picture timing SEI message syntax ... 298
D.1.3 Pan-scan rectangle SEI message syntax ... 299
D.1.4 Filler payload SEI message syntax ... 299
D.1.5 User data registered by ITU-T Rec. T.35 SEI message syntax ... 299
D.1.6 User data unregistered SEI message syntax ... 300
D.1.7 Recovery point SEI message syntax ... 300
D.1.8 Decoded reference picture marking repetition SEI message syntax ... 300
D.1.9 Spare picture SEI message syntax ... 301
D.1.10 Scene information SEI message syntax ... 301
D.1.11 Sub-sequence information SEI message syntax ... 302
D.1.12 Sub-sequence layer characteristics SEI message syntax ... 302
D.1.13 Sub-sequence characteristics SEI message syntax ... 303

D.1.14 Full-frame freeze SEI message syntax ... 303
D.1.15 Full-frame freeze release SEI message syntax ... 303
D.1.16 Full-frame snapshot SEI message syntax ... 303
D.1.17 Progressive refinement segment start SEI message syntax ... 304
D.1.18 Progressive refinement segment end SEI message syntax ... 304
D.1.19 Motion-constrained slice group set SEI message syntax ... 304
D.1.20 Film grain characteristics SEI message syntax ... 305
D.1.21 Deblocking filter display preference SEI message syntax ... 305
D.1.22 Stereo video information SEI message syntax ... 306
D.1.23 Reserved SEI message syntax ... 306
D.2 SEI payload semantics ... 306
D.2.1 Buffering period SEI message semantics ... 306
D.2.2 Picture timing SEI message semantics ... 307
D.2.3 Pan-scan rectangle SEI message semantics ... 310
D.2.4 Filler payload SEI message semantics ... 312
D.2.5 User data registered by ITU-T Rec. T.35 SEI message semantics ... 312
D.2.6 User data unregistered SEI message semantics ... 312
D.2.7 Recovery point SEI message semantics ... 312
D.2.8 Decoded reference picture marking repetition SEI message semantics ... 314
D.2.9 Spare picture SEI message semantics ... 314
D.2.10 Scene information SEI message semantics ... 316
D.2.11 Sub-sequence information SEI message semantics ... 317
D.2.12 Sub-sequence layer characteristics SEI message semantics ... 319
D.2.13 Sub-sequence characteristics SEI message semantics ... 320
D.2.14 Full-frame freeze SEI message semantics ... 321
D.2.15 Full-frame freeze release SEI message semantics ... 322
D.2.16 Full-frame snapshot SEI message semantics ... 322
D.2.17 Progressive refinement segment start SEI message semantics ... 322
D.2.18 Progressive refinement segment end SEI message semantics ... 323
D.2.19 Motion-constrained slice group set SEI message semantics ... 323
D.2.20 Film grain characteristics SEI message semantics ... 323
D.2.21 Deblocking filter display preference SEI message semantics ... 329
D.2.22 Stereo video information SEI message semantics ... 331
D.2.23 Reserved SEI message semantics ... 332
Annex E Video usability information ... 333
E.1 VUI syntax ... 334
E.1.1 VUI parameters syntax ... 334
E.1.2 HRD parameters syntax ... 335
E.2 VUI semantics ... 335
E.2.1 VUI parameters semantics ... 335
E.2.2 HRD parameters semantics ... 346
Annex G Scalable video coding ... 349
G.1 Scope ... 349
G.2 Normative References ... 350
G.3 Definitions ... 350
G.4 Abbreviations ... 350
G.5 Conventions ... 350
G.6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships ... 350
G.6.1 Derivation process for base quality slices ... 350
G.6.2 Derivation process for base macroblock ... 351
G.6.3 Derivation process for base partitions ... 353
G.6.4 Derivation process for macroblock type and sub-macroblock type in inter-layer prediction ... 354
G.6.5 Virtual base layer macroblock conversion ... 358
G.6.5.1 Field-to-Frame base macroblock conversion ... 359
G.6.5.2 Frame-to-Field base macroblock conversion ... 359
G.7 Syntax and semantics ... 360
G.7.1 Method of specifying syntax in tabular form ... 360
G.7.2 Specification of syntax functions, categories, and descriptors ... 360
G.7.3 Syntax in tabular form ... 367
G.7.3.1 NAL unit header SVC extension syntax ... 367
G.7.3.2 Sequence parameter set SVC extension syntax ... 367
G.7.3.3 Slice layer in scalable extension syntax ... 369

G.7.3.4 Slice header in scalable extension ... 369
G.7.3.4.1 Decoded reference picture marking syntax for decoded base pictures ... 372
G.7.3.5 Slice data in scalable extension syntax ... 372
G.7.3.6 Macroblock layer in scalable extension syntax ... 373
G.7.3.6.1 Macroblock prediction in scalable extension syntax ... 376
G.7.3.6.2 Sub-macroblock prediction in scalable extension syntax ... 377
G.7.3.6.3 Residual in scalable extension syntax ... 378
G.7.3.7 Progressive refinement slice data in scalable extension syntax ... 380
G.7.3.7.1 Motion data in progressive refinement slice data syntax ... 385
G.7.3.7.1.1 Macroblock prediction in progressive refinement slice data syntax ... 386
G.7.3.7.1.2 Sub-macroblock in progressive refinement slice data syntax ... 387
G.7.3.7.2 Macroblock Luma Coded Block Pattern in progressive slice data CAVLC syntax ... 387
G.7.3.7.3 Macroblock Chroma Coded Block Pattern in progressive slice data CAVLC syntax ... 388
G.7.3.7.4 Luma coefficient in progressive refinement slice data syntax ... 389
G.7.3.7.5 Chroma DC coefficient in progressive slice data syntax ... 393
G.7.3.7.6 Chroma AC coefficient in progressive refinement slice data syntax ... 396
G.7.3.7.7 Significant coefficient in progressive slice data CABAC syntax ... 399
G.7.3.7.8 Significant coefficient in progressive slice data CAVLC syntax ... 399
G.7.3.7.9 Luma coefficient refinement in progressive refinement slice data syntax ... 400
G.7.3.7.10 Chroma DC coefficient refinement in progressive refinement slice data syntax ... 400
G.7.3.7.11 Chroma AC coefficient refinement in progressive refinement slice data syntax ... 401
G.7.3.7.12 Coefficient refinement in progressive refinement slice data syntax ... 401
G.7.4 Semantics ... 401
G.7.4.1 NAL unit header SVC extension semantics ... 401
G.7.4.1.1 Order of VCL NAL units and association to coded pictures in the scalable extension ... 403
G.7.4.1.2 Detection of first VCL NAL units of a primary coded picture / access unit ... 403
G.7.4.2 Sequence parameter set SVC extension semantics ... 404
G.7.4.3 Slice layer in scalable extension semantics ... 405
G.7.4.4 Slice header in scalable extension semantics ... 406
G.7.4.4.1 Semantics of decoded reference picture marking syntax for decoded base pictures ... 410
G.7.4.5 Slice data in scalable extension semantics ... 411
G.7.4.6 Macroblock layer in scalable extension semantics ... 411
G.7.4.6.1 Macroblock prediction in scalable extension semantics ... 413
G.7.4.6.2 Sub-macroblock prediction in scalable extension semantics ... 413
G.7.4.6.3 Residual in scalable extension semantics ... 413
G.7.4.7 Progressive refinement slice data in scalable extension semantics ... 413
G.7.4.7.1 Motion data in progressive refinement slice data semantics ... 415
G.7.4.7.1.1 Macroblock prediction in progressive refinement slice data semantics ... 415
G.7.4.7.1.2 Sub-macroblock prediction in progressive refinement slice data semantics ... 415
G.7.4.7.2 Macroblock Luma Coded Block Pattern in progressive refinement slice data CAVLC semantics 415
G.7.4.7.3 Macroblock Chroma Coded Block Pattern in progressive refinement slice data CAVLC semantics 415
G.7.4.7.4 Luma coefficient in progressive refinement slice data semantics ... 415
G.7.4.7.5 Chroma DC coefficient in progressive slice data semantics ... 416
G.7.4.7.6 Chroma AC coefficient in progressive refinement slice data semantics ... 417
G.7.4.7.7 Significant coefficient in progressive slice data CABAC semantics ... 417
G.7.4.7.8 Significant coefficient in progressive slice data CAVLC semantics ... 417
G.7.4.7.9 Coefficient refinement in progressive refinement slice data semantics ... 417
G.8 Decoding process ... 417
G.8.1 Array assignment initialization process ... 420
G.8.2 Base slice decoding process without resolution change ... 421
G.8.3 Base slice decoding process with resolution change ... 424
G.8.3.1 Base slice decoding process prior to resampling ... 425
G.8.4 Target slice decoding process ... 426
G.8.5 Motion data derivation process ... 427
G.8.5.1 Derivation process for luma vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 ... 429
G.8.5.1.1 Derivation process for spatial direct luma motion vector and reference index prediction mode in scalable extension ... 430
G.8.6 Resampling process for motion data ... 430
G.8.7 Inter prediction process ... 434
G.8.7.1 Decoding process for differential Inter prediction samples ... 437
G.8.7.1.1 Differential reference picture selection process ... 437

G.8.7.1.2 Fractional differential sample interpolation process ... 438
G.8.7.1.3 Fractional differential luma sample interpolation process ... 440
G.8.7.1.4 Fractional differential chroma sample interpolation process ... 440
G.8.7.2 Scaling process for differential Inter prediction samples ... 441
G.8.7.2.1 Scaling process for differential Inter prediction samples of 4x4 luma blocks ... 442
G.8.7.2.1.1 Forward 4-point one dimensional transform process ... 443
G.8.7.2.1.2 Forward 4x4 transform process on differential Inter prediction samples ... 443
G.8.7.2.1.3 Normalization process for 4x4 blocks of transform coefficient values ... 443
G.8.7.2.2 Scaling process for differential Inter prediction samples of 8x8 luma blocks ... 444
G.8.7.2.2.1 Forward 8-point one dimensional transform process ... 445
G.8.7.2.2.2 Forward 8x8 transform process on differential Inter luma prediction samples ... 445
G.8.7.2.2.3 Normalization process for 8x8 blocks of transform coefficient values ... 445
G.8.7.2.3 Scaling process for differential Inter prediction samples of chroma blocks ... 446
G.8.7.2.3.1 Assignment process for a matrix of chroma DC transform coefficient values ... 448
G.8.7.2.3.2 Construction process for chroma DC coefficient values ... 448
G.8.7.3 Decoding process for smoothed reference inter prediction samples ... 449
G.8.7.3.1 Smoothing process for smoothed reference inter prediction samples ... 449
G.8.7.4 Derivation process for prediction weights ... 450
G.8.8 Intra macroblock decoding process ... 452
G.8.8.1 Intra_4x4 prediction and construction process for luma samples ... 453
G.8.8.2 Intra_8x8 prediction and construction process for luma samples ... 454
G.8.8.3 Intra_16x16 prediction and construction process for luma samples ... 455
G.8.8.4 Intra prediction and construction process for chroma samples ... 455
G.8.9 Resampling process for intra samples ... 456
G.8.9.1 Deblocking filter process for Intra_Base prediction ... 464
G.8.9.2 Construction process for not available samples ... 467
G.8.9.2.1 Constant border extension process ... 468
G.8.9.2.2 Diagonal border extension process ... 468
G.8.9.3 Upsampling process for Intra_Base prediction ... 469
G.8.9.4 Upsampling process for Intra_Base prediction of interlaced pictures ... 471
G.8.9.5 Vertical upsampling process for Intra_Base prediction ... 475
G.8.10 Resampling process for residual data ... 475
G.8.10.1 Upsampling process for residual prediction ... 478
G.8.10.2 Upsampling process for residual prediction in interlaced pictures ... 480
G.8.10.3 Bilinear interpolation process for residual prediction ... 485
G.8.10.4 Vertical upsampling process for residual prediction ... 487
G.8.11 Transformation, scaling, and picture construction processes ... 487
G.8.11.1 Initialization process for macroblock, sub-macroblock, and luma transform type ... 488
G.8.11.2 Transform coefficient scaling process ... 488
G.8.11.3 Progressive refinement process for scaled transform coefficients ... 489
G.8.11.3.1 Progressive refinement process for scaled transform coefficient of 4x4 luma blocks ... 490
G.8.11.3.2 Progressive refinement process for scaled transform coefficient of 8x8 luma blocks ... 490
G.8.11.3.3 Progressive refinement process for scaled luma transform coefficients of Intra_16x16 macroblocks 490
G.8.11.3.4 Progressive refinement process for scaled chroma transform coefficients ... 491
G.8.11.4 Residual accumulation process ... 492
G.8.11.5 Residual construction process ... 493
G.8.11.5.1 Macroblock residual construction process ... 494
G.8.11.5.2 Residual construction process for 4x4 luma blocks ... 494
G.8.11.5.3 Residual construction process for 8x8 luma blocks ... 495
G.8.11.5.4 Residual construction process for the luma component of Intra_16x16 macroblocks ... 495
G.8.11.5.5 Residual construction process for the chroma component ... 496
G.8.11.6 Picture construction process prior to deblocking filter process ... 497
G.8.11.7 Sample construction process ... 497
G.8.11.7.1 Picture sample array construction process ... 498
G.8.11.7.2 Macroblock sample array extraction process ... 498
G.8.11.7.3 Picture sample array construction process for a signal component ... 499
G.8.11.7.4 Macroblock sample array extraction process for a signal component ... 499
G.8.12 Decoding process for reference picture lists construction ... 500
G.8.12.1 Decoding process for picture numbers ... 501
G.8.13 Decoded reference picture marking process ... 503
G.8.14 Deblocking filter process in scalable extension ... 506
G.8.14.1 Extended derivation process for the luma content dependent boundary filtering strength ... 506

G.9 Parsing process *507*
G.9.1 CAVLC parsing process for progressive refinement slices 508
G.9.1.1 Initialisation process 508
G.9.1.2 Bit reading process for coded_block_flag_luma and coded_block_flag_chromaAC 509
G.9.1.3 Derivation process for coded block flag syntax elements 511
G.9.1.4 Derivation process for the syntax element coded_block_flag_luma 511
G.9.1.5 Derivation process for the syntax element coded_block_flag_chromaAC 511
G.9.1.6 Derivation process for the syntax element coeff_sig_vlc_symbol 512
G.9.1.7 Derivation process for the syntax element coeff_refinement_ flag and coeff_refinement_direction_flag 512
G.9.1.7.1 Derivation process for the syntax element coeff_refinement_ flag 513
G.9.1.7.2 Derivation process for the syntax element coeff_refinement_direction_flag 514
G.9.1.7.3 Updating process for parsing coeff_refinement_ flag and coeff_refinement_direction_flag 514
G.9.2 Alternative CABAC parsing process for slice data in scalable extension 515
G.9.2.1 Initialisation process 515
G.9.2.2 Binarization process 516
G.9.2.2.1 Signed truncated unary (STU) binarization process 517
G.9.2.3 Decoding process flow 517
G.9.2.3.1 Derivation process for ctxIdx 517
G.9.2.3.2 Assignment process of ctxIdxInc using neighbouring syntax elements 518
G.9.2.3.2.1 Derivation process of ctxIdxInc for the syntax element coded_block_pattern_bit 518
G.9.2.3.2.2 Derivation process of ctxIdxInc for the syntax element base_mode_flag 518
G.9.2.3.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 518
G.9.2.3.4 Derivation process of ctxIdxInc for the syntax element residual_prediction_flag 520
G.10 Supplemental enhancement information *521*
G.10.1 SEI payload syntax 521
G.10.1.1 Scalability information SEI message syntax 521
G.10.1.2 Scalability information layers not present SEI message syntax 523
G.10.1.3 Scalability information dependency change SEI message syntax 523
G.10.1.4 Sub-picture scalable layer SEI message syntax 523
G.10.1.5 Non-required picture SEI message syntax 524
G.10.1.6 Quality layer information SEI message syntax 524
G.10.1.7 Scalable nesting SEI message syntax 524
G.10.2 SEI payload semantics 525
G.10.2.1 Scalability information SEI message semantics 525
G.10.2.2 Scalability information layers not present SEI message semantics 528
G.10.2.3 Scalability information dependency change SEI message semantics 529
G.10.2.4 Sub-picture scalable layer SEI message semantics 529
G.10.2.5 Non-required picture SEI message semantics 529
G.10.2.6 Quality layer information SEI message semantics 530
G.10.2.7 Scalable nesting SEI message semantics 530

LIST OF FIGURES

Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame 20

Figure 6-2 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields 20

Figure 6-3 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a frame 21

Figure 6-4 – Nominal vertical and horizontal sampling locations of 4:2:2 samples top and bottom fields 21

Figure 6-5 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a frame 22

Figure 6-6 – Nominal vertical and horizontal sampling locations of 4:4:4 samples top and bottom fields 22

Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices 23

Figure 6-8 – Partitioning of the decoded frame into macroblock pairs 24

Figure 6-9 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans 25

Figure 6-10 – Scan for 4x4 luma blocks 26

Figure 6-11 – Scan for 8x8 luma blocks ... 27

Figure 6-12 – Neighbouring macroblocks for a given macroblock ... 28

Figure 6-13 – Neighbouring macroblocks for a given macroblock in MBAFF frames ... 29

Figure 6-14 – Determination of the neighbouring macroblock, blocks, and partitions (informative) ... 30

Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 22 to 31, inclusive ... 64

Figure 8-1 – Intra_4x4 prediction mode directions (informative) ... 123

Figure 8-2 – Example for temporal direct-mode motion vector inference (informative) ... 156

Figure 8-3 – Directional segmentation prediction (informative) ... 157

Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation ... 163

Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D ... 165

Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx ... 172

Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx: (a) chroma_format_idc equal to 1, (b) chroma_format_idc equal to 2, (c) chroma_format_idc equal to 3 ... 174

Figure 8-8 – 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative) ... 175

Figure 8-9 – 8x8 block scans. (a) 8x8 zig-zag scan. (b) 8x8 field scan (informative) ... 176

Figure 8-10 – Boundaries in a macroblock to be filtered ... 196

Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary ... 200

Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative) ... 222

Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative) ... 254

Figure 9-3 – Flowchart for decoding a decision ... 256

Figure 9-4 – Flowchart of renormalization ... 259

Figure 9-5 – Flowchart of bypass decoding process ... 259

Figure 9-6 – Flowchart of decoding a decision before termination ... 260

Figure 9-7 – Flowchart for encoding a decision ... 261

Figure 9-8 – Flowchart of renormalization in the encoder ... 262

Figure 9-9 – Flowchart of PutBit(B) ... 263

Figure 9-10 – Flowchart of encoding bypass ... 264

Figure 9-11 – Flowchart of encoding a decision before termination ... 265

Figure 9-12 – Flowchart of flushing at termination ... 265

Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks ... 284

Figure C-2 – HRD buffer model ... 285

Figure E-1 – Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field ... 343

LIST OF TABLES

Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc ... 19

Table 6-2 – Specification of input and output assignments for subclauses 6.4.8.1 to 6.4.8.5 ... 29

Table 6-3 – Specification of mbAddrN ... 34

Table 6-4 – Specification of mbAddrN and yM 36
Table 7-1 – NAL unit type codes 59
Table 7-2 – Assignment of mnemonic names to scaling list indices and specification of fall-back rule 68
Table 7-3 – Specification of default scaling lists Default_4x4_Intra and Default_4x4_Inter 68
Table 7-4 – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter 69
Table 7-5 – Meaning of primary_pic_type 76
Table 7-6 – Name association to slice_type 78
Table 7-7 – reordering_of_pic_nums_idc operations for reordering of reference picture lists 84
Table 7-8 – Interpretation of adaptive_ref_pic_marking_mode_flag 86
Table 7-9 – Memory management control operation (memory_management_control_operation) values 87
Table 7-10 – Allowed collective macroblock types for slice_type 89
Table 7-11 – Macroblock types for I slices 91
Table 7-12 – Macroblock type with value 0 for SI slices 92
Table 7-13 – Macroblock type values 0 to 4 for P and SP slices 93
Table 7-14 – Macroblock type values 0 to 22 for B slices 94
Table 7-15 – Specification of CodedBlockPatternChroma values 96
Table 7-16 – Relationship between intra_chroma_pred_mode and spatial prediction modes 96
Table 7-17 – Sub-macroblock types in P macroblocks 97
Table 7-18 – Sub-macroblock types in B macroblocks 98
Table 8-1 – Refined slice group map type 107
Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names 123
Table 8-3 – Specification of Intra8x8PredMode[luma8x8BlkIdx] and associated names 130
Table 8-4 – Specification of Intra16x16PredMode and associated names 138
Table 8-5 – Specification of Intra chroma prediction modes and associated names 140
Table 8-6 – Specification of the variable colPic 149
Table 8-7 – Specification of PicCodingStruct(X) 149
Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale 151
Table 8-9 – Assignment of prediction utilization flags 153
Table 8-10 – Derivation of the vertical component of the chroma vector in field coding mode 159
Table 8-11 – Differential full-sample luma locations 164
Table 8-12 – Assignment of the luma prediction sample $predPartLX_L[\ x_L, y_L\ ]$ 165
Table 8-13 – Specification of mapping of idx to c_{ij} for zig-zag and field scan 175
Table 8-14 – Specification of mapping of idx to c_{ij} for 8x8 zig-zag and 8x8 field scan 177
Table 8-15 – Specification of QP_C as a function of qP_I 178
Table 8-16 – Derivation of offset dependent threshold variables α' and β' from indexA and indexB 204
Table 8-17 – Value of variable t'_{C0} as a function of indexA and bS 206
Table 9-1 – Bit strings with “prefix” and “suffix” bits and assignment to codeNum ranges (informative) 208
Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative) 208
Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v) 209
Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes 209

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token) 214
Table 9-6 – Codeword table for level_prefix (informative) 218
Table 9-7 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7 219
Table 9-8 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 8 to 15 220
Table 9-9 – total_zeros tables for chroma DC 2x2 and 2x4 blocks 220
Table 9-10 – Tables for run_before 221
Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process 223
Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10 224
Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23 225
Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39 225
Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53 225
Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59, and 399 to 401 226
Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69 226
Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104 227
Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165 228
Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226 229
Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275 230
Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337 231
Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398 232
Table 9-24 – Values of variables m and n for ctxIdx from 402 to 459 233
Table 9-25 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset 235
Table 9-26 – Bin string of the unary binarization (informative) 236
Table 9-27 – Binarization for macroblock types in I slices 238
Table 9-28 – Binarization for macroblock types in P, SP, and B slices 240
Table 9-29 – Binarization for sub-macroblock types in P, SP, and B slices 241
Table 9-30 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 243
Table 9-31 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 244
Table 9-32 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx 251
Table 9-33 – Specification of ctxBlockCat for the different blocks 252
Table 9-34 – Mapping of scanning position to ctxIdxInc for ctxBlockCat = = 5 252
Table 9-35 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx 257
Table 9-36 – State transition table 258
Table A-1 – Level limits 272
Table A-2 – Specification of cpbBrVclFactor and cpbBrNalFactor 274
Table A-3 – Baseline profile level limits 275
Table A-4 – Main, High, High 10, High 4:2:2, or High 4:4:4 profile level limits 276
Table A-5 – Extended profile level limits 277
Table A-6 – Maximum frame rates (frames per second) for some example frame sizes 278

Table D-1 – Interpretation of pic_struct 308

Table D-2 – Mapping of ct_type to source picture scan 309

Table D-3 – Definition of counting_type values 309

Table D-4 – scene_transition_type values 316

Table D-5 – model_id values 324

Table D-6 – blending_mode_id values 325

Table E-1 – Meaning of sample aspect ratio indicator 336

Table E-2 – Meaning of video_format 337

Table E-3 – Colour primaries 338

Table E-4 – Transfer characteristics 339

Table E-5 – Matrix coefficients 342

Table E-6 – Divisor for computation of $\Delta t_{fi,dpb}(n)$ 344

Table G-1 – Derivation of mackroblock types. 357

Table G-2 – Derivation of sub-macroblock types. 357

Table G-3 – Name association to slice_type for NAL units with nal_unit_type equal to 20 or 21. 406

Table G-4 – Memory management control operation (memory_management_control_operation) values 411

Table G-5 – Allowed collective macroblock types for slice_type. 412

Table G-6 – Macroblock types for EI slices. 412

Table G-7 – Codeword table for sig_vlc_selector 414

Table G-8 – 16-phase interpolation filter for luma up-sampling in Intra_Base prediction 463

Table G-9 - Initialisation values for CAVLC internal variables for coded_block_flag_luma and coded_block_flag_chromaAC 508

Table G-10 - Initialisation values for CAVLC internal variables for coeff_ref_vlc_symbol 509

Table G-11 – Variable length codes used in parsing progressive refinement syntax elements 510

Table G-12 - Derivation of variable allowedVlcIdx[ ] 510

Table G-13 - Update table for variable vlcTable for coded_block_flag_luma and coded_block_flag_chromaAC 510

Table G-14 – Codebook table for coeff_sig_vlc_symbol depending on sigVlcSelector (informative) 512

Table G-15 – Codeword tables for coeff_ref_vlc_symbol group 513

Table G-17 - Association of ctxIdx and syntax elements for each slice type in the initialisation process [Ed (JR) modified Table 9-11 to add EI, EP, and EB slice type, note that syntax elements existing in I, P, B slice and used in PR slice should also have an entry in the table] 515

Table G-18 - Values of variables m and n for ctxIdx from 460 to 466 [Ed. Note(HS): The initialization values currently represent a simple uniform distribution. Suitable initialization values still need to be determined.] 515

Table G-19 - Values of variables m and n for ctxIdx from 467 to 482 [Ed. Note(HS): The initialization values are just copied from the corresponding context with levelListIdx-1. Suitable initialization values still need to be determined.] 516

Table G-20 - Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset [Ed. (DM) Table is incomplete due to missing data for last_significant_coeff_flag, coeff_abs_level_minus1 for PR slices.] 516

Table G-21 - Assignment of ctxIdxInc to binIdx for the ctxIdxOffset values related to the syntax elements base_mode_flag and residual_prediction_flag 518

Table G-22 - Determination of ctxIdxInc for special values of levelListIdx for the syntax element significant_coeff_flag depending on field_pic_flag, mb_field_decoding_flag, and ctxBlockCat 519

Foreword

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardising telecommunications on a world-wide basis. The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups that, in turn, produce Recommendations on these topics. The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1. In some areas of information technology that fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide standardisation. National Bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

This Recommendation | International Standard was prepared jointly by ITU-T SG 16 Q.6, also known as VCEG (Video Coding Experts Group), and by ISO/IEC JTC 1/SC 29/WG 11, also known as MPEG (Moving Picture Experts Group). VCEG was formed in 1997 to maintain prior ITU-T video coding standards and develop new video coding standard(s) appropriate for a wide range of conversational and non-conversational services. MPEG was formed in 1988 to establish standards for coding of moving pictures and associated audio for various applications such as digital storage media, distribution, and communication.

In this Recommendation | International Standard Annexes A through E contain normative requirements and are an integral part of this Recommendation | International Standard.

Advanced video coding for generic audiovisual services

0 Introduction

This clause does not form an integral part of this Recommendation | International Standard.

0.1 Prologue

This subclause does not form an integral part of this Recommendation | International Standard.

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Video Team (JVT) in 2001 for development of a new Recommendation | International Standard.

0.2 Purpose

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard was developed in response to the growing need for higher compression of moving pictures for various applications such as videoconferencing, digital storage media, television broadcasting, internet streaming, and communication. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. The use of this Recommendation | International Standard allows motion video to be manipulated as a form of computer data and to be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

0.3 Applications

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to cover a broad range of applications for video content including but not limited to the following:

CATV	Cable TV on optical networks, copper, etc.
DBS	Direct broadcast satellite video services
DSL	Digital subscriber line video services
DTTB	Digital terrestrial television broadcasting
ISM	Interactive storage media (optical disks, etc.)
MMM	Multimedia mailing
MSPN	Multimedia services over packet networks
RTC	Real-time conversational services (videoconferencing, videophone, etc.)
RVS	Remote video surveillance
SSM	Serial storage media (digital VTR, etc.)

0.4 Publication and versions of this specification

This subclause does not form an integral part of this Recommendation | International Standard.

This specification has been jointly developed by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group. It is published as technically-aligned twin text in both organizations ITU-T and ISO/IEC.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 1 refers to the first (2003) approved version of this Recommendation | International Standard.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 2 refers to the integrated text containing the corrections specified in the first technical corrigendum.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 3 refers to the integrated text containing both the first technical corrigendum (2004) and the first amendment, which is referred to as the "Fidelity range extensions".

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 4 (the current specification) refers to the integrated text containing the first technical corrigendum (2004), the first amendment (the "Fidelity range extensions"), and an additional technical corrigendum (2005). In the ITU-T, the next published version after version 2 was version 4 (due to the completion of the drafting work for version 4 prior to the approval opportunity for a final version 3 text).

0.5 Profiles and levels

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to be generic in the sense that it serves a wide range of applications, bit rates, resolutions, qualities, and services. Applications should cover, among other things, digital storage media, television broadcasting and real-time communications. In the course of creating this Specification, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and these have been integrated into a single syntax. Hence, this Specification will facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profiles" and "levels". These and other related terms are formally defined in clause 3.

A "profile" is a subset of the entire bitstream syntax that is specified by this Recommendation | International Standard. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by syntax elements in the bitstream such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economic to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "levels" are specified within each profile. A level is a specified set of constraints imposed on values of the syntax elements in the bitstream. These constraints may be simple limits on values. Alternatively they may take the form of constraints on arithmetic combinations of values (e.g. picture width multiplied by picture height multiplied by number of pictures decoded per second).

Coded video content conforming to this Recommendation | International Standard uses a common syntax. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream.

0.6 Overview of the design characteristics

This subclause does not form an integral part of this Recommendation | International Standard.

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality. With the exception of the transform bypass mode of operation for lossless coding in the High 4:4:4 profile and the I_PCM mode of operation in all profiles, the algorithm is typically not lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes. A number of techniques may be used to achieve highly efficient compression. Encoding algorithms (not specified in this Recommendation | International Standard) may select between inter and intra coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures. Intra coding uses various spatial prediction modes to exploit spatial statistical dependencies in the source signal for a single picture. Motion vectors and intra prediction modes may be specified for a variety of block sizes in the picture. The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantised, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion

vectors or intra prediction modes are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

0.6.1 Predictive coding

This subclause does not form an integral part of this Recommendation | International Standard.

Because of the conflicting requirements of random access and highly efficient compression, two main coding types are specified. Intra coding is done without reference to other pictures. Intra coding may provide access points to the coded sequence where decoding can begin and continue correctly, but typically also shows only moderate compression efficiency. Inter coding (predictive or bi-predictive) is more efficient using inter prediction of each block of sample values from some previously decoded picture selected by the encoder. In contrast to some other video coding standards, pictures coded using bi-predictive inter prediction may also be used as references for inter coding of other pictures.

The application of the three coding types to pictures in a sequence is flexible, and the order of the decoding process is generally not the same as the order of the source picture capture process in the encoder or the output order from the decoder for display. The choice is left to the encoder and will depend on the requirements of the application. The decoding order is specified such that the decoding of pictures that use inter-picture prediction follows later in decoding order than other pictures that are referenced in the decoding process.

0.6.2 Coding of progressive and interlaced video

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard specifies a syntax and decoding process for video that originated in either progressive-scan or interlaced-scan form, which may be mixed together in the same sequence. The two fields of an interlaced frame are separated in capture time while the two fields of a progressive frame share the same capture time. Each field may be coded separately or the two fields may be coded together as a frame. Progressive frames are typically coded as a frame. For interlaced video, the encoder can choose between frame coding and field coding. Frame coding or field coding can be adaptively selected on a picture-by-picture basis and also on a more localized basis within a coded frame. Frame coding is typically preferred when the video scene contains significant detail with limited motion. Field coding typically works better when there is fast picture-to-picture motion.

0.6.3 Picture partitioning into macroblocks and smaller partitions

This subclause does not form an integral part of this Recommendation | International Standard.

As in previous video coding Recommendations and International Standards, a macroblock, consisting of a 16x16 block of luma samples and two corresponding blocks of chroma samples, is used as the basic processing unit of the video decoding process.

A macroblock can be further partitioned for inter prediction. The selection of the size of inter prediction partitions is a result of a trade-off between the coding gain provided by using motion compensation with smaller blocks and the quantity of data needed to represent the data for motion compensation. In this Recommendation | International Standard the inter prediction process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one-quarter of the luma sample grid spacing displacement. The process for inter prediction of a sample block can also involve the selection of the picture to be used as the reference picture from a number of stored previously-decoded pictures. Motion vectors are encoded differentially with respect to predicted values formed from nearby encoded motion vectors.

Typically, the encoder calculates appropriate motion vectors and other data elements represented in the video data stream. This motion estimation process in the encoder and the selection of whether to use inter prediction for the representation of each region of the video content is not specified in this Recommendation | International Standard.

0.6.4 Spatial redundancy reduction

This subclause does not form an integral part of this Recommendation | International Standard.

Both source pictures and prediction residuals have high spatial redundancy. This Recommendation | International Standard is based on the use of a block-based transform method for spatial redundancy removal. After inter prediction from previously-decoded samples in other pictures or spatial-based prediction from previously-decoded samples within the current picture, the resulting prediction residual is split into 4x4 blocks. These are converted into the transform domain where they are quantised. After quantisation many of the transform coefficients are zero or have low amplitude and can thus be represented with a small amount of encoded data. The processes of transformation and quantisation in the encoder are not specified in this Recommendation | International Standard.

0.7 How to read this specification

This subclause does not form an integral part of this Recommendation | International Standard.

It is suggested that the reader starts with clause 1 (Scope) and moves on to clause 3 (Definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1-7.3 for syntactical order and see subclause 7.4 for semantics; i.e., the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in clause 9 (Parsing process). Finally, clause 8 (Decoding process) specifies how the syntax elements are mapped into decoded samples. Throughout reading this specification, the reader should refer to clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through E also form an integral part of this Recommendation | International Standard.

Annex A specifies seven profiles (Baseline, Main, Extended, High, High 10, High 4:2:2 and High 4:4:4), each being tailored to certain application domains, and defines the so-called levels of the profiles. Annex B specifies syntax and semantics of a byte stream format for delivery of coded video as an ordered stream of bytes. Annex C specifies the hypothetical reference decoder and its use to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information message payloads. Finally, Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

Throughout this specification, statements appearing with the preamble "NOTE -" are informative and are not an integral part of this Recommendation | International Standard.

1 Scope

This document specifies ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 video coding.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- ITU-T Recommendation T.35 (2000), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*.
- ISO/IEC 11578:1996, Annex A, *Universal Unique Identifier*.
- ISO/CIE 10527:1991, *Colorimetric Observers*.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.1 **access unit**: A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more ~~*redundant* ~~*coded pictures* or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*. All *slices* or *slice data partitions* in an access unit have the same value of *picture order count*.

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

3.3 **adaptive binary arithmetic decoding process**: An entropy *decoding process* that derives the values of *bins* from a *bitstream* produced by an *adaptive binary arithmetic encoding process*.

3.4 **adaptive binary arithmetic encoding process**: An entropy *encoding process*, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bins* and produces a *bitstream* that can be decoded using the *adaptive binary arithmetic decoding process*.

3.5 **alpha blending**: A process not specified by this Recommendation | International Standard, in which an *auxiliary coded picture* is used in combination with a *primary coded picture* and with other data not specified by this Recommendation | International Standard in the *display process.* In an alpha blending process, the samples of an *auxiliary coded picture* are interpreted as indications of the degree of opacity (or, equivalently, the degrees of transparency) associated with the corresponding *luma* samples of the *primary coded picture.*

3.6 **arbitrary slice order**: A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *picture* may be less than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *coded picture*.

3.7 **auxiliary coded picture**: A *picture* that supplements the *primary coded picture* that may be used in combination with other data not specified by this Recommendation | International Standard in the *display process.* An auxiliary coded picture has the same syntactic and semantic restrictions as a monochrome *redundant coded picture.* An auxiliary coded picture must contain the same number of *macroblocks* as the *primary coded picture*. Auxiliary coded pictures have no normative effect on the *decoding process*. See also *primary coded picture* and *redundant coded picture.*

3.8 **B slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block.*

3.9 **bin**: One bit of a *bin string.*

3.10 **binarization**: A set of *bin strings* for all possible values of a *syntax element*.

3.11 **binarization process**: A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.

3.12 **bin string**: A string of *bins*. A bin string is an intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.

3.13 **bi-predictive slice:** See *B slice***.**

3.14 **bitstream**: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*. Bitstream is a collective term used to refer either to a *NAL unit stream* or a *byte stream*.

3.15 **block**: An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.

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

3.17 **bottom macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the bottom row of samples for the *macroblock pair*. For a *field macroblock pair*, the bottom macroblock represents the samples from the region of the *bottom field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* that lie within the bottom half of the spatial region of the *macroblock pair.*

3.18 **broken link**: A location in a *bitstream* at which it is indicated that some subsequent *pictures* in *decoding order* may contain serious visual artefacts due to unspecified operations performed in the generation of the *bitstream*.

3.19 **byte**: A sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right. When represented in a sequence of data bits, the most significant bit of a byte is first.

3.20 **byte-aligned**: A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*. A bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned.

3.21 **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.

3.22 **can**: A term used to refer to behaviour that is allowed, but not necessarily required.

3.23 **category**: A number associated with each *syntax element*. The category is used to specify the allocation of *syntax elements* to *NAL units* for *slice data partitioning*. It may also be used in a manner determined by the application to refer to classes of *syntax elements* in a manner not specified in this Recommendation | International Standard.

3.24 **chroma**: An adjective specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours. The symbols used for a chroma array or sample are Cb and Cr.

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

3.25 **coded field**: A *coded representation* of a *field*.

3.26 **coded frame**: A *coded representation* of a *frame*.

3.27 **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.

3.28 **coded picture buffer (CPB)**: A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.

3.29 **coded representation**: A data element as represented in its coded form.

3.30 **coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.

3.31 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame*.

3.32 **complementary field pair:** A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.

3.33 **complementary non-reference field pair**: Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity where the first *field* is not already a paired *field.*

3.34 **complementary reference field pair**: Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of the frame_num *syntax element*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a memory_management_control_operation *syntax element* equal to 5.

3.35 **context variable**: A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.

3.36 **DC transform coefficient**: A *transform coefficient* for which the *frequency index* is zero in all dimensions.

3.37 **decoded picture**: A *decoded picture* is derived by decoding one or more *slices* or *slice data partitions* contained in one *access unit*~~*a coded picture*~~. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.

3.38 **decoded picture buffer (DPB)**: A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex C.

3.39 **decoder**: An embodiment of a *decoding process*.

3.40 **decoding order**: The order in which *syntax elements* are processed by the *decoding process*.

3.41 **decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.

3.42 **direct prediction**: An *inter prediction* for a *block* for which no *motion vector* is decoded. Two direct *prediction* modes are specified that are referred to as spatial direct *prediction* and temporal *prediction* mode.

3.43 **display process**: A process not specified in this Recommendation | International Standard having, as its input, the cropped decoded *pictures* that are the output of the *decoding process*.

3.44 **decoder under test (DUT)**: A *decoder* that is tested for conformance to this Recommendation | International Standard by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing of the output of the two *decoders*.

3.45 **emulation prevention byte**: A *byte* equal to 0x03 that may be present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* contains a *start code prefix*.

3.46 **encoder**: An embodiment of an *encoding process*.

3.47 **encoding process**: A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.

3.48 **field**: An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.

3.49 **field macroblock**: A macroblock containing samples from a single *field*. All *macroblocks* of a *coded field* are field macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be field macroblocks.

3.50 **field macroblock pair**: A *macroblock pair* decoded as two *field macroblocks*.

3.51 **field scan**: A specific sequential ordering of *transform coefficients* that differs from the *zig-zag scan* by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks*.

3.52 **flag**: A variable that can take one of the two possible values 0 and 1.

3.53 **frame**: A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples. A *frame* consists of two *fields*, a *top field* and a *bottom field*.

3.54 **frame macroblock**: A *macroblock* representing samples from the two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all macroblocks of a *coded frame* are frame macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some macroblocks of a *coded frame* may be frame macroblocks.

3.55 **frame macroblock pair**: A *macroblock pair* decoded as two *frame macroblocks*.

3.56 **frequency index**: A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.

3.57 **hypothetical reference decoder (HRD)**: A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.

3.58 **hypothetical stream scheduler (HSS)**: A hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*. The HSS is used for checking the conformance of a *bitstream* or a *decoder*.

3.59 **I slice**: A *slice* that is not an *SI slice* that is decoded using *prediction* only from decoded samples within the same *slice*.

3.60 **informative**: A term used to refer to content provided in this Recommendation | International Standard that is not an integral part of this Recommendation | International Standard. Informative content does not establish any mandatory requirements for conformance to this Recommendation | International Standard.

3.61 **instantaneous decoding refresh (IDR) access unit**: An *access unit* in which the *primary coded picture* is an *IDR picture*.

3.62 **instantaneous decoding refresh (IDR) picture**: A *coded picture* in which all *slices* are *I* or *SI slices* that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after decoding the IDR picture. After the decoding of an IDR picture all following *coded pictures* in *decoding order* can be decoded without *inter prediction* from any *picture* decoded prior to the IDR picture. The first *picture* of each *coded video sequence* is an IDR picture.

3.63 **inter coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter prediction*.

3.64 **inter prediction**: A *prediction* derived from decoded samples of *reference pictures* other than the current *decoded picture*.

3.65 **interpretation sample value**: A possibly-altered value corresponding to a decoded sample value of an *auxiliary coded picture* that may be generated for use in the *display process*. Interpretation sample values are not used in the *decoding process* and have no normative effect on the *decoding process*.

3.66 **intra coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *intra prediction*.

3.67 **intra prediction**: A *prediction* derived from the decoded samples of the same decoded *slice*.

3.68 **intra slice**: See *I slice*.

3.69 **inverse transform**: A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC transform coefficients*.

3.70 **layer**: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.

3.71 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, level is the value of a *transform coefficient* prior to *scaling*.

3.72 **list 0 (list 1) motion vector**: A *motion vector* associated with a *reference index* pointing into *reference picture list 0* (*list 1*).

3.73 **list 0 (list 1) prediction**: *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0* (*list 1*).

3.74 **luma**: An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol or subscript used for luma is Y or L.

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

3.75 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

3.76 **macroblock-adaptive frame/field decoding**: A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks*.

3.77 **macroblock address**: When *macroblock-adaptive frame/field decoding* is not in use, a macroblock address is the index of a macroblock in a *macroblock raster scan* of the *picture* starting with zero for the top-left *macroblock* in a *picture*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.

3.78 **macroblock location**: The two-dimensional coordinates of a *macroblock* in a *picture* denoted by (x, y). For the top left *macroblock* of the *picture* (x, y) is equal to (0, 0). x is incremented by 1 for each *macroblock* column from left to right. When *macroblock-adaptive frame/field decoding* is not in use, y is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, y is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a macroblock is a *bottom macroblock*.

3.79 **macroblock pair**: A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.

3.80 **macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction*.

3.81 **macroblock to slice group map**: A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.

3.82 **map unit to slice group map**: A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.

3.83 **may**: A term used to refer to behaviour that is allowed, but not necessarily required. In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

3.84 **memory management control operation**: Seven operations that control *reference picture marking*.

3.85 **motion vector**: A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.

3.86 **must**: A term used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Recommendation | International Standard. This term is used exclusively in an *informative* context.

3.87 **NAL unit**: A syntax structure containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.

3.88 **NAL unit stream**: A sequence of *NAL units*.

3.89 **non-paired field**: A collective term for a *non-paired reference field* or a *non-paired non-reference field*.

3.90 **non-paired non-reference field:** A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.

3.91 **non-paired reference field**: A decoded *reference field* that is not part of a *complementary reference field pair*.

3.92 **non-reference field**: A *field* coded with nal_ref_idc equal to 0.

3.93 **non-reference frame**: A *frame* coded with nal_ref_idc equal to 0.

3.94 **non-reference picture**: A *picture* coded with nal_ref_idc equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.

3.95 **note**: A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.

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

3.97 **output order**: The order in which the *decoded pictures* are output from the *decoded picture buffer*.

3.98 **P slice**: A *slice* that may be decoded using *intra prediction* from decoded samples within the same *slice* or *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

3.99 **parameter**: A *syntax element* of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.

3.100 **parity**: The parity of a *field* can be *top* or *bottom*.

3.101 **partitioning**: The division of a set into subsets such that each element of the set is in exactly one of the subsets.

3.102 **picture**: A collective term for a *field* or a *frame*.

3.103 **picture parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the pic_parameter_set_id *syntax element* found in each *slice header*.

3.104 **picture order count**: A variable having a value that is non-decreasing with increasing *picture* position in output order relative to the previous *IDR picture* in *decoding order* or relative to the previous *picture* containing the *memory management control operation* that marks all *reference pictures* as “unused for reference”.

3.105 **prediction**: An embodiment of the *prediction process*.

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

3.107 **predictive slice**: See *P slice*.

3.108 **predictor**: A combination of specified values or previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.

3.109 **primary coded picture**: The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | International Standard. The primary coded picture contains all *macroblocks* of the *picture.* The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. See also *redundant coded picture.*

3.110 **profile**: A specified subset of the syntax of this Recommendation | International Standard.

3.111 **quantisation parameter**: A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.

3.112 **random access**: The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.

3.113 **raster scan**: A mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc. rows of the pattern (going down) each scanned from left to right.

3.114 **raw byte sequence payload (RBSP):** A syntax structure containing an integer number of *bytes* that is encapsulated in a *NAL unit*. An RBSP is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and followed by zero or more subsequent bits equal to 0.

3.115 **raw byte sequence payload (RBSP) stop bit:** A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*. The location of the end of the *string of data bits* within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*.

3.116 **recovery point**: A point in the *bitstream* at which the recovery of an exact or an approximate representation of the *decoded pictures* represented by the *bitstream* is achieved after a *random access* or *broken link.*

3.117 **redundant coded picture**: A coded representation of a *picture* or a part of a *picture.* The content of a redundant coded picture shall not be used by the *decoding process* for a *bitstream* conforming to this Recommendation | International Standard. A *redundant coded picture* is not required to contain all *macroblocks* in the *primary coded picture*. Redundant coded pictures have no normative effect on the *decoding process*. See also *primary coded picture*.

3.118 **reference field**: A *reference field* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.

3.119 **reference frame**: A *reference frame* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded frame* are decoded. See also *reference picture*.

3.120 **reference index**: An index into a *reference picture list*.

3.121 **reference picture**: A *picture* with nal_ref_idc not equal to 0. A *reference picture* contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.

3.122 **reference picture list**: A list of *reference pictures* that is used for *inter prediction* of a *P*, *B*, or *SP slice.* For the *decoding process* of a *P* or *SP slice*, there is one reference picture list. For the *decoding process* of a *B slice*, there are two reference picture lists.

3.123 **reference picture list 0**: A *reference picture list* used for *inter prediction* of a *P*, *B*, or *SP slice*. All *inter prediction* used for *P* and *SP slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 1*.

3.124 **reference picture list 1**: A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two lists of *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 0*.

3.125 **reference picture marking**: Specifies, in the bitstream, how the *decoded pictures* are marked for *inter prediction.*

3.126 **reserved**: The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this Recommendation | International Standard, but may be used in future extensions of this Recommendation | International Standard by ITU-T | ISO/IEC.

3.127 **residual**: The decoded difference between a *prediction* of a sample or data element and its decoded value.

3.128 **run**: A number of consecutive data elements represented in the decoding process. In one context, the number of zero-valued *transform coefficient levels* preceding a non-zero *transform coefficient level* in the list of *transform coefficient levels* generated by a *zig-zag scan* or a *field scan*. In other contexts, run refers to a number of *macroblocks*.

3.129 **sample aspect ratio**: Specifies, for assisting the display process, which is not specified in this Recommendation | International Standard, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the *luma* sample array in a *frame*. Sample aspect ratio is expressed as *h*:*v*, where *h* is horizontal width and *v* is vertical height (in arbitrary units of spatial distance).

3.130 **scaling**: The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.

3.131 **sequence parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a seq_parameter_set_id *syntax element* found in the *picture parameter set* referred to by the pic_parameter_set_id *syntax element* found in each *slice header*.

3.132 **shall**: A term used to express mandatory requirements for conformance to this Recommendation | International Standard. When used to express a mandatory constraint on the values of *syntax elements* or on the results obtained by operation of the specified *decoding process*, it is the responsibility of the *encoder* to ensure that the constraint is fulfilled. When used in reference to operations performed by the *decoding process*, any *decoding process* that produces identical results to the *decoding process* described herein conforms to the *decoding process* requirements of this Recommendation | International Standard.

3.133 **should**: A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Recommendation | International Standard.

3.134 **SI slice**: A *slice* that is coded using *prediction* only from decoded samples within the same *slice* and using quantisation of the *prediction* samples. An SI slice can be coded such that its decoded samples can be constructed identically to an *SP slice*.

3.135 **skipped macroblock**: A *macroblock* for which no data is coded other than an indication that the *macroblock* is to be decoded as "skipped". This indication may be common to several *macroblocks*.

3.136 **slice**: An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. For the *primary coded picture*, the division of each *slice group* into slices is a *partitioning*. Although a slice contains *macroblocks* or *macroblock pairs* that are consecutive in the *raster scan* within a *slice group*, these *macroblocks* or *macroblock pairs* are not necessarily consecutive in the *raster scan* within the *picture*. The addresses of the *macroblocks* are derived from the address of the first *macroblock* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*.

3.137 **slice data partitioning**: A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.

3.138 **slice group**: A subset of the *macroblocks* or *macroblock pairs* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture*. The partitioning is specified by the *macroblock to slice group map*.

3.139 **slice group map units**: The units of the *map unit to slice group map*.

3.140 **slice header**: A part of a coded *slice* containing the data elements pertaining to the first or all *macroblocks* represented in the *slice*.

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

3.142 **SP slice**: A *slice* that is coded using *inter prediction* from previously-decoded *reference pictures*, using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*. An SP slice can be coded such that its decoded samples can be constructed identically to another SP slice or an *SI slice*.

3.143 **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*. The location of a start code prefix can be used by a *decoder* to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of *emulation prevention bytes*.

3.144 **string of data bits (SODB)**: A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*. Within an SODB, the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.

3.145 **sub-macroblock**: One quarter of the samples of a *macroblock*, i.e., an 8x8 *luma block* and two corresponding *chroma blocks* of which one corner is located at a corner of the *macroblock*.

3.146 **sub-macroblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction*.

3.147 **switching I slice**: See *SI slice*.

3.148 **switching P slice**: See *SP slice*.

3.149 **syntax element**: An element of data represented in the *bitstream*.

3.150 **syntax structure**: Zero or more *syntax elements* present together in the *bitstream* in a specified order.

3.151 **top field**: One of two *fields* that comprise a *frame*. Each row of a *top field* is spatially located immediately above the corresponding row of the *bottom field*.

3.152 **top macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the top row of samples for the *macroblock pair*. For a *field macroblock pair*, the top macroblock represents the samples from the region of the *top field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the top macroblock represents the samples of the *frame* that lie within the top half of the spatial region of the *macroblock pair*.

3.153 **transform coefficient**: A scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in an *inverse transform* part of the *decoding process*.

3.154 **transform coefficient level**: An integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.

3.155 **universal unique identifier (UUID)**: An identifier that is unique with respect to the space of all universal unique identifiers.

3.156 **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.

3.157 **variable length coding (VLC)**: A reversible procedure for entropy coding that assigns shorter bit strings to *symbols* expected to be more frequent and longer bit strings to *symbols* expected to be less frequent.

3.158 **zig-zag scan**: A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficient levels* in *frame macroblocks*.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply.

CABAC Context-based Adaptive Binary Arithmetic Coding

CAVLC Context-based Adaptive Variable Length Coding

CBR Constant Bit Rate

CPB Coded Picture Buffer

DPB Decoded Picture Buffer

DUT Decoder under test

FIFO First-In, First-Out

HRD Hypothetical Reference Decoder

HSS	Hypothetical Stream Scheduler
IDR	Instantaneous Decoding Refresh
LSB	Least Significant Bit
MB	Macroblock
MBAFF	Macroblock-Adaptive Frame-Field Coding
MSB	Most Significant Bit
NAL	Network Abstraction Layer
RBSP	Raw Byte Sequence Payload
SVC	Scalable Video Coding.
SEI	Supplemental Enhancement Information
SODB	String Of Data Bits
UUID	Universal Unique Identifier
VBR	Variable Bit Rate
VCL	Video Coding Layer
VLC	Variable Length Coding
VUI	Video Usability Information

5 Conventions

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, integer division and arithmetic shift operations are specifically defined. Numbering and counting conventions generally begin from 0.

5.1 Arithmetic operators

The following arithmetic operators are defined as follows.

+	Addition
–	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	Multiplication
x^y	Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	Integer division with truncation of the result toward zero. For example, 7/4 and –7/–4 are truncated to 1 and –7/4 and 7/–4 are truncated to –1.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^{y} f(i)$	The summation of f(i) with i taking all integer values from x up to and including y.
x % y	Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0.

When order of precedence is not indicated explicitly by use of parenthesis, the following rules apply:

- multiplication and division operations are considered to take place before addition and subtraction;
- multiplication and division operations in sequence are evaluated sequentially from left to right;
- addition and subtraction operations in sequence are evaluated sequentially from left to right.

5.2 Logical operators

The following logical operators are defined as follows:

x && y	Boolean logical "and" of x and y
x || y	Boolean logical "or" of x and y
!	Boolean logical "not"
x ? y : z	If x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z

5.3 Relational operators

The following relational operators are defined as follows:

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

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" is treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

5.4 Bit-wise operators

The following bit-wise operators are defined as follows:

&	Bit-wise "and". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
|	Bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
x >> y	Arithmetic right shift of a two’s complement integer representation of x by y binary digits. This function is defined only for positive integer values of y. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation.
x << y	Arithmetic left shift of a two’s complement integer representation of x by y binary digits. This function is defined only for positive integer values of y. Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

5.5 Assignment operators

The following arithmetic operators are defined as follows:

=	Assignment operator.
++	Increment, i.e., x++ is equivalent to $x = x + 1$; when used in an array index, evaluates to the value of the variable prior to the increment operation.
––	Decrement, i.e., x–– is equivalent to $x = x - 1$; when used in an array index, evaluates to the value of the variable prior to the decrement operation.
+=	Increment by amount specified, i.e., x += 3 is equivalent to x = x + 3, and x += (-3) is equivalent to x = x + (-3).
–=	Decrement by amount specified, i.e., x –= 3 is equivalent to x = x – 3, and x –= (-3) is equivalent to x = x – (-3).

5.6 Range notation

The following notation is used to specify a range of values

x = y .. z x takes on integer values starting from y to z inclusive, with x, y, and z being integer numbers.

5.7 Mathematical functions

The following mathematical functions are defined as follows:

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

Ceil(x) the smallest integer greater than or equal to x. (5-2)

$$\text{Clip1}_Y(x) = \text{Clip3}(0, (1 << \text{BitDepth}_Y) - 1, x) \tag{5-3}$$

$$\text{Clip1}_C(x) = \text{Clip3}(0, (1 << \text{BitDepth}_C) - 1, x) \tag{5-4}$$

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \tag{5-5}$$

Floor(x) the greatest integer less than or equal to x. (5-6)

$$\text{InverseRasterScan}(a, b, c, d, e) = \begin{cases} (a\%(d/b))*b; & e == 0 \\ (a/(d/b))*c; & e == 1 \end{cases} \tag{5-7}$$

Log2(x) returns the base-2 logarithm of x. (5-8)

Log10(x) returns the base-10 logarithm of x. (5-9)

$$\text{Median}(x, y, z) = x + y + z - \text{Min}(x, \text{Min}(y, z)) - \text{Max}(x, \text{Max}(y, z)) \tag{5-10}$$

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

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{5-12}$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \tag{5-13}$$

$$\text{Sign}(\ x\ ) = \begin{cases} 1 & ; \quad x >= 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{5-14}$$

$$\text{Sqrt}(\ x\ ) = \sqrt{x} \tag{5-15}$$

5.8 Variables, syntax elements, and tables

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

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

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

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

Functions are described by their names, which are constructed as syntax element names, with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

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

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

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

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

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

5.9 Text description of logical operations

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

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

may be described in the following manner:

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

- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- …
- Otherwise (informative remark on remaining condition), statement n

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

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

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

may be described in the following manner:

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

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

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

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

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

5.10 Processes

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

The assignment of variables is specified as follows.

– If invoking a process, variables are explicitly assigned to lower case input or output variables of the process specification in case these do not have the same name.

– Otherwise (when the variables at the invoking and specification have the same name), assignment is implied.

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

6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships

6.1 Bitstream formats

This subclause specifies the relationship between the NAL unit stream and byte stream, either of which are referred to as the bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes. Methods of framing the NAL units in a manner other than use of the byte stream format are outside the scope of this Recommendation | International Standard. The byte stream format is specified in Annex B.

6.2 Source, decoded, and output picture formats

This subclause specifies the relationship between source and decoded frames and fields that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of either or both frames or fields (called collectively pictures) in decoding order.

The source and decoded pictures (frames or fields) are each comprised of one or more sample arrays:

– Luma (Y) only (monochrome), with or without an auxiliary array.

– Luma and two Chroma (YCbCr or YCgCo), with or without an auxiliary array.

– Green, Blue and Red (GBR, also known as RGB), with or without an auxiliary array.

– Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ), with or without an auxiliary array.

For convenience of notation and terminology in this specification, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in Annex E. The (monochrome) auxiliary arrays, which may or may not be present as auxiliary pictures in a coded video sequence, are optional for decoding and can be used for such purposes as alpha blending.

The variables SubWidthC, and SubHeightC are specified in Table 6-1, depending on the chroma format sampling structure, which is specified through chroma_format_idc. An entry marked as "-" in Table 6-1 denotes an undefined value for SubWidthC or SubHeightC. Other values of chroma_format_idc, SubWidthC, and SubHeightC may be specified in the future by ITU-T | ISO/IEC.

Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc

chroma_format_idc	Chroma Format	SubWidthC	SubHeightC
0	monochrome	–	–
1	4:2:0	2	2
2	4:2:2	2	1
3	4:4:4	1	1

In monochrome sampling there is only one sample array, which is nominally considered the luma array.

In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array.

In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array.

In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array.

The width and height of the luma sample arrays are each an integer multiple of 16. In bitstreams using 4:2:0 chroma sampling, the width and height of chroma sample arrays are each an integer multiple of 8. In bitstreams using 4:2:2 sampling, the width of the chroma sample arrays is an integer multiple of 8 and the height is an integer multiple of 16. The height of a luma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) is an integer multiple of 32. In bitstreams using 4:2:0 chroma sampling, the height of each chroma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) is an integer multiple of 16. The width or height of pictures output from the decoding process need not be an integer multiple of 16 and can be specified using a cropping rectangle.

The syntax for the luma and (when present) chroma arrays are ordered such when data for all three colour components is present, the data for the luma array is first, followed by any data for the Cb array, followed by any data for the Cr array, unless otherwise specified.

The width of fields coded referring to a specific sequence parameter set is the same as that of frames coded referring to the same sequence parameter set (see below). The height of fields coded referring to a specific sequence parameter set is half that of frames coded referring to the same sequence parameter set (see below).

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 12, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

When the value of chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in frames are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

A frame consists of two fields as described below. A coded picture may represent a coded frame or an individual coded field. A coded video sequence conforming to this Recommendation | International Standard may contain arbitrary combinations of coded frames and coded fields. The decoding process is also specified in a manner that allows smaller regions of a coded frame to be coded either as a frame or field region, by use of macroblock-adaptive frame-field coding.

Source and decoded fields are one of two types: top field or bottom field. When two fields are output at the same time, or are combined to be used as a reference frame (see below), the two fields (which shall be of opposite parity) are interleaved. The first (i.e., top), third, fifth, etc. rows of a decoded frame are the top field rows. The second, fourth, sixth, etc. rows of a decoded frame are the bottom field rows. A top field consists of only the top field rows of a decoded frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even rows (for a top field) or the odd rows (for a bottom field) of the decoded frame are used.

When the value of chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in top and bottom fields are shown in Figure 6-2. The nominal vertical sampling relative locations of the chroma samples in a top field are specified as shifted up by one-quarter luma sample height relative to the field-sampling grid. The vertical sampling locations of the chroma samples in a bottom field are specified as shifted down by one-quarter luma sample height relative to the field-sampling grid. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

NOTE – The shifting of the chroma samples is in order for these samples to align vertically to the usual location relative to the full-frame sampling grid as shown in Figure 6-1.

Figure 6-2 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields

When the value of chroma_format_idc is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a frame and in fields are as shown in Figure 6-3 andFigure 6-4, respectively.

Figure 6-3 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a frame

Figure 6-4 – Nominal vertical and horizontal sampling locations of 4:2:2 samples top and bottom fields

When the value of chroma_format_idc is equal to 3, all array samples are co-sited for all cases of frames and fields and the nominal locations in a frame and in fields are as shown in Figure 6-5 and Figure 6-6, respectively.

Figure 6-5 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a frame

Figure 6-6 – Nominal vertical and horizontal sampling locations of 4:4:4 samples top and bottom fields

The samples are processed in units of macroblocks. The luma array for each macroblock is 16 samples in both width and height. The variables MbWidthC and MbHeightC, which specify the width and height, respectively, of the chroma arrays for each macroblock, are derived as follows.

– If chroma_format_idc is equal to 0 (monochrome), MbWidthC and MbHeightC are both equal to 0 (as no chroma arrays are specified for monochrome video).

– Otherwise, MbWidthC and MbHeightC are derived as

$$MbWidthC = 16 / SubWidthC \quad (6\text{-}1)$$
$$MbHeightC = 16 / SubHeightC \quad (6\text{-}2)$$

6.3 Spatial subdivision of pictures and slices

This subclause specifies how a picture is partitioned into slices and macroblocks. Pictures are divided into slices. A slice is a sequence of macroblocks, or, when macroblock-adaptive frame/field decoding is in use, a sequence of macroblock pairs.

Each macroblock is comprised of one 16x16 luma array and, when the video format is not monochrome, two corresponding chroma sample arrays. When macroblock-adaptive frame/field decoding is not in use, each macroblock represents a spatial rectangular region of the picture. For example, a picture may be divided into two slices as shown in Figure 6-7.

Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices

When macroblock-adaptive frame/field decoding is in use, the picture is partitioned into slices containing an integer number of macroblock pairs as shown in Figure 6-8. Each macroblock pair consists of two macroblocks.

Figure 6-8 – Partitioning of the decoded frame into macroblock pairs

6.4 Inverse scanning processes and derivation processes for neighbours

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

6.4.1 Inverse macroblock scanning process

Input to this process is a macroblock address mbAddr.

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

The inverse macroblock scanning process is specified as follows.

– If MbaffFrameFlag is equal to 0,

$$x = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0) \quad (6\text{-}3)$$

$$y = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1) \quad (6\text{-}4)$$

– Otherwise (MbaffFrameFlag is equal to 1), the following applies.

$$xO = \text{InverseRasterScan}(\text{mbAddr} / 2, 16, 32, \text{PicWidthInSamples}_L, 0) \quad (6\text{-}5)$$

$$yO = \text{InverseRasterScan}(\text{mbAddr} / 2, 16, 32, \text{PicWidthInSamples}_L, 1) \quad (6\text{-}6)$$

Depending on the current macroblock the following applies.

– If the current macroblock is a frame macroblock

$$x = xO \quad (6\text{-}7)$$

$$y = yO + (\text{mbAddr} \% 2) * 16 \quad (6\text{-}8)$$

– Otherwise (the current macroblock is a field macroblock),

$$x = xO \tag{6-9}$$

$$y = yO + (mbAddr \% 2) \tag{6-10}$$

6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process

Macroblocks or sub-macroblocks may be partitioned, and the partitions are scanned for inter prediction as shown in Figure 6-9. The outer rectangles refer to the samples in a macroblock or sub-macroblock, respectively. The rectangles refer to the partitions. The number in each rectangle specifies the index of the inverse macroblock partition scan or inverse sub-macroblock partition scan.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Tables 7-13, 7-14, 7-17, and 7-18. MbPartWidth( ) and MbPartHeight( ) are set to appropriate values for each macroblock, depending on the macroblock type. SubMbPartWidth( ) and SubMbPartHeight( ) are set to appropriate values for each sub-macroblock of a macroblock with mb_type equal to P_8x8, P_8x8ref0, or B_8x8, depending on the sub-macroblock type.

Figure 6-9 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans

6.4.2.1 Inverse macroblock partition scanning process

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

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

The inverse macroblock partition scanning process is specified by

$$x = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 0) \tag{6-11}$$

$$y = InverseRasterScan(mbPartIdx, MbPartWidth(mb_type), MbPartHeight(mb_type), 16, 1) \tag{6-12}$$

6.4.2.2 Inverse sub-macroblock partition scanning process

Inputs to this process are the index of a macroblock partition mbPartIdx and the index of a sub-macroblock partition subMbPartIdx.

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

The inverse sub-macroblock partition scanning process is specified as follows.

– If mb_type is equal to P_8x8, P_8x8ref0, or B_8x8,

x = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]), SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 0) (6-13)

y = InverseRasterScan(subMbPartIdx, SubMbPartWidth(sub_mb_type[mbPartIdx]), SubMbPartHeight(sub_mb_type[mbPartIdx]), 8, 1) (6-14)

– Otherwise,

x = InverseRasterScan(subMbPartIdx, 4, 4, 8, 0) (6-15)

y = InverseRasterScan(subMbPartIdx, 4, 4, 8, 1) (6-16)

6.4.3 Inverse 4x4 luma block scanning process

Input to this process is the index of a 4x4 luma block luma4x4BlkIdx.

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

Figure 6-10 shows the scan for the 4x4 luma blocks.

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

Figure 6-10 – Scan for 4x4 luma blocks

The inverse 4x4 luma block scanning process is specified by

x = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 0) + InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 0) (6-17)

y = InverseRasterScan(luma4x4BlkIdx / 4, 8, 8, 16, 1) + InverseRasterScan(luma4x4BlkIdx % 4, 4, 4, 8, 1) (6-18)

6.4.4 Inverse 8x8 luma block scanning process

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

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

Figure 6-11 shows the scan for the 8x8 luma blocks.

0	1
2	3

Figure 6-11 – Scan for 8x8 luma blocks

The inverse 8x8 luma block scanning process is specified by

x = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 0) (6-19)

y = InverseRasterScan(luma8x8BlkIdx, 8, 8, 16, 1) (6-20)

6.4.5 Derivation process of the availability for macroblock addresses

Input to this process is a macroblock address mbAddr.

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

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

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

– mbAddr < 0

– mbAddr > CurrMbAddr

– the macroblock with address mbAddr belongs to a different slice than the macroblock with address CurrMbAddr

6.4.6 Derivation process for neighbouring macroblock addresses and their availability

This process can only be invoked when MbaffFrameFlag is equal to 0.

The outputs of this process are

– mbAddrA: the address and availability status of the macroblock to the left of the current macroblock.

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

– mbAddrC: the address and availability status of the macroblock above-right of the current macroblock.

– mbAddrD: the address and availability status of the macroblock above-left of the current macroblock.

Figure 6-12 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

mbAddrD	mbAddrB	mbAddrC
mbAddrA	CurrMbAddr	

Figure 6-12 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.5 is mbAddrA = CurrMbAddr – 1 and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when CurrMbAddr % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.5 is mbAddrB = CurrMbAddr – PicWidthInMbs and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.5 is mbAddrC = CurrMbAddr – PicWidthInMbs + 1 and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when (CurrMbAddr + 1) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.5 is mbAddrD = CurrMbAddr – PicWidthInMbs - 1 and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when CurrMbAddr % PicWidthInMbs is equal to 0.

6.4.7 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames

This process can only be invoked when MbaffFrameFlag is equal to 1.

The outputs of this process are

– mbAddrA: the address and availability status of the top macroblock of the macroblock pair to the left of the current macroblock pair.

– mbAddrB: the address and availability status of the top macroblock of the macroblock pair above the current macroblock pair.

– mbAddrC: the address and availability status of the top macroblock of the macroblock pair above-right of the current macroblock pair.

– mbAddrD: the address and availability status of the top macroblock of the macroblock pair above-left of the current macroblock pair.

Figure 6-13 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

mbAddrA, mbAddrB, mbAddrC, and mbAddrD have identical values regardless whether the current macroblock is the top or the bottom macroblock of a macroblock pair.

mbAddrD	mbAddrB	mbAddrC
mbAddrA	CurrMbAddr or	

Figure 6-13 – Neighbouring macroblocks for a given macroblock in MBAFF frames

Input to the process in subclause 6.4.5 is mbAddrA =2 * (CurrMbAddr / 2 – 1) and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.5 is mbAddrB =2 * (CurrMbAddr / 2 – PicWidthInMbs) and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.5 is mbAddrC = 2 * (CurrMbAddr / 2 – PicWidthInMbs + 1) and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when (CurrMbAddr / 2 + 1) % PicWidthInMbs is equal to 0.

Input to the process in subclause 6.4.5 is mbAddrD = 2 * (CurrMbAddr / 2 – PicWidthInMbs - 1) and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when (CurrMbAddr / 2) % PicWidthInMbs is equal to 0.

6.4.8 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.8.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.8.2 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.4.8.3 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.8.4 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.8.5 specifies the derivation process for neighbouring partitions.

Table 6-2 specifies the values for the difference of luma location (xD, yD) for the input and the replacement for N in mbAddrN, mbPartIdxN, subMbPartIdxN, luma8x8BlkIdxN, luma4x4BlkIdxN, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.8.1 to 6.4.8.5. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.8.1 to 6.4.8.5

N	xD	yD
A	-1	0
B	0	-1
C	predPartWidth	-1
D	-1	-1

Figure 6-14 illustrates the relative location of the neighbouring macroblocks, blocks, or partitions A, B, C, and D to the current macroblock, partition, or block, when the current macroblock, partition, or block is in frame coding mode.

Figure 6-14 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.4.8.1 Derivation process for neighbouring macroblocks

Outputs of this process are

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

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

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

– The difference of luma location (xD, yD) is set according to Table 6-2.

– The derivation process for neighbouring locations as specified in subclause 6.4.9 is invoked for luma locations with (xN, yN) equal to (xD, yD), and the output is assigned to mbAddrN.

6.4.8.2 Derivation process for neighbouring 8x8 luma block

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

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

Outputs of this process are

– mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,

– luma8x8BlkIdxA: the index of the 8x8 luma block to the left of the 8x8 block with index luma8x8BlkIdx and its availability status,

– mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,

– luma8x8BlkIdxB: the index of the 8x8 luma block above the 8x8 block with index luma8x8BlkIdx and its availability status.

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

– The difference of luma location (xD, yD) is set according to Table 6-2.

– The luma location (xN, yN) is specified by

$$xN = (luma8x8BlkIdx \% 2) * 8 + xD \tag{6-21}$$

$$yN = (luma8x8BlkIdx / 2) * 8 + yD \tag{6-22}$$

– The derivation process for neighbouring locations as specified in subclause 6.4.9 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

– The variable luma8x8BlkIdxN is derived as follows.

- If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.
- Otherwise (mbAddrN is available), the 8x8 luma block in the macroblock mbAddrN covering the luma location (xW, yW) is assigned to luma8x8BlkIdxN.

6.4.8.3 Derivation process for neighbouring 4x4 luma blocks

Input to this process is a 4x4 luma block index luma4x4BlkIdx.

Outputs of this process are

- mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- luma4x4BlkIdxA: the index of the 4x4 luma block to the left of the 4x4 block with index luma4x4BlkIdx and its availability status,
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma4x4BlkIdxB: the index of the 4x4 luma block above the 4x4 block with index luma4x4BlkIdx and its availability status.

mbAddrN and luma4x4BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (xD, yD) is set according to Table 6-2.
- The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) as the output.
- The luma location (xN, yN) is specified by

$$xN = x + xD \tag{6-23}$$

$$yN = y + yD \tag{6-24}$$

- The derivation process for neighbouring locations as specified in subclause 6.4.9 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- The variable luma4x4BlkIdxN is derived as follows.
  - If mbAddrN is not available, luma4x4BlkIdxN is marked as not available.
  - Otherwise (mbAddrN is available), the 4x4 luma block in the macroblock mbAddrN covering the luma location (xW, yW) is assigned to luma4x4BlkIdxN.

6.4.8.4 Derivation process for neighbouring 4x4 chroma blocks

Input to this process is a 4x4 chroma block index chroma4x4BlkIdx.

Outputs of this process are

- mbAddrA (either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock) and its availability status,
- chroma4x4BlkIdxA (the index of the 4x4 chroma block to the left of the 4x4 chroma block with index chroma4x4BlkIdx) and its availability status,
- mbAddrB (either equal to CurrMbAddr or the address of the macroblock above the current macroblock) and its availability status,
- chroma4x4BlkIdxB (the index of the 4x4 chroma block above the 4x4 chroma block with index chroma4x4BlkIdx) and its availability status.

mbAddrN and chroma4x4BlkIdxN (with N being A or B) are derived as follows.

- The difference of chroma location (xD, yD) is set according to Table 6-2.

– Depending on chroma_format_idc, the position (x, y) of the upper-left sample of the 4x4 chroma block with index chroma4x4BlkIdx is derived as follows

  – If chroma_format_idc is equal to 1 or 2, the following applies

$$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \quad (6\text{-}25)$$

$$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \quad (6\text{-}26)$$

  – Otherwise (chroma_format_idc is equal to 3), the following applies

$$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx} / 4, 8, 8, 16, 0) + \text{InverseRasterScan}(\text{chroma4x4BlkIdx} \% 4, 4, 4, 8, 0) \quad (6\text{-}27)$$

$$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx} / 4, 8, 8, 16, 1) + \text{InverseRasterScan}(\text{chroma4x4BlkIdx} \% 4, 4, 4, 8, 1) \quad (6\text{-}28)$$

– The chroma location (xN, yN) is specified by

$$xN = x + xD \quad (6\text{-}29)$$

$$yN = y + yD \quad (6\text{-}30)$$

– The derivation process for neighbouring locations as specified in subclause 6.4.9 is invoked for chroma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

– The variable chroma4x4BlkIdxN is derived as follows.

  – If mbAddrN is not available, chroma4x4BlkIdxN is marked as not available.

  – Otherwise (mbAddrN is available), the 4x4 chroma block in the macroblock mbAddrN covering the chroma location (xW, yW) is assigned to chroma4x4BlkIdxN.

6.4.8.5 Derivation process for neighbouring partitions

Inputs to this process are

– a macroblock partition index mbPartIdx

– a current sub-macroblock type currSubMbType

– a sub-macroblock partition index subMbPartIdx

Outputs of this process are

– mbAddrA\mbPartIdxA\subMbPartIdxA: specifying the macroblock or sub-macroblock partition to the left of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrB\mbPartIdxB\subMbPartIdxB: specifying the macroblock or sub-macroblock partition above the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrC\mbPartIdxC\subMbPartIdxC: specifying the macroblock or sub-macroblock partition to the right-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,

– mbAddrD\mbPartIdxD\subMbPartIdxD: specifying the macroblock or sub-macroblock partition to the left-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status.

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

– The inverse macroblock partition scanning process as described in subclause 6.4.2.1 is invoked with mbPartIdx as the input and (x, y) as the output.

– The location of the upper-left luma sample inside a macroblock partition (xS, yS) is derived as follows.

  – If mb_type is equal to P_8x8, P_8x8ref0 or B_8x8, the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 is invoked with subMbPartIdx as the input and (xS, yS) as the output.

  – Otherwise, (xS, yS) are set to (0, 0).

– The variable predPartWidth in Table 6-2 is specified as follows.

  – If mb_type is equal to P_Skip, B_Skip, or B_Direct_16x16, predPartWidth = 16.

  – Otherwise, if mb_type is equal to B_8x8, the following applies.

    – If currSubMbType is equal to B_Direct_8x8, predPartWidth = 16.

      NOTE 1 – When currSubMbType is equal to B_Direct_8x8 and direct_spatial_mv_pred_flag is equal to 1, the predicted motion vector is the predicted motion vector for the complete macroblock.

    – Otherwise, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).

  – Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).

  – Otherwise, predPartWidth = MbPartWidth(mb_type).

– The difference of luma location (xD, yD) is set according to Table 6-2.

– The neighbouring luma location (xN, yN) is specified by

$$xN = x + xS + xD \tag{6-31}$$

$$yN = y + yS + yD \tag{6-32}$$

– The derivation process for neighbouring locations as specified in subclause 6.4.9 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).

– Depending on mbAddrN, the following applies.

  – If mbAddrN is not available, the macroblock or sub-macroblock partition – mbAddrN\mbPartIdxN\subMbPartIdxN is marked as not available.

  – Otherwise (mbAddrN is available), the following applies.

    – The macroblock partition in the macroblock mbAddrN covering the luma location (xW, yW) is assigned to mbPartIdxN and the sub-macroblock partition inside the macroblock partition mbPartIdxN covering the sample (xW, yW) in the macroblock mbAddrN is assigned to subMbPartIdxN.

    – When the partition given by mbPartIdxN and subMbPartIdxN is not yet decoded, the macroblock partition mbPartIdxN and the sub-macroblock partition subMbPartIdxN are marked as not available.

  NOTE– The latter condition is, for example, the case when mbPartIdx = 2, subMbPartIdx = 3, xD = 4, yD = -1, i.e., when neighbour C of the last 4x4 luma block of the third sub-macroblock is requested.

6.4.9 Derivation process for neighbouring locations

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

Outputs of this process are

– mbAddrN: either equal to CurrMbAddr or to the address of neighbouring macroblock that contains (xN, yN) and its availability status,

– (xW, yW): the location (xN, yN) expressed relative to the upper-left corner of the macroblock mbAddrN (rather than relative to the upper-left corner of the current macroblock).

Let maxW and maxH be variables specifying maximum values of the location components xN, xW, and yN, yW, respectively. maxW and maxH are derived as follows.

– If this process is invoked for neighbouring luma locations,

$$maxW = maxH = 16 \tag{6-33}$$

– Otherwise (this process is invoked for neighbouring chroma locations),

$$maxW = MbWidthC \tag{6-34}$$

$$maxH = MbHeightC \tag{6-35}$$

Depending on the variable MbaffFrameFlag, the neighbouring locations are derived as follows.

– If MbaffFrameFlag is equal to 0, the specification for neighbouring locations in fields and non-MBAFF frames as described in subclause 6.4.9.1 is applied.

– Otherwise (MbaffFrameFlag is equal to 1), the specification for neighbouring locations in MBAFF frames as described in subclause 6.4.9.2 is applied.

6.4.9.1 Specification for neighbouring locations in fields and non-MBAFF frames

The specifications in this subclause are applied when MbaffFrameFlag is equal to 0.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.6 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-3 specifies mbAddrN depending on (xN, yN).

Table 6-3 – Specification of mbAddrN

xN	**yN**	**mbAddrN**
< 0	< 0	mbAddrD
< 0	0 .. maxH - 1	mbAddrA
0 .. maxW - 1	< 0	mbAddrB
0 .. maxW - 1	0 .. maxH - 1	CurrMbAddr
> maxW - 1	< 0	mbAddrC
> maxW - 1	0 .. maxH - 1	not available
	> maxH - 1	not available

The neighbouring location (xW, yW) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$xW = (xN + maxW) \% maxW \tag{6-36}$$

$$yW = (yN + maxH) \% maxH \tag{6-37}$$

6.4.9.2 Specification for neighbouring locations in MBAFF frames

The specifications in this subclause are applied when MbaffFrameFlag is equal to 1.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.7 is invoked with mbAddrA, mbAddrB, mbAddrC, and mbAddrD as well as their availability status as the output.

Table 6-4 specifies the macroblock addresses mbAddrN and yM in two ordered steps:

1. Specification of a macroblock address mbAddrX depending on (xN, yN) and the following variables:
   - The variable currMbFrameFlag is derived as follows.
     - If the macroblock with address CurrMbAddr is a frame macroblock, currMbFrameFlag is set equal to 1,
     - Otherwise (the macroblock with address CurrMbAddr is a field macroblock), currMbFrameFlag is set equal to 0.
   - The variable mbIsTopMbFlag is derived as follows.
     - If the macroblock with address CurrMbAddr is a top macroblock (CurrMbAddr % 2 is equal to 0), mbIsTopMbFlag is set equal to 1;
     - Otherwise (the macroblock with address CurrMbAddr is a bottom macroblock, CurrMbAddr % 2 is equal to 1), mbIsTopMbFlag is set equal to 0.
2. Depending on the availability of mbAddrX, the following applies.
   - If mbAddrX is not available, mbAddrN is marked as not available.
   - Otherwise (mbAddrX is available), mbAddrN is marked as available and Table 6-4 specifies mbAddrN and yM depending on (xN, yN), currMbFrameFlag, mbIsTopMbFlag, and the variable mbAddrXFrameFlag, which is derived as follows.
     - If the macroblock mbAddrX is a frame macroblock, mbAddrXFrameFlag is set equal to 1,
     - Otherwise (the macroblock mbAddrX is a field macroblock), mbAddrXFrameFlag is set equal to 0.

Unspecified values (na) of the above flags in Table 6-4 indicate that the value of the corresponding flag is not relevant for the current table rows.

Table 6-4 – Specification of mbAddrN and yM

xN	yN	currMbFrameFlag	mbIsTopMbFlag	mbAddrX	mbAddrXFrameFlag	additional condition	mbAddrN	yM
< 0	< 0	1	1	mbAddrD			mbAddrD + 1	yN
			0	mbAddrA	1		mbAddrA	yN
					0		mbAddrA + 1	(yN + maxH) >> 1
		0	1	mbAddrD	1		mbAddrD + 1	2*yN
					0		mbAddrD	yN
			0	mbAddrD			mbAddrD + 1	yN
< 0	0 .. maxH - 1	1	1	mbAddrA	1		mbAddrA	yN
					0	yN % 2 = = 0	mbAddrA	yN >> 1
						yN % 2 != 0	mbAddrA + 1	yN >> 1
			0	mbAddrA	1		mbAddrA + 1	yN
					0	yN % 2 = = 0	mbAddrA	(yN + maxH) >> 1
						yN % 2 != 0	mbAddrA + 1	(yN + maxH) >> 1
		0	1	mbAddrA	1	yN < (maxH / 2)	mbAddrA	yN <<1
						yN >= (maxH / 2)	mbAddrA + 1	(yN <<1) - maxH
					0		mbAddrA	yN
			0	mbAddrA	1	yN < (maxH / 2)	mbAddrA	(yN <<1) + 1
						yN >= (maxH / 2)	mbAddrA + 1	(yN <<1) + 1 – maxH
					0		mbAddrA + 1	yN
0 .. maxW – 1	< 0	1	1	mbAddrB			mbAddrB + 1	yN
			0	CurrMbAddr			CurrMbAddr - 1	yN
		0	1	mbAddrB	1		mbAddrB + 1	2 * yN
					0		mbAddrB	yN
			0	mbAddrB			mbAddrB + 1	yN
0 .. maxW – 1	0 .. maxH - 1			CurrMbAddr			CurrMbAddr	yN
> maxW – 1	<0	1	1	mbAddrC			mbAddrC + 1	yN
			0	not available			not available	na
		0	1	mbAddrC	1		mbAddrC + 1	2 * yN
					0		mbAddrC	yN
			0	mbAddrC			mbAddrC + 1	yN
> maxW – 1	0 .. maxH - 1			not available			not available	na
	> maxH - 1			not available			not available	na

The neighbouring luma location (xW, yW) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$xW = (xN + maxW) \% maxW \tag{6-38}$$

$$yW = (yM + maxH) \% maxH \tag{6-39}$$

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

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

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

The following table lists examples of pseudo code used to describe the syntax. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

	C	Descriptor
/* A statement can be a syntax element with an associated syntax category and descriptor or can be an expression used to specify conditions for the existence, type, and quantity of syntax elements, as in the following two examples */		
syntax_element	3	ue(v)
conditioning statement		
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */		
{		
statement		
statement		
…		
}		
/* A “while” structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */		
while(condition)		
statement		
/* A “do … while” structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */		
do		
statement		
while(condition)		
/* An “if … else” structure specifies a test of whether a condition is true, and if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The “else” part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */		
if(condition)		
primary statement		
else		
alternative statement		
/* A “for” structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */		

for(initial statement; condition; subsequent statement)		
primary statement		

7.2 Specification of syntax functions, categories, and descriptors

The functions presented here are used in the syntactical description. These functions assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream.

byte_aligned( ) is specified as follows.

- If the current position in the bitstream is on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of byte_aligned( ) is equal to TRUE.
- Otherwise, the return value of byte_aligned( ) is equal to FALSE.

more_data_in_byte_stream( ), which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows.

- If more data follow in the byte stream, the return value of more_data_in_byte_stream( ) is equal to TRUE.
- Otherwise, the return value of more_data_in_byte_stream( ) is equal to FALSE.

more_rbsp_data( ) is specified as follows.

- If there is more data in an RBSP before rbsp_trailing_bits( ), the return value of more_rbsp_data( ) is equal to TRUE.
- Otherwise, the return value of more_rbsp_data( ) is equal to FALSE.

The method for enabling determination of whether there is more data in the RBSP is specified by the application (or in Annex B for applications that use the byte stream format).

more_rbsp_trailing_data( ) is specified as follows.

- If there is more data in an RBSP, the return value of more_rbsp_trailing_data( ) is equal to TRUE.
- Otherwise, the return value of more_rbsp_trailing_data( ) is equal to FALSE.

next_bits(n) provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument. When used within the byte stream as specified in Annex B, next_bits(n) returns a value of 0 if fewer than n bits remain within the byte stream.

read_bits(n) reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) is specified to return a value equal to 0 and to not advance the bitstream pointer.

Categories (labelled in the table as **C**) specify the partitioning of slice data into at most three slice data partitions. Slice data partition A contains all syntax elements of category 2. Slice data partition B contains all syntax elements of category 3. Slice data partition C contains all syntax elements of category 4. The meaning of other category values is not specified. For some syntax elements, two category values, separated by a vertical bar, are used. In these cases, the category value to be applied is further specified in the text. For syntax structures used within other syntax structures, the categories of all syntax elements found within the included syntax structure are listed, separated by a vertical bar. A syntax element or syntax structure with category marked as "All" is present within all syntax structures that include that syntax element or syntax structure. For syntax structures used within other syntax structures, a numeric category value provided in a syntax table at the location of the inclusion of a syntax structure containing a syntax element with category marked as "All" is considered to apply to the syntax elements with category "All".

The following descriptors specify the parsing process of each syntax element. For some syntax elements, two descriptors, separated by a vertical bar, are used. In these cases, the left descriptors apply when entropy_coding_mode_flag is equal to 0 and the right descriptor applies when entropy_coding_mode_flag is equal to 1.

- ae(v): context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.3.
- b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read_bits(8).

– ce(v): context-adaptive variable-length entropy-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.

– f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read_bits(n).

– i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a two's complement integer representation with most significant bit written first.

– me(v): mapped Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

– se(v): signed integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

– te(v): truncated Exp-Golomb-coded syntax element with left bit first. The parsing process for this descriptor is specified in subclause 9.1.

– u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with most significant bit written first.

– ue(v): unsigned integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

nal_unit(NumBytesInNALunit) {	C	Descriptor
forbidden_zero_bit	All	f(1)
nal_ref_idc	All	u(2)
nal_unit_type	All	u(5)
nalUnitHeaderBytes = 1		
if(nal_unit_type = = 20 || nal_unit_type = = 21) {		
nal_unit_header_svc_extension( ) /* specified in Annex G */		
}		
NumBytesInRBSP = 0		
for(i = nalUnitHeaderBytes~~1~~; i < NumBytesInNALunit; i++) {		
if(i + 2 < NumBytesInNALunit && next_bits(24) = = 0x000003) {		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
i += 2		
emulation_prevention_three_byte /* equal to 0x03 */	All	f(8)
} else		
rbsp_byte[NumBytesInRBSP++ **]**	All	b(8)
}		
}		

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc = = 100 | | profile_idc = = 110 | | profile_idc = = 122 | | profile_idc = = 144 | | profile_idc = = 83) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc = = 3)		
residual_colour_transform_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < 8; i++) {		
seq_scaling_list_present_flag[i **]**	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i – 6], 64, UseDefaultScalingMatrix8x8Flag[i – 6])	0	
}		
}		
if(profile_idc = = 83) {		
seq_parameter_set_svc_extension( ) /* specified in Annex G */		
}		
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type = = 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type = = 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		

offset_for_ref_frame[i **]**	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters( )	0	
rbsp_trailing_bits( )	0	
}		

7.3.2.1.1 Scaling list syntax

scaling_list(scalingList, sizeOfScalingList, useDefaultScalingMatrixFlag) {	**C**	**Descriptor**
lastScale = 8		
nextScale = 8		
for(j = 0; j < sizeOfScalingList; j++) {		
if(nextScale != 0) {		
delta_scale	0 | 1	se(v)
nextScale = (lastScale + delta_scale + 256) % 256		
useDefaultScalingMatrixFlag = (j = = 0 && nextScale = = 0)		
}		
scalingList[j] = (nextScale = = 0) ? lastScale : nextScale		
lastScale = scalingList[j]		
}		
}		

7.3.2.1.2 Sequence parameter set extension RBSP syntax

seq_parameter_set_extension_rbsp( ) {	**C**	**Descriptor**
seq_parameter_set_id	10	ue(v)
aux_format_idc	10	ue(v)
if(aux_format_idc != 0) {		
bit_depth_aux_minus8	10	ue(v)

alpha_incr_flag	10	u(1)
alpha_opaque_value	10	u(v)
alpha_transparent_value	10	u(v)
}		
additional_extension_flag	10	u(1)
rbsp_trailing_bits()	10	
}		

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type = = 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup **]**	1	ue(v)
else if(slice_group_map_type = = 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup **]**	1	ue(v)
bottom_right[iGroup **]**	1	ue(v)
}		
else if(slice_group_map_type = = 3 || slice_group_map_type = = 4 || slice_group_map_type = = 5) {		
slice_group_change_direction_flag	1	u(1)
slice_group_change_rate_minus1	1	ue(v)
} else if(slice_group_map_type = = 6) {		
pic_size_in_map_units_minus1	1	ue(v)
for(i = 0; i <= pic_size_in_map_units_minus1; i++)		
slice_group_id[i **]**	1	u(v)
}		
}		
num_ref_idx_l0_active_minus1	1	ue(v)
num_ref_idx_l1_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)

redundant_pic_cnt_present_flag	1	u(1)
if(more_rbsp_data( )) {		
transform_8x8_mode_flag	1	u(1)
pic_scaling_matrix_present_flag	1	u(1)
if(pic_scaling_matrix_present_flag)		
for(i = 0; i < 6 + 2* transform_8x8_mode_flag; i++) {		
pic_scaling_list_present_flag[i **]**	1	u(1)
if(pic_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	1	
else		
scaling_list(ScalingList8x8[i – 6], 64, UseDefaultScalingMatrix8x8Flag[i – 6])	1	
}		
second_chroma_qp_index_offset	1	se(v)
}		
rbsp_trailing_bits( )	1	
}		

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	**C**	**Descriptor**
do		
sei_message( )	5	
while(more_rbsp_data( ))		
rbsp_trailing_bits( )	5	
}		

7.3.2.3.1 Supplemental enhancement information message syntax

sei_message( ) {	C	Descriptor
payloadType = 0		
while(next_bits(8) = = 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadType += 255		
}		
last_payload_type_byte	5	u(8)
payloadType += last_payload_type_byte		
payloadSize = 0		
while(next_bits(8) = = 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadSize += 255		
}		
last_payload_size_byte	5	u(8)
payloadSize += last_payload_size_byte		
sei_payload(payloadType, payloadSize)	5	
}		

7.3.2.4 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp( ) {	C	Descriptor
primary_pic_type	6	u(3)
rbsp_trailing_bits( )	6	
}		

7.3.2.5 End of sequence RBSP syntax

end_of_seq_rbsp( ) {	C	Descriptor
}		

7.3.2.6 End of stream RBSP syntax

end_of_stream_rbsp( ) {	C	Descriptor
}		

7.3.2.7 Filler data RBSP syntax

filler_data_rbsp( ) {	C	Descriptor
while(next_bits(8) = = 0xFF)		
ff_byte /* equal to 0xFF */	9	f(8)
rbsp_trailing_bits( )	9	
}		

7.3.2.8 Slice layer without partitioning RBSP syntax

slice_layer_without_partitioning_rbsp( ) {	C	Descriptor
slice_header( )	2	
slice_data( ) /* all categories of slice_data( ) syntax */	2 | 3 | 4	
rbsp_slice_trailing_bits( )	2	
}		

7.3.2.9 Slice data partition RBSP syntax

7.3.2.9.1 Slice data partition A RBSP syntax

slice_data_partition_a_layer_rbsp( ) {	C	Descriptor
slice_header( )	2	
slice_id	All	ue(v)
slice_data( ) /* only category 2 parts of slice_data( ) syntax */	2	
rbsp_slice_trailing_bits( )	2	
}		

7.3.2.9.2 Slice data partition B RBSP syntax

slice_data_partition_b_layer_rbsp( ) {	C	Descriptor
slice_id	All	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	ue(v)
slice_data( ) /* only category 3 parts of slice_data( ) syntax */	3	
rbsp_slice_trailing_bits( )	3	
}		

7.3.2.9.3 Slice data partition C RBSP syntax

slice_data_partition_c_layer_rbsp( ) {	C	Descriptor
slice_id	All	ue(v)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	ue(v)
slice_data( ) /* only category 4 parts of slice_data( ) syntax */	4	
rbsp_slice_trailing_bits( )	4	
}		

7.3.2.10 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits( ) {	C	Descriptor
rbsp_trailing_bits( )	All	
if(entropy_coding_mode_flag)		
while(more_rbsp_trailing_data( ))		
cabac_zero_word /* equal to 0x0000 */	All	f(16)
}		

7.3.2.11 RBSP trailing bits syntax

rbsp_trailing_bits( ) {	C	Descriptor
rbsp_stop_one_bit /* equal to 1 */	All	f(1)
while(!byte_aligned( ))		
rbsp_alignment_zero_bit /* equal to 0 */	All	f(1)
}		

7.3.3 Slice header syntax

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type = = 5)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type = = 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0 **]**	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1 **]**	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type = = B)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type = = P | | slice_type = = SP | | slice_type = = B) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type = = B)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering( )	2	
if((weighted_pred_flag && (slice_type = = P | | slice_type = = SP)) | | (weighted_bipred_idc = = 1 && slice_type = = B))		
pred_weight_table( )	2	
if(nal_ref_idc != 0)		
dec_ref_pic_marking( )	2	
if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)		
cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type = = SP | | slice_type = = SI) {		
if(slice_type = = SP)		

sp_for_switch_flag	2	u(1)
slice_qs_delta	2	se(v)
}		
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
}		

7.3.3.1 Reference picture list reordering syntax

ref_pic_list_reordering( ) {	C	Descriptor
if(slice_type != I && slice_type != SI && slice_type != EI) { /* slice_type EI is specified in Annex G */		
ref_pic_list_reordering_flag_l0	2	u(1)
if(ref_pic_list_reordering_flag_l0)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 || reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
if(slice_type == B || slice_type == EB) { /* slice_type EB is specified in Annex G */		
ref_pic_list_reordering_flag_l1	2	u(1)
if(ref_pic_list_reordering_flag_l1)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 || reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
}		

7.3.3.2 Prediction weight table syntax

pred_weight_table( ) {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(chroma_format_idc != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i **]**	2	se(v)
luma_offset_l0[i **]**	2	se(v)
}		
if (chroma_format_idc != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j =0; j < 2; j++) {		
chroma_weight_l0[i **][** j **]**	2	se(v)
chroma_offset_l0[i **][** j **]**	2	se(v)
}		
}		
}		
if(slice_type == B || slice_type == EB) /* slice_type EB is specified in Annex G */		
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {		
luma_weight_l1_flag	2	u(1)
if(luma_weight_l1_flag) {		
luma_weight_l1[i **]**	2	se(v)
luma_offset_l1[i **]**	2	se(v)
}		
if(chroma_format_idc != 0) {		
chroma_weight_l1_flag	2	u(1)
if(chroma_weight_l1_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l1[i **][** j **]**	2	se(v)
chroma_offset_l1[i **][** j **]**	2	se(v)
}		
}		
}		
}		

7.3.3.3 Decoded reference picture marking syntax

dec_ref_pic_marking( ) {	C	Descriptor
if(nal_unit_type = = 5 || nal_unit_type = = 21) { /* nal_unit_type 21 is specified in Annex G */		
no_output_of_prior_pics_flag	2 | 5	u(1)
long_term_reference_flag	2 | 5	u(1)
} else {		
adaptive_ref_pic_marking_mode_flag	2 | 5	u(1)
if(adaptive_ref_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2 | 5	ue(v)
if(memory_management_control_operation = = 1 || memory_management_control_operation = = 3)		
difference_of_pic_nums_minus1	2 | 5	ue(v)
if(memory_management_control_operation = = 2)		
long_term_pic_num	2 | 5	ue(v)
if(memory_management_control_operation = = 3 || memory_management_control_operation = = 6)		
long_term_frame_idx	2 | 5	ue(v)
if(memory_management_control_operation = = 4)		
max_long_term_frame_idx_plus1	2 | 5	ue(v)
} while(memory_management_control_operation != 0)		
}		
}		

7.3.4 Slice data syntax

slice_data( ) {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I && slice_type != SI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i=0; i<mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && (CurrMbAddr % 2 == 0 || (CurrMbAddr % 2 == 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != I && slice_type != SI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && CurrMbAddr % 2 == 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	C	Descriptor
mb_type	2	ue(v) | ae(v)
if(mb_type = = I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	2	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i **]**	2	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i **]**	2	u(v)
} else {		
noSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) = = 4) {		
sub_mb_pred(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type[mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type = = I_NxN)		
transform_size_8x8_flag	2	u(1) | ae(v)
mb_pred(mb_type)	2	
}		
if(MbPartPredMode(mb_type, 0) != Intra_16x16) {		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && mb_type != I_NxN && noSubMbPartSizeLessThan8x8Flag && (mb_type != B_Direct_16x16 || direct_8x8_inference_flag))		
transform_size_8x8_flag	2	u(1) | ae(v)
}		
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
mb_qp_delta	2	se(v) | ae(v)
residual( )	3 | 4	
}		
}		
}		

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) = = Intra_4x4 || MbPartPredMode(mb_type, 0) = = Intra_8x8 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) = = Intra_4x4)		
for(luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag**[** luma4x4BlkIdx **]**)		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) = = Intra_8x8)		
for(luma8x8BlkIdx=0; luma8x8BlkIdx<4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(chroma_format_idc != 0)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

7.3.5.2 Sub-macroblock prediction syntax

sub_mb_pred(mb_type) {	C	Descriptor
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
}		

7.3.5.3 Residual data syntax

residual( ) {	C	Descriptor
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16DCLevel, 16)	3	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
if(!transform_size_8x8_flag | | !entropy_coding_mode_flag)		
for(i4x4 = 0; i4x4 < 4; i4x4++) { /* each 4x4 sub-block of block */		
if(CodedBlockPatternLuma & (1 << i8x8))		
if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16ACLevel[i8x8 * 4 + i4x4], 15)	3	
else		
residual_block(LumaLevel[i8x8 * 4 + i4x4], 16)	3 | 4	
else if(MbPartPredMode(mb_type, 0) = = Intra_16x16)		
for(i = 0; i < 15; i++)		
Intra16x16ACLevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
LumaLevel[i8x8 * 4 + i4x4][i] = 0		
if(!entropy_coding_mode_flag && transform_size_8x8_flag)		
for(i = 0; i < 16; i++)		
LumaLevel8x8[i8x8][4 * i + i4x4] = LumaLevel[i8x8 * 4 + i4x4][i]		
}		
else if(CodedBlockPatternLuma & (1 << i8x8))		
residual_block(LumaLevel8x8[i8x8], 64)	3 | 4	
else		
for(i = 0; i < 64; i++)		
LumaLevel8x8[i8x8][i] = 0		
if(chroma_format_idc != 0) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
if(CodedBlockPatternChroma & 3) /* chroma DC residual present */		
residual_block(ChromaDCLevel[iCbCr], 4 * NumC8x8)	3 | 4	
else		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[iCbCr][i] = 0		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
if(CodedBlockPatternChroma & 2) /* chroma AC residual present */		
residual_block(ChromaACLevel[iCbCr][i8x8*4+i4x4], 15)	3 | 4	
else		
for(i = 0; i < 15; i++)		
ChromaACLevel[iCbCr][i8x8*4+i4x4][i] = 0		

}		

7.3.5.3.1 Residual block CAVLC syntax

residual_block_cavlc(coeffLevel, maxNumCoeff) {	C	Descriptor
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
coeff_token	3 | 4	ce(v)
if(TotalCoeff(coeff_token) > 0) {		
if(TotalCoeff(coeff_token) > 10 && TrailingOnes(coeff_token) < 3)		
suffixLength = 1		
else		
suffixLength = 0		
for(i = 0; i < TotalCoeff(coeff_token); i++)		
if(i < TrailingOnes(coeff_token)) {		
trailing_ones_sign_flag	3 | 4	u(1)
level[i] = 1 – 2 * trailing_ones_sign_flag		
} else {		
level_prefix	3 | 4	ce(v)
levelCode = (Min(15, level_prefix) << suffixLength)		
if(suffixLength > 0 || level_prefix >= 14) {		
level_suffix	3 | 4	u(v)
levelCode += level_suffix		
}		
if(level_prefix > = 15 && suffixLength = = 0)		
levelCode += 15		
if(level_prefix > = 16)		
levelCode += (1 << (level_prefix – 3)) – 4096		
if(i = = TrailingOnes(coeff_token) && TrailingOnes(coeff_token) < 3)		
levelCode += 2		
if(levelCode % 2 = = 0)		
level[i] = (levelCode + 2) >> 1		
else		
level[i] = (–levelCode – 1) >> 1		
if(suffixLength = = 0)		
suffixLength = 1		
if(Abs(level[i]) > (3 << (suffixLength – 1)) && suffixLength < 6)		
suffixLength++		
}		
if(TotalCoeff(coeff_token) < maxNumCoeff) {		
total_zeros	3 | 4	ce(v)
zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for(i = 0; i < TotalCoeff(coeff_token) – 1; i++) {		
if(zerosLeft > 0) {		

run_before	3 | 4	ce(v)
run[i] = run_before		
} else		
run[i] = 0		
zerosLeft = zerosLeft – run[i]		
}		
run[TotalCoeff(coeff_token) – 1] = zerosLeft		
coeffNum = -1		
for(i = TotalCoeff(coeff_token) – 1; i >= 0; i--) {		
coeffNum += run[i] + 1		
coeffLevel[coeffNum] = level[i]		
}		
}		
}		

7.3.5.3.2 Residual block CABAC syntax

residual_block_cabac(coeffLevel, maxNumCoeff) {	**C**	**Descriptor**
if(maxNumCoeff = = 64)		
coded_block_flag = 1		
else		
coded_block_flag	3 | 4	ae(v)
if(coded_block_flag) {		
numCoeff = maxNumCoeff		
i = 0		
do {		
significant_coeff_flag[i **]**	3 | 4	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i **]**	3 | 4	ae(v)
if(last_significant_coeff_flag[i]) {		
numCoeff = i + 1		
for(j = numCoeff; j < maxNumCoeff; j++)		
coeffLevel[j] = 0		
}		
}		
i++		
} while(i < numCoeff - 1)		
coeff_abs_level_minus1[numCoeff - 1 **]**	3 | 4	ae(v)
coeff_sign_flag[numCoeff - 1 **]**	3 | 4	ae(v)
coeffLevel[numCoeff - 1] = (coeff_abs_level_minus1[numCoeff – 1] + 1) * (1 – 2 * coeff_sign_flag[numCoeff – 1])		
for(i = numCoeff - 2; i >= 0; i--)		
if(significant_coeff_flag[i]) {		
coeff_abs_level_minus1[i **]**	3 | 4	ae(v)
coeff_sign_flag[i **]**	3 | 4	ae(v)
coeffLevel[i] = (coeff_abs_level_minus1[i] + 1) * (1 – 2 * coeff_sign_flag[i])		

} else		
coeffLevel[i] = 0		
} else		
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
}		

7.4 Semantics

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this subclause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Recommendation | International Standard.

7.4.1 NAL unit semantics

NOTE 1 – The VCL is specified to efficiently represent the content of the video data. The NAL is specified to format that data and provide header information in a manner appropriate for conveyance on a variety of communication channels or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and byte stream is identical except that each NAL unit can be preceded by a start code prefix and extra padding bytes in the byte stream format.

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit. Some form of demarcation of NAL unit boundaries is necessary to enable inference of NumBytesInNALunit. One such demarcation method is specified in Annex B for the byte stream format. Other methods of demarcation may be specified outside of this Recommendation | International Standard.

forbidden_zero_bit shall be equal to 0.

nal_ref_idc not equal to 0 specifies that the content of the NAL unit contains a sequence parameter set or a picture parameter set or a slice of a reference picture or a slice data partition of a reference picture.

nal_ref_idc equal to 0 for a NAL unit containing a slice or slice data partition indicates that the slice or slice data partition is part of a non-reference picture.

nal_ref_idc shall not be equal to 0 for sequence parameter set or sequence parameter set extension or picture parameter set NAL units. When nal_ref_idc is equal to 0 for one slice or slice data partition NAL unit of a particular picture, it shall be equal to 0 for all slice and slice data partition NAL units of the picture.

nal_ref_idc shall not be equal to 0 for IDR NAL units, i.e., NAL units with nal_unit_type equal to 5.

nal_ref_idc shall be equal to 0 for all NAL units having nal_unit_type equal to 6, 9, 10, 11, or 12.

nal_unit_type specifies the type of RBSP data structure contained in the NAL unit as specified in Table 7-1. VCL NAL units are specified as those NAL units having nal_unit_type equal to 1 to 5, inclusive, or equal to 20 to 21, inclusive. All remaining NAL units are called non-VCL NAL units. NAL units having nal_unit_type equal to 20 to 21 are specified in Annex G.

The column marked "C" in Table 7-1 lists the categories of the syntax elements that may be present in the NAL unit. In addition, syntax elements with syntax category "All" may be present, as determined by the syntax and semantics of the RBSP data structure. The presence or absence of any syntax elements of a particular listed category is determined from the syntax and semantics of the associated RBSP data structure. nal_unit_type shall not be equal to 3 or 4 unless at least one syntax element is present in the RBSP data structure having a syntax element category value equal to the value of nal_unit_type and not categorized as "All".

Table 7-1 – NAL unit type codes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13	Sequence parameter set extension seq_parameter_set_extension_rbsp( )	10
14..18	Reserved	
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp( )	2, 3, 4
20	Coded slice of a non-IDR picture in scalable extension slice_layer_in_scalable_extension_rbsp( )	2, 3, 4
21	Coded slice of an IDR picture in scalable extension slice_layer_in_scalable_extension_rbsp( )	2, 3
~~20~~22..23	Reserved	
24..31	Unspecified	

NAL units having nal_unit_type equal to 13 and 19 may be discarded by decoders without affecting the decoding process for NAL units having nal_unit_type not equal to 13 or 19 and without affecting conformance to this Recommendation | International Standard.

NAL units that use nal_unit_type equal to 0 or in the range of 24..31, inclusive, shall not affect the decoding process specified in this Recommendation | International Standard.

NOTE 2 – NAL unit types 0 and 24..31 may be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this Recommendation | International Standard.

Decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of nal_unit_type.

NOTE 3 – This requirement allows future definition of compatible extensions to this Recommendation | International Standard.

In the text, coded slice NAL unit collectively refers to a coded slice of a non-IDR picture NAL unit or to a coded slice of an IDR picture NAL unit.

When the value of nal_unit_type is equal to 5 for a NAL unit containing a slice of a coded picture, the value of nal_unit_type shall be 5 in all other VCL NAL units of the same coded picture. Such a picture is referred to as an IDR picture.

NOTE 4 – Slice data partitioning cannot be used for IDR pictures.

NAL units having nal_unit_type equal to 20 and 21 are specified in Annex G.

rbsp_byte[i **]** is the i-th byte of an RBSP. An RBSP is specified as an ordered sequence of bytes as follows.

The RBSP contains an SODB as follows.

– If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.

– Otherwise, the RBSP contains the SODB as follows.

1) The first byte of the RBSP contains the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP shall contain the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.

2) rbsp_trailing_bits( ) are present after the SODB as follows:

   i) The first (most significant, left-most) bits of the final RBSP byte contains the remaining bits of the SODB, (if any)

   ii) The next bit consists of a single rbsp_stop_one_bit equal to 1, and

   iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more rbsp_alignment_zero_bit is present to result in byte alignment.

3) One or more cabac_zero_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp_trailing_bits( ) at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "_rbsp" suffix. These structures shall be carried within NAL units as the content of the rbsp_byte[i] data bytes. The association of the RBSP syntax structures to the NAL units shall be as specified in Table 7-1.

NOTE 5 – When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the rbsp_stop_one_bit, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

emulation_prevention_three_byte is a byte equal to 0x03. When an emulation_prevention_three_byte is present in the NAL unit, it shall be discarded by the decoding process.

The last byte of the NAL unit shall not be equal to 0x00.

Within the NAL unit, the following three-byte sequences shall not occur at any byte-aligned position:

– 0x000000

– 0x000001

– 0x000002

Within the NAL unit, any four-byte sequence that starts with 0x000003 other than the following sequences shall not occur at any byte-aligned position:

– 0x00000300

– 0x00000301

– 0x00000302

– 0x00000303

NOTE 6 – When nal_unit_type is equal to 0, particular care must be exercised in the design of encoders to avoid the presence of the above-listed three-byte and four-byte patterns at the beginning of the NAL unit syntax structure, as the syntax element emulation_prevention_three_byte cannot be the third byte of a NAL unit.

7.4.1.1 Encapsulation of an SODB within an RBSP (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

The form of encapsulation of an SODB within an RBSP and the use of the emulation_prevention_three_byte for encapsulation of an RBSP within a NAL unit is specified for the following purposes:

– to prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit,

– to enable identification of the end of the SODB within the NAL unit by searching the RBSP for the rbsp_stop_one_bit starting at the end of the RBSP, and

– to enable a NAL unit to have a size larger than that of the SODB under some circumstances (using one or more cabac_zero_word).

The encoder can produce a NAL unit from an RBSP by the following procedure:

The RBSP data is searched for byte-aligned bits of the following binary patterns:

'00000000 00000000 000000xx' (where xx represents any 2 bit pattern: 00, 01, 10, or 11),

and a byte equal to 0x03 is inserted to replace these bit patterns with the patterns

'00000000 00000000 00000011 000000xx',

and finally, when the last byte of the RBSP data is equal to 0x00 (which can only occur when the RBSP ends in a cabac_zero_word), a final byte equal to 0x03 is appended to the end of the data.

The resulting sequence of bytes is then prefixed with the first byte of the NAL unit containing the indication of the type of RBSP data structure it contains. This results in the construction of the entire NAL unit.

This process can allow any SODB to be represented in a NAL unit while ensuring that

– no byte-aligned start code prefix is emulated within the NAL unit, and

– no sequence of 8 zero-valued bits followed by a start code prefix, regardless of byte-alignment, is emulated within the NAL unit.

7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences

This subclause specifies constraints on the order of NAL units in the bitstream. Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclauses 7.3, D.1, and E.1 specifies the decoding order of syntax elements. Decoders conforming to this Recommendation | International Standard shall be capable of receiving NAL units and their syntax elements in decoding order.

7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units or coded slice data partition A NAL units of one or more coded pictures. Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular picture parameter set RBSP results in the deactivation of the previously-active picture parameter set RBSP (if any).

When a picture parameter set RBSP (with a particular value of pic_parameter_set_id) is not active and it is referred to by a coded slice NAL unit or coded slice data partition A NAL unit (using that value of pic_parameter_set_id), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of pic_parameter_set_id, shall be available to the decoding process prior to its activation.

Any picture parameter set NAL unit containing the value of pic_parameter_set_id for the active picture parameter set RBSP shall have the same content as that of the active picture parameter set RBSP unless it follows the last VCL NAL unit of a coded picture and precedes the first VCL NAL unit of another coded picture.

A sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more SEI NAL units containing a buffering period SEI message. Each sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one sequence parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular sequence parameter set RBSP results in the deactivation of the previously-active sequence parameter set RBSP (if any).

When a sequence parameter set RBSP (with a particular value of seq_parameter_set_id) is not already active and it is referred to by activation of a picture parameter set RBSP (using that value of seq_parameter_set_id) or is referred to by an SEI NAL unit containing a buffering period SEI message (using that value of seq_parameter_set_id), it is activated. This sequence parameter set RBSP is called the active sequence parameter set RBSP until it is deactivated by the activation of another sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of seq_parameter_set_id, shall be available to the decoding process prior to its activation. An activated sequence parameter set RBSP shall remain active for the entire coded video sequence.

NOTE 2 – Because an IDR access unit begins a new coded video sequence and an activated sequence parameter set RBSP must remain active for the entire coded video sequence, a sequence parameter set RBSP can only be activated by a buffering period SEI message when the buffering period SEI message is part of an IDR access unit.

Any sequence parameter set NAL unit containing the value of seq_parameter_set_id for the active sequence parameter set RBSP shall have the same content as that of the active sequence parameter set RBSP unless it follows the last access unit of a coded video sequence and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

NOTE 3 – If picture parameter set RBSP or sequence parameter set RBSP are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the picture parameter set RBSP or sequence parameter set RBSP, respectively. Otherwise (picture parameter set RBSP or sequence parameter set RBSP are conveyed by other means not specified in this Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

When present, a sequence parameter set extension RBSP includes parameters having a similar function to those of a sequence parameter set RBSP. For purposes of establishing constraints on the syntax elements of the sequence parameter set extension RBSP and for purposes of determining activation of a sequence parameter set extension RBSP, the sequence parameter set extension RBSP shall be considered part of the preceding sequence parameter set RBSP with the same value of seq_parameter_set_id. When a sequence parameter set RBSP is present that is not followed by a sequence parameter set extension RBSP with the same value of seq_parameter_set_id prior to the activation of the sequence parameter set RBSP, the sequence parameter set extension RBSP and its syntax elements shall be considered not present for the active sequence parameter set RBSP.

All constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active sequence parameter set and the active picture parameter set. If any sequence parameter set RBSP is present that is not activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream. If any picture parameter set RBSP is present that is not ever activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream.

During operation of the decoding process (see clause 8), the values of parameters of the active picture parameter set and the active sequence parameter set shall be considered in effect. For interpretation of SEI messages, the values of the parameters of the picture parameter set and sequence parameter set that are active for the operation of the decoding process for the VCL NAL units of the primary coded picture in the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

7.4.1.2.2 Order of access units and association to coded video sequences

A bitstream conforming to this Recommendation | International Standard consists of one or more coded video sequences.

A coded video sequence consists of one or more access units. The order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.2.3.

The first access unit of each coded video sequence is an IDR access unit. All subsequent access units in the coded video sequence are non-IDR access units.

The values of picture order count for the coded pictures in consecutive access units in decoding order containing non-reference pictures shall be non-decreasing.

When present, an access unit following an access unit that contains an end of sequence NAL unit shall be an IDR access unit.

When an SEI NAL unit contains data that pertain to more than one access unit (for example, when the SEI NAL unit has a coded video sequence as its scope), it shall be contained in the first access unit to which it applies.

When an end of stream NAL unit is present in an access unit, this access unit shall be the last access unit in the bitstream and the end of stream NAL unit shall be the last NAL unit in that access unit.

7.4.1.2.3 Order of NAL units and coded pictures and association to access units

An access unit consists of one primary coded picture, zero or more corresponding redundant coded pictures, and zero or more non-VCL NAL units. The association of VCL NAL units to primary or redundant coded pictures is described in subclause 7.4.1.2.5.

The first access unit in the bitstream starts with the first NAL unit of the bitstream.

The first of any of the following NAL units after the last VCL NAL unit of a primary coded picture specifies the start of a new access unit.

– access unit delimiter NAL unit (when present)

– sequence parameter set NAL unit (when present)

– picture parameter set NAL unit (when present)

– SEI NAL unit (when present)

– NAL units with nal_unit_type in the range of 14 to 18, inclusive

– first VCL NAL unit of a primary coded picture (always present)

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause 7.4.1.2.4.

The following constraints shall be obeyed by the order of the coded pictures and non-VCL NAL units within an access unit.

– When an access unit delimiter NAL unit is present, it shall be the first NAL unit. There shall be at most one access unit delimiter NAL unit in any access unit.

– When any SEI NAL units are present, they shall precede the primary coded picture.

– When an SEI NAL unit containing a buffering period SEI message is present, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit

– The primary coded picture shall precede the corresponding redundant coded pictures.

– When redundant coded pictures are present, they shall be ordered in ascending order of the value of redundant_pic_cnt.

– When a sequence parameter set extension NAL unit is present, it shall be the next NAL unit after a sequence parameter set NAL unit having the same value of seq_parameter_set_id as in the sequence parameter set extension NAL unit.

– When one or more coded slice of an auxiliary coded picture without partitioning NAL units is present, they shall follow the primary coded picture and all redundant coded pictures (if any).

– When an end of sequence NAL unit is present, it shall follow the primary coded picture and all redundant coded pictures (if any) and all coded slice of an auxiliary coded picture without partitioning NAL units (if any).

– When an end of stream NAL unit is present, it shall be the last NAL unit.

– NAL units having nal_unit_type equal to 0, 12, or in the range of 20 to 31, inclusive, shall not precede the first VCL NAL unit of the primary coded picture.

NOTE 1 – Sequence parameter set NAL units or picture parameter set NAL units may be present in an access unit, but cannot follow the last VCL NAL unit of the primary coded picture within the access unit, as this condition would specify the start of a new access unit.

NOTE 2 – When a NAL unit having nal_unit_type equal to 7 or 8 is present in an access unit, it may or may not be referred to in the coded pictures of the access unit in which it is present, and may be referred to in coded pictures of subsequent access units.

The structure of access units not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 22 ~~0~~to 31, inclusive, is shown in Figure 7-1. The NAL units labelled as "coded picture (not primary)" refer to any combination of redundant coded pictures, auxiliary coded pictures, or enhancement coded pictures. Enhancement coded pictures are specified in Annex G.

Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of ~~20~~ 22 to 31, inclusive

7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture.

Any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the current access unit shall be different from any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the previous access unit in one or more of the following ways.

– frame_num differs in value. The value of frame_num used to test this condition is the value of frame_num that appears in the syntax of the slice header, regardless of whether that value is inferred to have been equal to 0 for subsequent use in the decoding process due to the presence of memory_management_control_operation equal to 5.

NOTE 1 – A consequence of the above statement is that a primary coded picture having frame_num equal to 1 cannot contain a memory_management_control_operation equal to 5 unless some other condition listed below is fulfilled for the next primary coded picture that follows after it (if any).

– pic_parameter_set_id differs in value.

– field_pic_flag differs in value.

– bottom_field_flag is present in both and differs in value.

– nal_ref_idc differs in value with one of the nal_ref_idc values being equal to 0.

– pic_order_cnt_type is equal to 0 for both and either pic_order_cnt_lsb differs in value, or delta_pic_order_cnt_bottom differs in value.

– pic_order_cnt_type is equal to 1 for both and either delta_pic_order_cnt[0] differs in value, or delta_pic_order_cnt[1] differs in value.

– nal_unit_type differs in value with one of the nal_unit_type values being equal to 5.

– nal_unit_type is equal to 5 for both and idr_pic_id differs in value.

NOTE 2 – Some of the VCL NAL units in redundant coded pictures or some non-VCL NAL units (e.g. an access unit delimiter NAL unit) may also be used for the detection of the boundary between access units, and may therefore aid in the detection of the start of a new primary coded picture.

7.4.1.2.5 Order of VCL NAL units and association to coded pictures

Each VCL NAL unit is part of a coded picture.

The order of the VCL NAL units within a coded IDR picture is constrained as follows.

– If arbitrary slice order is allowed as specified in Annex A, coded slice of an IDR picture NAL units may have any order relative to each other.

– Otherwise (arbitrary slice order is not allowed), the order of coded slice of an IDR picture NAL units shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit.

The order of the VCL NAL units within a coded non-IDR picture is constrained as follows.

– If arbitrary slice order is allowed as specified in Annex A, coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units may have any order relative to each other. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall precede any present coded slice data partition C NAL unit with the same value of slice_id. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall precede any present coded slice data partition C NAL unit with the same value of slice_id.

– Otherwise (arbitrary slice order is not allowed), the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition B NAL unit with the same value of slice_id. A coded slice data partition A NAL unit with a particular value of slice_id shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id, when a coded slice data partition B NAL unit with the same value of slice_id is not present. When a coded slice data partition B NAL unit with a particular value of slice_id is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of slice_id.

NAL units having nal_unit_type equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having nal_unit_type in the range of ~~20~~ 22 to 23, inclusive, which are reserved, shall not precede the first VCL NAL unit of the primary coded picture within the access unit (when specified in the future by ITU-T | ISO/IEC).

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

profile_idc and **level_idc** indicate the profile and level to which the bitstream conforms, as specified in Annex A.

constraint_set0_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.1. constraint_set0_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.1.

constraint_set1_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.2. constraint_set1_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.2.

constraint_set2_flag equal to 1 indicates that the bitstream obeys all constraints specified in subclause A.2.3. constraint_set2_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in subclause A.2.3.

NOTE 1 – When one or more than one of constraint_set0_flag, constraint_set1_flag, or constraint_set2_flag are equal to 1, the bitstream must obey the constraints of all of the indicated subclauses of subclause A.2. When profile_idc is equal to 100, 110, 122, or 144, the values of constraint_set0_flag, constraint_set1_flag, and constraint_set2_flag must all be equal to 0.

constraint_set3_flag indicates the following.

– If profile_idc is equal to 66, 77, or 88 and level_idc is equal to 11, constraint_set3_flag equal to 1 indicates that the bitstream obeys all constraints specified in Annex A for level 1b and constraint_set3_flag equal to 0 indicates that the bitstream may or may not obey all constraints specified in Annex A for level 1b.

– Otherwise (profile_idc is equal to 100, 110, 122, or 144 or level_idc is not equal to 11), the value of 1 for constraint_set3_flag is reserved for future use by ITU-T | ISO/IEC. constraint_set3_flag shall be equal to 0 in bitstreams conforming to this Recommendation | International Standard when profile_idc is equal to 100, 110, 122, or 144 or level_idc is not equal to 11. Decoders conforming to this Recommendation | International Standard shall ignore the value of constraint_set3_flag when profile_idc is equal to 100, 110, 122, or 144 or level_idc is not equal to 11.

reserved_zero_4bits shall be equal to 0. Other values of reserved_zero_4bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_4bits.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

NOTE 2 – When feasible, encoders should use distinct values of seq_parameter_set_id when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of seq_parameter_set_id.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in subclause 6.2. The value of chroma_format_idc shall be in the range of 0 to 3, inclusive. When chroma_format_idc is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

residual_colour_transform_flag equal to 1 specifies that the residual colour transform is applied as specified in subclause 8.5. residual_colour_transform_flag equal to 0 specifies that the residual colour transform is not applied. When residual_colour_transform_flag is not present, it shall be inferred to be equal to 0.

bit_depth_luma_minus8 specifies the bit depth of the samples of the luma array and the value of the luma quantisation parameter range offset $QpBdOffset_Y$, as specified by

$$BitDepth_Y = 8 + bit_depth_luma_minus8 \tag{7-1}$$

$$QpBdOffset_Y = 6 * bit_depth_luma_minus8 \tag{7-2}$$

When bit_depth_luma_minus8 is not present, it shall be inferred to be equal to 0. bit_depth_luma_minus8 shall be in the range of 0 to 4, inclusive.

bit_depth_chroma_minus8 specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantisation parameter range offset $QpBdOffset_C$, as specified by

$$BitDepth_C = 8 + bit_depth_chroma_minus8 \tag{7-3}$$

$$QpBdOffset_C = 6 * (bit_depth_chroma_minus8 + residual_colour_transform_flag) \quad (7\text{-}4)$$

When bit_depth_chroma_minus8 is not present, it shall be inferred to be equal to 0. bit_depth_chroma_minus8 shall be in the range of 0 to 4, inclusive.

The variable RawMbBits is derived as

$$RawMbBits = 256 * BitDepth_Y + 2 * MbWidthC * MbHeightC * BitDepth_C \quad (7\text{-}5)$$

qpprime_y_zero_transform_bypass_flag equal to 1 specifies that, when QP'_Y is equal to 0, a transform bypass operation for the transform coefficient decoding process and picture construction process prior to deblocking filter process as specified in subclause 8.5 shall be applied. qpprime_y_zero_transform_bypass_flag equal to 0 specifies that the transform coefficient decoding process and picture construction process prior to deblocking filter process shall not use the transform bypass operation. When qpprime_y_zero_transform_bypass_flag is not present, it shall be inferred to be equal to 0.

seq_scaling_matrix_present_flag equal to 1 specifies that the flags seq_scaling_list_present_flag[i] for i = 0..7 are present. seq_scaling_matrix_present_flag equal to 0 specifies that these flags are not present and the sequence-level scaling list specified by Flat_4x4_16 shall be inferred for i = 0..5 and the sequence-level scaling list specified by Flat_8x8_16 shall be inferred for i = 6..7. When seq_scaling_matrix_present_flag is not present, it shall be inferred to be equal to 0.

The scaling lists Flat_4x4_16 and Flat_8x8_16 are specified as follows:

$$Flat_4x4_16[i] = 16, \quad \text{with } i = 0..15, \quad (7\text{-}6)$$

$$Flat_8x8_16[i] = 16, \quad \text{with } i = 0..63. \quad (7\text{-}7)$$

seq_scaling_list_present_flag[i **]** equal to 1 specifies that the syntax structure for scaling list i is present in the sequence parameter set. seq_scaling_list_present_flag[i] equal to 0 specifies that the syntax structure for scaling list i is not present in the sequence parameter set and the scaling list fall-back rule set A specified in Table 7-2 shall be used to infer the sequence-level scaling list for index i.

Table 7-2 – Assignment of mnemonic names to scaling list indices and specification of fall-back rule

Value of scaling list index	Mnemonic name	Block size	MB prediction type	Component	Scaling list fall-back rule set A	Scaling list fall-back rule set B	Default scaling list
0	Sl_4x4_Intra_Y	4x4	Intra	Y	default scaling list	sequence-level scaling list	Default_4x4_Intra
1	Sl_4x4_Intra_Cb	4x4	Intra	Cb	scaling list for i = 0	scaling list for i = 0	Default_4x4_Intra
2	Sl_4x4_Intra_Cr	4x4	Intra	Cr	scaling list for i = 1	scaling list for i = 1	Default_4x4_Intra
3	Sl_4x4_Inter_Y	4x4	Inter	Y	default scaling list	sequence-level scaling list	Default_4x4_Inter
4	Sl_4x4_Inter_Cb	4x4	Inter	Cb	scaling list for i = 3	scaling list for i = 3	Default_4x4_Inter
5	Sl_4x4_Inter_Cr	4x4	Inter	Cr	scaling list for i = 4	scaling list for i = 4	Default_4x4_Inter
6	Sl_8x8_Intra_Y	8x8	Intra	Y	default scaling list	sequence-level scaling list	Default_8x8_Intra
7	Sl_8x8_Inter_Y	8x8	Inter	Y	default scaling list	sequence-level scaling list	Default_8x8_Inter

Table 7-3 specifies the default scaling lists Default_4x4_Intra and Default_4x4_Inter. Table 7-4 specifies the default scaling lists Default_8x8_Intra and Default_8x8_Inter.

Table 7-3 – Specification of default scaling lists Default_4x4_Intra and Default_4x4_Inter

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Default_4x4_Intra[idx]	6	13	13	20	20	20	28	28	28	28	32	32	32	37	37	42
Default_4x4_Inter[idx]	10	14	14	20	20	20	24	24	24	24	27	27	27	30	30	34

Table 7-4 – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Default_8x8_Intra[idx]	6	10	10	13	11	13	16	16	16	16	18	18	18	18	18	23
Default_8x8_Inter[idx]	9	13	13	15	13	15	17	17	17	17	19	19	19	19	19	21

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Default_8x8_Intra[idx]	23	23	23	23	23	25	25	25	25	25	25	25	27	27	27	27
Default_8x8_Inter[idx]	21	21	21	21	21	22	22	22	22	22	22	22	24	24	24	24

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Default_8x8_Intra[idx]	27	27	27	27	29	29	29	29	29	29	29	31	31	31	31	31
Default_8x8_Inter[idx]	24	24	24	24	25	25	25	25	25	25	25	27	27	27	27	27

Table 7-4 (concluded) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Default_8x8_Intra[idx]	31	33	33	33	33	33	36	36	36	36	38	38	38	40	40	42
Default_8x8_Inter[idx]	27	28	28	28	28	28	30	30	30	30	32	32	32	33	33	35

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{log2_max_frame_num_minus4} + 4)} \tag{7-8}$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in subclause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following

– an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture;

– two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture;

– an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in subclause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{log2_max_pic_order_cnt_lsb_minus4} + 4)} \tag{7-9}$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

delta_pic_order_always_zero_flag equal to 1 specifies that delta_pic_order_cnt[0] and delta_pic_order_cnt[1] are not present in the slice headers of the sequence and shall be inferred to be equal to 0. delta_pic_order_always_zero_flag equal to 0 specifies that delta_pic_order_cnt[0] is present in the slice headers of the sequence and delta_pic_order_cnt[1] may be present in the slice headers of the sequence.

offset_for_non_ref_pic is used to calculate the picture order count of a non-reference picture as specified in 8.2.1. The value of offset_for_non_ref_pic shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

offset_for_top_to_bottom_field is used to calculate the picture order count of a bottom field as specified in subclause 8.2.1. The value of offset_for_top_to_bottom_field shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

num_ref_frames_in_pic_order_cnt_cycle is used in the decoding process for picture order count as specified in subclause 8.2.1. The value of num_ref_frames_in_pic_order_cnt_cycle shall be in the range of 0 to 255, inclusive.

offset_for_ref_frame[i **]** is an element of a list of num_ref_frames_in_pic_order_cnt_cycle values used in the decoding process for picture order count as specified in subclause 8.2.1. The value of offset_for_ref_frame[i] shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any picture in the sequence. num_ref_frames also determines the size of the sliding window operation as specified in subclause 8.2.5.3. The value of num_ref_frames shall be in the range of 0 to MaxDpbSize (as specified in subclause A.1.1 or A.3.1), inclusive.

gaps_in_frame_num_value_allowed_flag specifies the allowed values of frame_num as specified in subclause 7.4.3 and the decoding process in case of an inferred gap between values of frame_num as specified in subclause 8.2.5.2.

pic_width_in_mbs_minus1 plus 1 specifies the width of each decoded picture in units of macroblocks.

The variable for the picture width in units of macroblocks is derived as follows

$$\text{PicWidthInMbs} = \text{pic_width_in_mbs_minus1} + 1 \tag{7-10}$$

The variable for picture width for the luma component is derived as follows

$$\text{PicWidthInSamples}_L = \text{PicWidthInMbs} * 16 \tag{7-11}$$

The variable for picture width for the chroma components is derived as follows

$$\text{PicWidthInSamples}_C = \text{PicWidthInMbs} * \text{MbWidthC} \tag{7-12}$$

pic_height_in_map_units_minus1 plus 1 specifies the height in slice group map units of a decoded frame or field.

The variables PicHeightInMapUnits and PicSizeInMapUnits are derived as follows

$$\text{PicHeightInMapUnits} = \text{pic_height_in_map_units_minus1} + 1 \tag{7-13}$$

$$\text{PicSizeInMapUnits} = \text{PicWidthInMbs} * \text{PicHeightInMapUnits} \tag{7-14}$$

frame_mbs_only_flag equal to 0 specifies that coded pictures of the coded video sequence may either be coded fields or coded frames. frame_mbs_only_flag equal to 1 specifies that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.

The allowed range of values for pic_width_in_mbs_minus1, pic_height_in_map_units_minus1, and frame_mbs_only_flag is specified by constraints in Annex A.

Depending on frame_mbs_only_flag, semantics are assigned to pic_height_in_map_units_minus1 as follows.

– If frame_mbs_only_flag is equal to 0, pic_height_in_map_units_minus1 plus 1 is the height of a field in units of macroblocks.

– Otherwise (frame_mbs_only_flag is equal to 1), pic_height_in_map_units_minus1 plus 1 is the height of a frame in units of macroblocks.

The variable FrameHeightInMbs is derived as follows

$$\text{FrameHeightInMbs} = (2 - \text{frame_mbs_only_flag}) * \text{PicHeightInMapUnits} \quad (7\text{-}15)$$

mb_adaptive_frame_field_flag equal to 0 specifies no switching between frame and field macroblocks within a picture. mb_adaptive_frame_field_flag equal to 1 specifies the possible use of switching between frame and field macroblocks within frames. When mb_adaptive_frame_field_flag is not present, it shall be inferred to be equal to 0.

direct_8x8_inference_flag specifies the method used in the derivation process for luma motion vectors for B_Skip, B_Direct_16x16 and B_Direct_8x8 as specified in subclause 8.4.1.2. When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.

frame_cropping_flag equal to 1 specifies that the frame cropping offset parameters follow next in the sequence parameter set. frame_cropping_flag equal to 0 specifies that the frame cropping offset parameters are not present.

frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, frame_crop_bottom_offset specify the samples of the pictures in the coded video sequence that are output from the decoding process, in terms of a rectangular region specified in frame coordinates for output.

The variables CropUnitX and CropUnitY are derived as follows:

– If chroma_format_idc is equal to 0, CropUnitX and CropUnitY are derived as

$$\text{CropUnitX} = 1 \quad (7\text{-}16)$$
$$\text{CropUnitY} = 2 - \text{frame_mbs_only_flag} \quad (7\text{-}17)$$

– Otherwise (chroma_format_idc is equal to 1, 2, or 3), CropUnitX and CropUnitY are derived as

$$\text{CropUnitX} = \text{SubWidthC} \quad (7\text{-}18)$$
$$\text{CropUnitY} = \text{SubHeightC} * (2 - \text{frame_mbs_only_flag}) \quad (7\text{-}19)$$

The frame cropping rectangle contains luma samples with horizontal frame coordinates from CropUnitX * frame_crop_left_offset to $\text{PicWidthInSamples}_L$ – (CropUnitX * frame_crop_right_offset + 1) and vertical frame coordinates from CropUnitY * frame_crop_top_offset to (16 * FrameHeightInMbs) – (CropUnitY * frame_crop_bottom_offset + 1), inclusive. The value of frame_crop_left_offset shall be in the range of 0 to ($\text{PicWidthInSamples}_L$ / CropUnitX) – (frame_crop_right_offset + 1), inclusive; and the value of frame_crop_top_offset shall be in the range of 0 to (16 * FrameHeightInMbs / CropUnitY) – (frame_crop_bottom_offset + 1), inclusive.

When frame_cropping_flag is equal to 0, the values of frame_crop_left_offset, frame_crop_right_offset, frame_crop_top_offset, and frame_crop_bottom_offset shall be inferred to be equal to 0.

When chroma_format_idc is not equal to 0, the corresponding specified samples of the two chroma arrays are the samples having frame coordinates (x / SubWidthC, y / SubHeightC), where (x, y) are the frame coordinates of the specified luma samples.

For decoded fields, the specified samples of the decoded field are the samples that fall within the rectangle specified in frame coordinates.

vui_parameters_present_flag equal to 1 specifies that the vui_parameters( ) syntax structure as specified in Annex E is present. vui_parameters_present_flag equal to 0 specifies that the vui_parameters( ) syntax structure as specified in Annex E is not present.

7.4.2.1.1 Scaling list semantics

delta_scale is used to derive the j-th element of the scaling list for j in the range of 0 to sizeOfScalingList - 1, inclusive. The value of delta_scale shall be in the range of -128 to +127, inclusive.

When useDefaultScalingMatrixFlag is derived to be equal to 1, the scaling list shall be inferred to be equal to the default scaling list as specified in Table 7-2.

7.4.2.1.2 Sequence parameter set extension RBSP semantics

seq_parameter_set_id identifies the sequence parameter set associated with the sequence parameter set extension. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

aux_format_idc equal to 0 indicates that there are no auxiliary coded pictures in the coded video sequence. aux_format_idc equal to 1 indicates that exactly one auxiliary coded picture is present in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in each access unit should be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. aux_format_idc equal to 2 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in each access unit should not be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. aux_format_idc equal to 3 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that the usage of the auxiliary coded pictures is unspecified. The value of aux_format_idc shall be in the range of 0 to 3, inclusive. Values greater than 3 for aux_format_idc are reserved to indicate the presence of exactly one auxiliary coded picture in each access unit of the coded video sequence for purposes to be specified in the future by ITU-T | ISO/IEC. When aux_format_idc is not present, it shall be inferred to be equal to 0.

NOTE 1 – Decoders conforming to this Recommendation | International Standard are not required to decode auxiliary coded pictures.

bit_depth_aux_minus8 specifies the bit depth of the samples of the sample array of the auxiliary coded picture. bit_depth_aux_minus8 shall be in the range of 0 to 4, inclusive.

alpha_incr_flag equal to 0 indicates that the interpretation sample value for each decoded auxiliary coded picture sample value is equal to the decoded auxiliary coded picture sample value for purposes of alpha blending. alpha_incr_flag equal to 1 indicates that, for purposes of alpha blending, after decoding the auxiliary coded picture samples, any auxiliary coded picture sample value that is greater than Min(alpha_opaque_value, alpha_transparent_value) should be increased by one to obtain the interpretation sample value for the auxiliary coded picture sample, and any auxiliary coded picture sample value that is less than or equal to Min(alpha_opaque_value, alpha_transparent_value) should be used without alteration as the interpretation sample value for the decoded auxiliary coded picture sample value.

alpha_opaque_value specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered opaque for purposes of alpha blending. The number of bits used for the representation of the alpha_opaque_value syntax element is bit_depth_aux_minus8 + 9 bits.

alpha_transparent_value specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered transparent for purposes of alpha blending. The number of bits used for the representation of the alpha_transparent_value syntax element is bit_depth_aux_minus8 + 9 bits.

When alpha_incr_flag is equal to 1, alpha_transparent_value shall not be equal to alpha_opaque_value and Log2(Abs(alpha_opaque_value – alpha_transparent_value)) shall have an integer value. A value of alpha_transparent_value that is equal to alpha_opaque_value indicates that the auxiliary coded picture is not intended for alpha blending purposes.

NOTE 2 – For alpha blending purposes, alpha_opaque_value may be greater than alpha_transparent_value, or it may be less than alpha_transparent_value. Interpretation sample values should be clipped to the range of alpha_opaque_value to alpha_transparent_value, inclusive.

The decoding of the sequence parameter set extension and the decoding of auxiliary coded pictures is not required for conformance with this Recommendation | International Standard.

The syntax of each coded slice of an auxiliary coded picture shall obey the same constraints as a coded slice of a redundant picture, with the following differences of constraints.

- The following applies in regard to whether the primary coded picture is an IDR picture.
  - If the primary coded picture is an IDR picture, the auxiliary coded slice syntax shall correspond to that of a slice having nal_unit_type equal to 5 (a slice of an IDR picture);
  - Otherwise (the primary coded picture is not an IDR picture), the auxiliary coded slice syntax shall correspond to that of a slice having nal_unit_type equal to 1 (a slice of a non-IDR picture).
- The slices of an auxiliary coded picture (when present) shall contain all macroblocks corresponding to those of the primary coded picture.
- redundant_pic_cnt shall be equal to 0 in all auxiliary coded slices.

The (optional) decoding process for the decoding of auxiliary coded pictures is the same as if the auxiliary coded pictures were primary coded pictures in a separate coded video stream that differs from the primary coded pictures in the current coded video stream in the following ways.

– The IDR or non-IDR status of each auxiliary coded picture shall be inferred to be the same as the IDR or non-IDR status of the primary picture in the same access unit, rather than being inferred from the value of nal_ref_idc.

– The value of chroma_format_idc shall be inferred to be equal to 0 for the decoding of the auxiliary coded pictures.

– The value of bit_depth_luma_minus8 shall be inferred to be equal to bit_depth_aux_minus8 for the decoding of the auxiliary coded pictures.

NOTE 3 – Alpha blending composition is normally performed with a background picture B, a foreground picture F, and a decoded auxiliary coded picture A, all of the same size. Assume for purposes of example illustration that the chroma resolution of B and F have been upsampled to the same resolution as the luma. Denote corresponding samples of B, F and A by b, f and a, respectively. Denote luma and chroma samples by subscripts Y, Cb and Cr.

Define the variables alphaRange, alphaFwt and alphaBwt as follows:

alphaRange = Abs(alpha_opaque_value - alpha_transparent_value)

alphaFwt = Abs(a - alpha_transparent_value)

alphaBwt = Abs(a - alpha_opaque_value)

Then, in alpha blending composition, samples d of the displayed picture D may be calculated as

d_Y = (alphaFwt*f_Y + alphaBwt*b_Y + alphaRange/2) / alphaRange

d_{CB} = (alphaFwt*f_{CB} + alphaBwt*b_{CB} + alphaRange/2) / alphaRange

d_{CR} = (alphaFwt*f_{CR} + alphaBwt*b_{CR} + alphaRange/2) / alphaRange

The samples of pictures D, F and B could also represent red, green, and blue component values (see subclause E.2.1). Here we have assumed Y, Cb and Cr component values. Each component, e.g. Y, is assumed for purposes of example illustration above to have the same bit depth in each of the pictures D, F and B. However, different components, e.g. Y and Cb, need not have the same bit depth in this example.

When aux_format_idc is equal to 1, F would be the decoded picture obtained from the decoded luma and chroma, and A would be the decoded picture obtained from the decoded auxiliary coded picture. In this case, the indicated example alpha blending composition involves multiplying the samples of F by factors obtained from the samples of A.

A picture format that is useful for editing or direct viewing, and that is commonly used, is called pre-multiplied-black video. If the foreground picture was F, then the pre-multiplied-black video S is given by

s_Y = (alphaFwt*f_Y) / alphaRange

s_{CB} = (alphaFwt*f_{CB}) / alphaRange

s_{CR} = (alphaFwt*f_{CR}) / alphaRange

Pre-multiplied-black video has the characteristic that the picture S will appear correct if displayed against a black background. For a non-black background B, the composition of the displayed picture D may be calculated as

d_Y = s_Y + (alphaBwt*b_Y + alphaRange/2) / alphaRange

d_{CB} = s_{CB} + (alphaBwt*b_{CB} + alphaRange/2) / alphaRange

d_{CR} = s_{CR} + (alphaBwt*b_{CR} + alphaRange/2) / alphaRange

When aux_format_idc is equal to 2, S would be the decoded picture obtained from the decoded luma and chroma, and A would again be the decoded picture obtained from the decoded auxiliary coded picture. In this case, alpha blending composition does not involve multiplication of the samples of S by factors obtained from the samples of A.

additional_extension_flag equal to 0 indicates that no additional data follows within the sequence parameter set extension syntax structure prior to the RBSP trailing bits. The value of additional_extension_flag shall be equal to 0. The value of 1 for additional_extension_flag is reserved for future use by ITU-T | ISO/IEC. Decoders that conform to this Recommendation | International Standard shall ignore all data that follows the value of 1 for additional_extension_flag in a sequence parameter set extension NAL unit.

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id identifies the picture parameter set that is referred to in the slice header. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

seq_parameter_set_id refers to the active sequence parameter set. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

entropy_coding_mode_flag selects the entropy decoding method to be applied for the syntax elements for which two descriptors appear in the syntax tables as follows.

– If entropy_coding_mode_flag is equal to 0, the method specified by the left descriptor in the syntax table is applied (Exp-Golomb coded, see subclause 9.1 or CAVLC, see subclause 9.2).

– Otherwise (entropy_coding_mode_flag is equal to 1), the method specified by the right descriptor in the syntax table is applied (CABAC, see subclause 9.3).

pic_order_present_flag equal to 1 specifies that the picture order count related syntax elements are present in the slice headers as specified in subclause 7.3.3. pic_order_present_flag equal to 0 specifies that the picture order count related syntax elements are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When num_slice_groups_minus1 is equal to 0, all slices of the picture belong to the same slice group. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of slice_group_map_type shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more "foreground" slice groups and a "leftover" slice group.

slice_group_map_type values equal to 3, 4, and 5 specify changing slice groups. When num_slice_groups_minus1 is not equal to 1, slice_group_map_type shall not be equal to 3, 4, or 5.

slice_group_map_type equal to 6 specifies an explicit assignment of a slice group to each slice group map unit.

Slice group map units are specified as follows.

– If frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 1 and the coded picture is a frame, the slice group map units are macroblock pair units.

– Otherwise, if frame_mbs_only_flag is equal to 1 or a coded picture is a field, the slice group map units are units of macroblocks.

– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and the coded picture is a frame), the slice group map units are units of two macroblocks that are vertically contiguous as in a frame macroblock pair of an MBAFF frame.

run_length_minus1[i **]** is used to specify the number of consecutive slice group map units to be assigned to the i-th slice group in raster scan order of slice group map units. The value of run_length_minus1[i] shall be in the range of 0 to PicSizeInMapUnits - 1, inclusive.

top_left[i **]** and **bottom_right[** i **]** specify the top-left and bottom-right corners of a rectangle, respectively. top_left[i] and bottom_right[i] are slice group map unit positions in a raster scan of the picture for the slice group map units. For each rectangle i, all of the following constraints shall be obeyed by the values of the syntax elements top_left[i] and bottom_right[i]

– top_left[i] shall be less than or equal to bottom_right[i] and bottom_right[i] shall be less than PicSizeInMapUnits.

– (top_left[i] % PicWidthInMbs) shall be less than or equal to the value of (bottom_right[i] % PicWidthInMbs).

slice_group_change_direction_flag is used with slice_group_map_type to specify the refined map type when slice_group_map_type is 3, 4, or 5.

slice_group_change_rate_minus1 is used to specify the variable SliceGroupChangeRate. SliceGroupChangeRate specifies the multiple in number of slice group map units by which the size of a slice group can change from one picture to the next. The value of slice_group_change_rate_minus1 shall be in the range of 0 to PicSizeInMapUnits – 1, inclusive. The SliceGroupChangeRate variable is specified as follows:

$$\text{SliceGroupChangeRate} = \text{slice_group_change_rate_minus1} + 1 \qquad (7\text{-}20)$$

pic_size_in_map_units_minus1 is used to specify the number of slice group map units in the picture. pic_size_in_map_units_minus1 shall be equal to PicSizeInMapUnits - 1.

slice_group_id[i **]** identifies a slice group of the i-th slice group map unit in raster scan order. The size of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits. The value of slice_group_id[i] shall be in the range of 0 to num_slice_groups_minus1, inclusive.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode each slice of the picture in which list 0 prediction is used when num_ref_idx_active_override_flag is equal to 0 for the slice. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks. The value of num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively.

weighted_pred_flag equal to 0 specifies that weighted prediction shall not be applied to P and SP slices. weighted_pred_flag equal to 1 specifies that weighted prediction shall be applied to P and SP slices.

weighted_bipred_idc equal to 0 specifies that the default weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 1 specifies that explicit weighted prediction shall be applied to B slices. weighted_bipred_idc equal to 2 specifies that implicit weighted prediction shall be applied to B slices. The value of weighted_bipred_idc shall be in the range of 0 to 2, inclusive.

pic_init_qp_minus26 specifies the initial value minus 26 of $SliceQP_Y$ for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value of pic_init_qp_minus26 shall be in the range of $-(26 + QpBdOffset_Y)$ to +25, inclusive.

pic_init_qs_minus26 specifies the initial value minus 26 of $SliceQS_Y$ for all macroblocks in SP or SI slices. The initial value is modified at the slice layer when a non-zero value of slice_qs_delta is decoded. The value of pic_init_qs_minus26 shall be in the range of -26 to +25, inclusive.

chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cb chroma component. The value of chroma_qp_index_offset shall be in the range of -12 to +12, inclusive.

deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter is present in the slice header. deblocking_filter_control_present_flag equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter is not present in the slice headers and their inferred values are in effect.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring macroblocks coded using Inter macroblock prediction modes for the prediction of macroblocks coded using Intra macroblock prediction modes. constrained_intra_pred_flag equal to 1 specifies constrained intra prediction, in which case prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data and decoded samples from I or SI macroblock types.

redundant_pic_cnt_present_flag equal to 0 specifies that the redundant_pic_cnt syntax element is not present in slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set. redundant_pic_cnt_present_flag equal to 1 specifies that the redundant_pic_cnt syntax element is present in all slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set.

transform_8x8_mode_flag equal to 1 specifies that the 8x8 transform decoding process may be in use (see subclause 8.5). transform_8x8_mode_flag equal to 0 specifies that the 8x8 transform decoding process is not in use. When transform_8x8_mode_flag is not present, it shall be inferred to be 0.

pic_scaling_matrix_present_flag equal to 1 specifies that parameters are present to modify the scaling lists specified in the sequence parameter set. pic_scaling_matrix_present_flag equal to 0 specifies that the scaling lists used for the picture shall be inferred to be equal to those specified by the sequence parameter set. When pic_scaling_matrix_present_flag is not present, it shall be inferred to be equal to 0.

pic_scaling_list_present_flag[i **]** equal to 1 specifies that the scaling list syntax structure is present to specify the scaling list for index i. pic_scaling_list_present_flag[i] equal to 0 specifies that the syntax structure for scaling list i is not present in the picture parameter set and that depending on the value of seq_scaling_matrix_present_flag, the following applies.

– If seq_scaling_matrix_present_flag is equal to 0, the scaling list fall-back rule set A as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.

– Otherwise (seq_scaling_matrix_present_flag is equal to 1), the scaling list fall-back rule set B as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.

second_chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cr chroma component. The value of second_chroma_qp_index_offset shall be in the range of -12 to +12, inclusive.

When second_chroma_qp_index_offset is not present, it shall be inferred to be equal to chroma_qp_index_offset.

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode the samples of coded pictures from VCL NAL units.

7.4.2.3.1 Supplemental enhancement information message semantics

An SEI NAL unit contains one or more SEI messages. Each SEI message consists of the variables specifying the type payloadType and size payloadSize of the SEI payload. SEI payloads are specified in Annex D. The derived SEI payload size payloadSize is specified in bytes and shall be equal to the number of bytes in the SEI payload.

ff_byte is a byte equal to 0xFF identifying a need for a longer representation of the syntax structure that it is used within.

last_payload_type_byte is the last byte of the payload type of an SEI message.

last_payload_size_byte is the last byte of the size of an SEI message.

7.4.2.4 Access unit delimiter RBSP semantics

The access unit delimiter may be used to indicate the type of slices present in a primary coded picture and to simplify the detection of the boundary between access units. There is no normative decoding process associated with the access unit delimiter.

primary_pic_type indicates that the slice_type values for all slices of the primary coded picture are members of the set listed in Table 7-5 for the given value of primary_pic_type.

Table 7-5 – Meaning of primary_pic_type

primary_pic_type	slice_type values that may be present in the primary coded picture
0	I
1	I, P
2	I, P, B
3	SI
4	SI, SP
5	I, SI
6	I, SI, P, SP
7	I, SI, P, SP, B

7.4.2.5 End of sequence RBSP semantics

The end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) shall be an IDR access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty. No normative decoding process is specified for an end of sequence RBSP.

7.4.2.6 End of stream RBSP semantics

The end of stream RBSP indicates that no additional NAL units shall be present in the bitstream that are subsequent to the end of stream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of stream RBSP are empty. No normative decoding process is specified for an end of stream RBSP.

7.4.2.7 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte equal to 0xFF.

7.4.2.8 Slice layer without partitioning RBSP semantics

The slice layer without partitioning RBSP consists of a slice header and slice data.

7.4.2.9 Slice data partition RBSP semantics

7.4.2.9.1 Slice data partition A RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains all syntax elements of category 2.

Category 2 syntax elements include all syntax elements in the slice header and slice data syntax structures other than the syntax elements in the residual( ) syntax structure.

slice_id identifies the slice associated with the data partition. Each slice shall have a unique slice_id value within the coded picture that contains the slice. When arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture, in decoding order, shall have slice_id equal to 0 and the value of slice_id shall be incremented by one for each subsequent slice of the coded picture in decoding order.

The range of slice_id is specified as follows.

– If MbaffFrameFlag is equal to 0, slice_id shall be in the range of 0 to PicSizeInMbs - 1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1), slice_id shall be in the range of 0 to PicSizeInMbs / 2 - 1, inclusive.

7.4.2.9.2 Slice data partition B RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into one to three separate partitions. Slice data partition B contains all syntax elements of category 3.

Category 3 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types I and SI as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The redundant_pic_cnt shall be greater than 0 for coded slices and coded slice data partitions in redundant coded pictures. When redundant_pic_cnt is not present, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

The presence of a slice data partition B RBSP is specified as follows.

– If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 3 in the slice data for a slice, a slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

– Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 3 in the slice data for a slice), no slice data partition B RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.9.3 Slice data partition C RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Slice data partition C contains all syntax elements of category 4.

Category 4 syntax elements include all syntax elements in the residual( ) syntax structure and in syntax structures used within that syntax structure for collective macroblock types P and B as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

redundant_pic_cnt has the same semantics as specified in subclause 7.4.2.9.2.

The presence of a slice data partition C RBSP is specified as follows.

– If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 4 in the slice data for a slice, a slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

– Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 4 in the slice data for a slice), no slice data partition C RBSP shall be present having the same value of slice_id and redundant_pic_cnt as in the slice data partition A RBSP.

7.4.2.10 RBSP slice trailing bits semantics

cabac_zero_word is a byte-aligned sequence of two bytes equal to 0x0000.

Let NumBytesInVclNALunits be the sum of the values of NumBytesInNALunit for all VCL NAL units of a coded picture.

Let BinCountsInNALunits be the number of times that the parsing process function DecodeBin( ), specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of a coded picture. When entropy_coding_mode_flag is equal to 1, BinCountsInNALunits shall not exceed (32 ÷ 3) * NumBytesInVclNALunits + (RawMbBits * PicSizeInMbs) ÷ 32.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the slice layer NAL units can be met by inserting a number of cabac_zero_word syntax elements to increase the value of NumBytesInVclNALunits. Each cabac_zero_word is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an emulation_prevention_three_byte for each cabac_zero_word).

7.4.2.11 RBSP trailing bits semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements pic_parameter_set_id, frame_num, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], sp_for_switch_flag, and slice_group_change_cycle shall be the same in all slice headers of a coded picture.

first_mb_in_slice specifies the address of the first macroblock in the slice. When arbitrary slice order is not allowed as specified in Annex A, the value of first_mb_in_slice shall not be less than the value of first_mb_in_slice for any other slice of the current picture that precedes the current slice in decoding order.

The first macroblock address of the slice is derived as follows.

– If MbaffFrameFlag is equal to 0, first_mb_in_slice is the macroblock address of the first macroblock in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs - 1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1), first_mb_in_slice * 2 is the macroblock address of the first macroblock in the slice, which is the top macroblock of the first macroblock pair in the slice, and first_mb_in_slice shall be in the range of 0 to PicSizeInMbs / 2 - 1, inclusive.

slice_type specifies the coding type of the slice according to Table 7-6.

Table 7-6 – Name association to slice_type

slice_type	Name of slice_type
0	P (P slice)
1	B (B slice)
2	I (I slice)
3	SP (SP slice)
4	SI (SI slice)
5	P (P slice)
6	B (B slice)
7	I (I slice)
8	SP (SP slice)
9	SI (SI slice)

slice_type values in the range 5..9 specify, in addition to the coding type of the current slice, that all other slices of the current coded picture shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type – 5.

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2, 4, 7, or 9.

When num_ref_frames is equal to 0, slice_type shall be equal to 2, 4, 7, or 9.

pic_parameter_set_id specifies the picture parameter set in use. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows.

- If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.
- Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows.
  - If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.
  - Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows.

- If the current picture is an IDR picture, frame_num shall be equal to 0.
- Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true.
  - the current picture and the preceding reference picture belong to consecutive access units in decoding order
  - the current picture and the preceding reference picture are reference fields having opposite parity
  - one or more of the following conditions is true
    - the preceding reference picture is an IDR picture
    - the preceding reference picture includes a memory_management_control_operation syntax element equal to 5

      NOTE 1 – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture

When the value of frame_num is not equal to PrevRefFrameNum, the following applies.

- There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum  !=  frame_num )                    (7-21)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

- The value of frame_num is constrained as follows.

- If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.
- Otherwise (gaps_in_frame_num_value_allowed_flag is equal to 1), the following applies.
  - If frame_num is greater than PrevRefFrameNum, there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which either of the following conditions is true.
    - The value of frame_num for the non-reference picture is less than PrevRefFrameNum.
    - The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.
  - Otherwise (frame_num is less than PrevRefFrameNum), there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which both of the following conditions are true.
    - The value of frame_num for the non-reference picture is less than PrevRefFrameNum.
    - The value of frame_num for the non-reference picture is greater than the value of frame_num for the current picture.

A picture including a memory_management_control_operation equal to 5 shall have frame_num constraints as described above and, after the decoding of the current picture and the processing of the memory management control operations, the picture shall be inferred to have had frame_num equal to 0 for all subsequent use in the decoding process, except as specified in subclause 7.4.1.2.4.

NOTE 2 – When the primary coded picture is not an IDR picture and does not contain memory_management_control_operation syntax element equal to 5, the value of frame_num of a corresponding redundant coded picture is the same as the value of frame_num in the primary coded picture. Alternatively, the redundant coded picture includes a memory_management_control_operation syntax element equal to 5 and the corresponding primary coded picture is an IDR picture.

field_pic_flag equal to 1 specifies that the slice is a slice of a coded field. field_pic_flag equal to 0 specifies that the slice is a slice of a coded frame. When field_pic_flag is not present it shall be inferred to be equal to 0.

The variable MbaffFrameFlag is derived as follows.

MbaffFrameFlag = (mb_adaptive_frame_field_flag && !field_pic_flag) (7-22)

The variable for the picture height in units of macroblocks is derived as follows

PicHeightInMbs = FrameHeightInMbs / (1 + field_pic_flag) (7-23)

The variable for picture height for the luma component is derived as follows

$PicHeightInSamples_L$ = PicHeightInMbs * 16 (7-24)

The variable for picture height for the chroma component is derived as follows

$PicHeightInSamples_C$ = PicHeightInMbs * MbHeightC (7-25)

The variable PicSizeInMbs for the current picture is derived according to:

PicSizeInMbs = PicWidthInMbs * PicHeightInMbs (7-26)

The variable MaxPicNum is derived as follows.

- If field_pic_flag is equal to 0, MaxPicNum is set equal to MaxFrameNum.
- Otherwise (field_pic_flag is equal to 1), MaxPicNum is set equal to 2*MaxFrameNum.

The variable CurrPicNum is derived as follows.

– If field_pic_flag is equal to 0, CurrPicNum is set equal to frame_num.

– Otherwise (field_pic_flag is equal to 1), CurrPicNum is set equal to 2 * frame_num + 1.

bottom_field_flag equal to 1 specifies that the slice is part of a coded bottom field. bottom_field_flag equal to 0 specifies that the picture is a coded top field. When this syntax element is not present for the current slice, it shall be inferred to be equal to 0.

idr_pic_id identifies an IDR picture. The values of idr_pic_id in all the slices of an IDR picture shall remain unchanged. When two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the first such IDR access unit shall differ from the idr_pic_id in the second such IDR access unit. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the top field of a coded frame or for a coded field. The size of the pic_order_cnt_lsb syntax element is log2_max_pic_order_cnt_lsb_minus4 + 4 bits. The value of the pic_order_cnt_lsb shall be in the range of 0 to MaxPicOrderCntLsb – 1, inclusive.

delta_pic_order_cnt_bottom specifies the picture order count difference between the bottom field and the top field of a coded frame as follows.

– If the current picture includes a memory_management_control_operation equal to 5, the value of delta_pic_order_cnt_bottom shall be in the range of (1 – MaxPicOrderCntLsb) to 2^{31} - 1, inclusive.

– Otherwise (the current picture does not include a memory_management_control_operation equal to 5), the value of delta_pic_order_cnt_bottom shall be in the range of -2^{31} to 2^{31} - 1, inclusive.

When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[0 **]** specifies the picture order count difference from the expected picture order count for the top field of a coded frame or for a coded field as specified in subclause 8.2.1. The value of delta_pic_order_cnt[0] shall be in the range of -2^{31} to 2^{31} - 1, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[1 **]** specifies the picture order count difference from the expected picture order count for the bottom field of a coded frame specified in subclause 8.2.1. The value of delta_pic_order_cnt[1] shall be in the range of -2^{31} to 2^{31} - 1, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The value of redundant_pic_cnt shall be greater than 0 for coded slices or coded slice data partitions of a redundant coded picture. When redundant_pic_cnt is not present in the bitstream, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

NOTE 3 – Any area of the decoded primary picture and the corresponding area that would result from application of the decoding process specified in clause 8 for any redundant picture in the same access unit should be visually similar in appearance.

The value of pic_parameter_set_id in a coded slice or coded slice data partition of a redundant coded picture shall be such that the value of pic_order_present_flag in the picture parameter set in use in a redundant coded picture is equal to the value of pic_order_present_flag in the picture parameter set in use in the corresponding primary coded picture.

When present in the primary coded picture and any redundant coded picture, the following syntax elements shall have the same value: field_pic_flag, bottom_field_flag, and idr_pic_id.

When the value of nal_ref_idc in one VCL NAL unit of an access unit is equal to 0, the value of nal_ref_idc in all other VCL NAL units of the same access unit shall be equal to 0.

NOTE 4 – The above constraint also has the following implications. If the value of nal_ref_idc for the VCL NAL units of the primary coded picture is equal to 0, the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are equal to 0; otherwise (the value of nal_ref_idc for the VCL NAL units of the primary coded picture is greater than 0), the value of nal_ref_idc for the VCL NAL units of any corresponding redundant coded picture are also greater than 0.

The marking status of reference pictures and the value of frame_num after the decoded reference picture marking process as specified in subclause 8.2.5 is invoked for the primary coded picture or any redundant coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

NOTE 5 – The above constraint also has the following implications.

If a primary coded picture is not an IDR picture, the contents of the dec_ref_pic_marking( ) syntax structure must be identical in all slice headers of the primary coded picture and all redundant coded pictures corresponding to the primary coded picture.

Otherwise (a primary coded picture is an IDR picture), the following applies.

If a redundant coded picture corresponding to the primary coded picture is an IDR picture, the contents of the dec_ref_pic_marking( ) syntax structure must be identical in all slice headers of the primary coded picture and the redundant coded picture corresponding to the primary coded picture.

Otherwise (a redundant picture corresponding to the primary coded picture is not an IDR picture), all slice headers of the redundant picture must contain a dec_ref_pic_marking syntax( ) structure including a memory_management_control_operation syntax element equal to 5, and the following applies.

If the value of long_term_reference_flag in the primary coded picture is equal to 0, the dec_ref_pic_marking syntax structure of the redundant coded picture must not include a memory_management_control_operation syntax element equal to 6.

Otherwise (the value of long_term_reference_flag in the primary coded picture is equal to 1), the dec_ref_pic_marking syntax structure of the redundant coded picture must include memory_management_control_operation syntax elements equal to 5, 4, and 6 in decoding order, and the value of max_long_term_frame_idx_plus1 must be equal to 1, and the value of long_term_frame_idx must be equal to 0.

The values of TopFieldOrderCnt and BottomFieldOrderCnt (if applicable) that result after completion of the decoding process for any redundant coded picture or the primary coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

There is no required decoding process for a coded slice or coded slice data partition of a redundant coded picture. When the redundant_pic_cnt in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. However, a coded slice or coded slice data partition of any redundant coded picture shall obey the same constraints as a coded slice or coded slice data partition of a primary picture.

NOTE 6 – When some of the samples in the decoded primary picture cannot be correctly decoded due to errors or losses in transmission of the sequence and a coded redundant slice can be correctly decoded, the decoder should replace the samples of the decoded primary picture with the corresponding samples of the decoded redundant slice. When more than one redundant slice covers the relevant region of the primary picture, the redundant slice having the lowest value of redundant_pic_cnt should be used.

Redundant slices and slice data partitions having the same value of redundant_pic_cnt belong to the same redundant picture. Decoded slices within the same redundant picture need not cover the entire picture area and shall not overlap.

direct_spatial_mv_pred_flag specifies the method used in the decoding process to derive motion vectors and reference indices for inter prediction as follows.

– If direct_spatial_mv_pred_flag is equal to 1, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use spatial direct mode prediction as specified in subclause 8.4.1.2.2.

– Otherwise (direct_spatial_mv_pred_flag is equal to 0), the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in subclause 8.4.1.2 shall use temporal direct mode prediction as specified in subclause 8.4.1.2.3.

num_ref_idx_active_override_flag equal to 0 specifies that the values of the syntax elements num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 specified in the referred picture parameter set are in effect. num_ref_idx_active_override_flag equal to 1 specifies that the num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 specified in the referred picture parameter set are overridden for the current slice (and only for the current slice) by the following values in the slice header.

When the current slice is a P, SP, or B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l0_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

When the current slice is a B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l1_active_minus1 in the picture parameter set exceeds 15, num_ref_idx_active_override_flag shall be equal to 1.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

The range of num_ref_idx_l0_active_minus1 is specified as follows.

– If field_pic_flag is equal to 0, num_ref_idx_l0_active_minus1 shall be in the range of 0 to 15, inclusive. When MbaffFrameFlag is equal to 1, num_ref_idx_l0_active_minus1 is the maximum index value for the decoding of frame macroblocks and 2 * num_ref_idx_l0_active_minus1 + 1 is the maximum index value for the decoding of field macroblocks.

– Otherwise (field_pic_flag is equal to 1), num_ref_idx_l0_active_minus1 shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as num_ref_idx_l0_active_minus1 with l0 and list 0 replaced by l1 and list 1, respectively.

cabac_init_idc specifies the index for determining the initialisation table used in the initialisation process for context variables. The value of cabac_init_idc shall be in the range of 0 to 2, inclusive.

slice_qp_delta specifies the initial value of QP_Y to be used for all the macroblocks in the slice until modified by the value of mb_qp_delta in the macroblock layer. The initial QP_Y quantisation parameter for the slice is computed as:

$$SliceQP_Y = 26 + pic_init_qp_minus26 + slice_qp_delta \quad (7\text{-}27)$$

The value of slice_qp_delta shall be limited such that $SliceQP_Y$ is in the range of $-QpBdOffset_Y$ to +51, inclusive.

sp_for_switch_flag specifies the decoding process to be used to decode P macroblocks in an SP slice as follows.

– If sp_for_switch_flag is equal to 0, the P macroblocks in the SP slice shall be decoded using the SP decoding process for non-switching pictures as specified in subclause 8.6.1.

– Otherwise (sp_for_switch_flag is equal to 1), the P macroblocks in the SP slice shall be decoded using the SP and SI decoding process for switching pictures as specified in subclause 8.6.2.

slice_qs_delta specifies the value of QS_Y for all the macroblocks in SP and SI slices. The QS_Y quantisation parameter for the slice is computed as:

$$QS_Y = 26 + pic_init_qs_minus26 + slice_qs_delta \quad (7\text{-}28)$$

The value of slice_qs_delta shall be limited such that QS_Y is in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in SI slices with mb_type equal to SI and all macroblocks in SP slices with prediction mode equal to inter.

disable_deblocking_filter_idc specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_deblocking_filter_idc is not present in the slice header, the value of disable_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_deblocking_filter_idc shall be in the range of 0 to 2, inclusive.

slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{C0} deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as:

$$FilterOffsetA = slice_alpha_c0_offset_div2 << 1 \quad (7\text{-}29)$$

The value of slice_alpha_c0_offset_div2 shall be in the range of -6 to +6, inclusive. When slice_alpha_c0_offset_div2 is not present in the slice header, the value of slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations controlled by the macroblocks within the slice. From this value, the offset that is applied when addressing the β table of the deblocking filter shall be computed as:

$$FilterOffsetB = slice_beta_offset_div2 << 1 \quad (7\text{-}30)$$

The value of slice_beta_offset_div2 shall be in the range of -6 to +6, inclusive. When slice_beta_offset_div2 is not present in the slice header the value of slice_beta_offset_div2 shall be inferred to be equal to 0.

slice_group_change_cycle is used to derive the number of slice group map units in slice group 0 when slice_group_map_type is equal to 3, 4, or 5, as specified by

$$MapUnitsInSliceGroup0 = Min(slice_group_change_cycle * SliceGroupChangeRate, PicSizeInMapUnits) \quad (7\text{-}31)$$

The value of slice_group_change_cycle is represented in the bitstream by the following number of bits

$$Ceil(Log2(PicSizeInMapUnits \div SliceGroupChangeRate + 1)) \quad (7\text{-}32)$$

The value of slice_group_change_cycle shall be in the range of 0 to Ceil(PicSizeInMapUnits÷SliceGroupChangeRate), inclusive.

7.4.3.1 Reference picture list reordering semantics

The syntax elements reordering_of_pic_nums_idc, abs_diff_pic_num_minus1, and long_term_pic_num specify the change from the initial reference picture lists to the reference picture lists to be used for decoding the slice.

ref_pic_list_reordering_flag_l0 equal to 1 specifies that the syntax element reordering_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_reordering_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_reordering_flag_l0 is equal to 1, the number of times that reordering_of_pic_nums_idc is not equal to 3 following ref_pic_list_reordering_flag_l0 shall not exceed num_ref_idx_l0_active_minus1 + 1.

When RefPicList0[num_ref_idx_l0_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", ref_pic_list_reordering_flag_l0 shall be equal to 1 and reordering_of_pic_nums_idc shall not be equal to 3 until RefPicList0[num_ref_idx_l0_active_minus1] in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

ref_pic_list_reordering_flag_l1 equal to 1 specifies that the syntax element reordering_of_pic_nums_idc is present for specifying reference picture list 1. ref_pic_list_reordering_flag_l1 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_reordering_flag_l1 is equal to 1, the number of times that reordering_of_pic_nums_idc is not equal to 3 following ref_pic_list_reordering_flag_l1 shall not exceed num_ref_idx_l1_active_minus1 + 1.

When decoding a B slice and RefPicList1[num_ref_idx_l1_active_minus1] in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", ref_pic_list_reordering_flag_l1 shall be equal to 1 and reordering_of_pic_nums_idc shall not be equal to 3 until RefPicList1[num_ref_idx_l1_active_minus1] in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

reordering_of_pic_nums_idc together with abs_diff_pic_num_minus1 or long_term_pic_num specifies which of the reference pictures are re-mapped. The values of reordering_of_pic_nums_idc are specified in Table 7-7. The value of the first reordering_of_pic_nums_idc that follows immediately after ref_pic_list_reordering_flag_l0 or ref_pic_list_reordering_flag_l1 shall not be equal to 3.

Table 7-7 – reordering_of_pic_nums_idc operations for reordering of reference picture lists

reordering_of_pic_nums_idc	Reordering specified
0	abs_diff_pic_num_minus1 is present and corresponds to a difference to subtract from a picture number prediction value
1	abs_diff_pic_num_minus1 is present and corresponds to a difference to add to a picture number prediction value
2	long_term_pic_num is present and specifies the long-term picture number for a reference picture
3	End loop for reordering of the initial reference picture list

abs_diff_pic_num_minus1 plus 1 specifies the absolute difference between the picture number of the picture being moved to the current index in the list and the picture number prediction value. abs_diff_pic_num_minus1 shall be in the range of 0 to MaxPicNum – 1. The allowed values of abs_diff_pic_num_minus1 are further restricted as specified in subclause 8.2.4.3.1.

long_term_pic_num specifies the long-term picture number of the picture being moved to the current index in the list. When decoding a coded frame, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference frames or complementary reference field pairs marked as "used for long-term reference". When decoding a coded field, long_term_pic_num shall be equal to a LongTermPicNum assigned to one of the reference fields marked as "used for long-term reference".

7.4.3.2 Prediction weight table semantics

luma_log2_weight_denom is the base 2 logarithm of the denominator for all luma weighting factors. The value of luma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

chroma_log2_weight_denom is the base 2 logarithm of the denominator for all chroma weighting factors. The value of chroma_log2_weight_denom shall be in the range of 0 to 7, inclusive.

luma_weight_l0_flag equal to 1 specifies that weighting factors for the luma component of list 0 prediction are present. luma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

luma_weight_l0[i **]** is the weighting factor applied to the luma prediction value for list 0 prediction using RefPicList0[i]. When luma_weight_l0_flag is equal to 1, the value of luma_weight_l0[i] shall be in the range of –128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to $2^{\text{luma_log2_weight_denom}}$ for RefPicList0[i].

luma_offset_l0[i **]** is the additive offset applied to the luma prediction value for list 0 prediction using RefPicList0[i]. The value of luma_offset_l0[i] shall be in the range of –128 to 127, inclusive. When luma_weight_l0_flag is equal to 0, luma_offset_l0[i] shall be inferred as equal to 0 for RefPicList0[i].

chroma_weight_l0_flag equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction are present. chroma_weight_l0_flag equal to 0 specifies that these weighting factors are not present.

chroma_weight_l0[i **][** j **]** is the weighting factor applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. When chroma_weight_l0_flag is equal to 1, the value of chroma_weight_l0[i][j] shall be in the range of –128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0**,** chroma_weight_l0[i][j] shall be inferred to be equal to $2^{\text{chroma_log2_weight_denom}}$ for RefPicList0[i].

chroma_offset_l0[i **][** j **]** is the additive offset applied to the chroma prediction values for list 0 prediction using RefPicList0[i] with j equal to 0 for Cb and j equal to 1 for Cr. The value of chroma_offset_l0[i][j] shall be in the range of -128 to 127, inclusive. When chroma_weight_l0_flag is equal to 0**,** chroma_offset_l0[i][j] shall be inferred to be equal to 0 for RefPicList0[i].

luma_weight_l1_flag, luma_weight_l1, **luma_offset_l1**, **chroma_weight_l1_flag**, **chroma_weight_l1**, **chroma_offset_l1** have the same semantics as luma_weight_l0_flag, luma_weight_l0, luma_offset_l0, chroma_weight_l0_flag, chroma_weight_l0, chroma_offset_l0, respectively, with l0, list 0, and List0 replaced by l1, list 1, and List1, respectively.

7.4.3.3 Decoded reference picture marking semantics

The syntax elements no_output_of_prior_pics_flag, long_term_reference_flag, adaptive_ref_pic_marking_mode_flag, memory_management_control_operation, difference_of_pic_nums_minus1, long_term_frame_idx, long_term_pic_num, and max_long_term_frame_idx_plus1 specify marking of the reference pictures.

The marking of a reference picture can be "unused for reference", "used for short-term reference", or "used for long-term reference", but only one among these three. When a reference picture is referred to as being marked as "used for reference", this collectively refers to the picture being marked as "used for short-term reference" or "used for long-term reference" (but not both). A reference picture that is marked as "used for short-term reference" is referred to as a short-term reference picture. A reference picture that is marked as "used for long-term reference" is referred to as a long-term reference picture.

The syntax element adaptive_ref_pic_marking_mode_flag and the content of the decoded reference picture marking syntax structure shall be identical for all coded slices of a coded picture.

The syntax category of the decoded reference picture marking syntax structure shall be inferred as follows.

– If the decoded reference picture marking syntax structure is in a slice header, the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 2.

– Otherwise (the decoded reference picture marking syntax structure is in a decoded reference picture marking repetition SEI message as specified in Annex D), the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 5.

no_output_of_prior_pics_flag specifies how the previously-decoded pictures in the decoded picture buffer are treated after decoding of an IDR picture. See Annex C. When the IDR picture is the first IDR picture in the bitstream, the value of no_output_of_prior_pics_flag has no effect on the decoding process. When the IDR picture is not the first IDR picture in the bitstream and the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived

from the active sequence parameter set is different from the value of PicWidthInMbs, FrameHeightInMbs, or max_dec_frame_buffering derived from the sequence parameter set active for the preceding sequence, no_output_of_prior_pics_flag equal to 1 may be inferred by the decoder, regardless of the actual value of no_output_of_prior_pics_flag.

long_term_reference_flag equal to 0 specifies that the MaxLongTermFrameIdx variable is set equal to “no long-term frame indices” and that the IDR picture is marked as “used for short-term reference”. long_term_reference_flag equal to 1 specifies that the MaxLongTermFrameIdx variable is set equal to 0 and that the current IDR picture is marked “used for long-term reference” and is assigned LongTermFrameIdx equal to 0. When num_ref_frames is equal to 0, long_term_reference_flag shall be equal to 0.

adaptive_ref_pic_marking_mode_flag selects the reference picture marking mode of the currently decoded picture as specified in Table 7-8. adaptive_ref_pic_marking_mode_flag shall be equal to 1 when the number of frames, complementary field pairs, and non-paired fields that are currently marked as "used for long-term reference" is equal to Max(num_ref_frames, 1).

Table 7-8 – Interpretation of adaptive_ref_pic_marking_mode_flag

adaptive_ref_pic_marking_mode_flag	Reference picture marking mode specified
0	Sliding window reference picture marking mode: A marking mode providing a first-in first-out mechanism for short-term reference pictures.
1	Adaptive reference picture marking mode: A reference picture marking mode providing syntax elements to specify marking of reference pictures as “unused for reference” and to assign long-term frame indices.

memory_management_control_operation specifies a control operation to be applied to affect the reference picture marking. The memory_management_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_control_operation. The values and control operations associated with memory_management_control_operation are specified in Table 7-9. The memory_management_control_operation syntax elements are processed by the decoding process in the order in which they appear in the slice header, and the semantics constraints expressed for each memory_management_control_operation apply at the specific position in that order at which that individual memory_management_control_operation is processed.

For interpretation of memory_management_control_operation, the term reference picture is interpreted as follows.

– If the current picture is a frame, the term reference picture refers either to a reference frame or a complementary reference field pair.

– Otherwise (the current picture is a field), the term reference picture refers either to a reference field or a field of a reference frame.

memory_management_control_operation shall not be equal to 1 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 2 in a slice header unless the specified long-term picture number refers to a reference picture that is marked as "used for long-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 in a slice header unless the specified reference picture is marked as "used for short-term reference" when the memory_management_control_operation is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 or 6 if the value of the variable MaxLongTermFrameIdx is equal to "no long-term frame indices" when the memory_management_control_operation is processed by the decoding process.

Not more than one memory_management_control_operation equal to 4 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 5 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 6 shall be present in a slice header.

memory_management_control_operation shall not be equal to 5 in a slice header unless no memory_management_control_operation in the range of 1 to 3 is present in the same decoded reference picture marking syntax structure.

A memory_management_control_operation equal to 5 shall not follow a memory_management_control_operation equal to 6 in the same slice header.

When a memory_management_control_operation equal to 6 is present, any memory_management_control_operation equal to 2, 3, or 4 that follows the memory_management_control_operation equal to 6 within the same slice header shall not specify the current picture to be marked as "unused for reference".

NOTE 1 – These constraints prohibit any combination of multiple memory_management_control_operation syntax elements that would specify the current picture to be marked as "unused for reference". However, some other combinations of memory_management_control_operation syntax elements are permitted that may affect the marking status of other reference pictures more than once in the same slice header. In particular, it is permitted for a memory_management_control_operation equal to 3 that specifies a long-term frame index to be assigned to a particular short-term reference picture to be followed in the same slice header by a memory_management_control_operation equal to 2, 3, 4 or 6 that specifies the same reference picture to subsequently be marked as "unused for reference".

Table 7-9 – Memory management control operation (memory_management_control_operation) values

memory_management_control_operation	Memory Management Control Operation
0	End memory_management_control_operation syntax element loop
1	Mark a short-term reference picture as “unused for reference”
2	Mark a long-term reference picture as “unused for reference”
3	Mark a short-term reference picture as "used for long-term reference" and assign a long-term frame index to it
4	Specify the maximum long-term frame index and mark all long-term reference pictures having long-term frame indices greater than the maximum value as "unused for reference"
5	Mark all reference pictures as "unused for reference" and set the MaxLongTermFrameIdx variable to "no long-term frame indices"
6	Mark the current picture as "used for long-term reference" and assign a long-term frame index to it

When decoding a field and a memory_management_control_operation command equal to 3 is present that assigns a long-term frame index to a field that is part of a short-term reference frame or part of a short-term complementary reference field pair, another memory_management_control_operation command to assign the same long-term frame index to the other field of the same frame or complementary reference field pair shall be present in the same decoded reference picture marking syntax structure.

NOTE 2 – The above requirement must be fulfilled even when the field referred to by the memory_management_control_operation equal to 3 is subsequently marked as "unused for reference" (for example when a memory_management_control_operation equal to 2 is present in the same slice header that causes the field to be marked as "unused for reference").

When the first field (in decoding order) of a complementary reference field pair includes a long_term_reference_flag equal to 1 or a memory_management_control_operation command equal to 6, the decoded reference picture marking syntax structure for the other field of the complementary reference field pair shall contain a memory_management_control_operation command equal to 6 that assigns the same long-term frame index to the other field.

NOTE 3 – The above requirement must be fulfilled even when the first field of the complementary reference field pair is subsequently marked as "unused for reference" (for example, when a memory_management_control_operation equal to 2 is present in the slice header of the second field that causes the first field to be marked as "unused for reference").

difference_of_pic_nums_minus1 is used (with memory_management_control_operation equal to 3 or 1) to assign a long-term frame index to a short-term reference picture or to mark a short-term reference picture as “unused for reference”. When the associated memory_management_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for reference" and not previously assigned to a long-term frame index.

The resulting picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 4 – When field_pic_flag is equal to 0, the resulting picture number must be a picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 1.

– Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields.

long_term_pic_num is used (with memory_management_control_operation equal to 2) to mark a long-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, long_term_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures that is currently marked as "used for long-term reference".

The resulting long-term picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 5 – When field_pic_flag is equal to 0, the resulting long-term picture number must be a long-term picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when field_pic_flag is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a memory_management_control_operation equal to 2.

– Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields.

long_term_frame_idx is used (with memory_management_control_operation equal to 3 or 6) to assign a long-term frame index to a picture. When the associated memory_management_control_operation is processed by the decoding process, the value of long_term_frame_idx shall be in the range of 0 to MaxLongTermFrameIdx, inclusive.

max_long_term_frame_idx_plus1 minus 1 specifies the maximum value of long-term frame index allowed for long-term reference pictures (until receipt of another value of max_long_term_frame_idx_plus1). The value of max_long_term_frame_idx_plus1 shall be in the range of 0 to num_ref_frames, inclusive.

7.4.4 Slice data semantics

cabac_alignment_one_bit is a bit equal to 1.

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as a B macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs – CurrMbAddr, inclusive.

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

mb_field_decoding_flag equal to 0 specifies that the current macroblock pair is a frame macroblock pair. mb_field_decoding_flag equal to 1 specifies that the macroblock pair is a field macroblock pair. Both macroblocks of a

frame macroblock pair are referred to in the text as frame macroblocks, whereas both macroblocks of a field macroblock pair are referred to in the text as field macroblocks.

When mb_field_decoding_flag is not present for either macroblock of a macroblock pair, the value of mb_field_decoding_flag is derived as follows.

– If there is a neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice, the value of mb_field_decoding_flag shall be inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately to the left of the current macroblock pair,

– Otherwise, if there is no neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice and there is a neighbouring macroblock pair immediately above the current macroblock pair in the same slice, the value of mb_field_decoding_flag shall be inferred to be equal to the value of mb_field_decoding_flag for the neighbouring macroblock pair immediately above the current macroblock pair,

– Otherwise (there is no neighbouring macroblock pair either immediately to the left or immediately above the current macroblock pair in the same slice), the value of mb_field_decoding_flag shall be inferred to be equal to 0.

end_of_slice_flag equal to 0 specifies that another macroblock is following in the slice. end_of_slice_flag equal to 1 specifies the end of the slice and that no further macroblock follows.

The function NextMbAddress( ) used in the slice data syntax table is specified in subclause 8.2.2.

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for I, SI, P, SP, and B slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by the NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function). When a value is not applicable it is designated by “na”. In the text, the value of mb_type may be referred to as the macroblock type and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as “X prediction macroblocks”.

Table 7-10 shows the allowed collective macroblock types for each slice_type.

NOTE 1 – There are some macroblock types with Pred_L0 prediction mode that are classified as B macroblock types.

Table 7-10 – Allowed collective macroblock types for slice_type

slice_type	allowed collective macroblock types
I (slice)	I (see Table 7-11) (macroblock types)
P (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)
B (slice)	B (see Table 7-14) and I (see Table 7-11) (macroblock types)
SI (slice)	SI (see Table 7-12) and I (see Table 7-11) (macroblock types)
SP (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)

transform_size_8x8_flag equal to 1 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 8x8 blocks shall be invoked for luma samples. transform_size_8x8_flag equal to 0 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 4x4 blocks shall be invoked for luma samples. When transform_size_8x8_flag is not present in the bitstream, it shall be inferred to be equal to 0.

NOTE 2 – When the current macroblock prediction mode MbPartPredMode(mb_type, 0) is equal to Intra_16x16, transform_size_8x8_flag is not present in the bitstream and then inferred to be equal to 0.

When sub_mb_type[mbPartIdx] (see subclause 7.4.5.2) is present in the bitstream for all 8x8 blocks indexed by mbPartIdx = 0..3, the variable noSubMbPartSizeLessThan8x8Flag indicates whether for each of the four 8x8 blocks the corresponding SubMbPartWidth(sub_mb_type[mbPartIdx]) and SubMbPartHeight(sub_mb_type[mbPartIdx]) are both equal to 8.

NOTE 3 – When noSubMbPartSizeLessThan8x8Flag is equal to 0 and the current macroblock type is not equal to I_NxN, transform_size_8x8_flag is not present in the bitstream and then inferred to be equal to 0.

Macroblock types that may be collectively referred to as I macroblock types are specified in Table 7-11.

The macroblock types for I slices are all I macroblock types.

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-33	Equation 7-33
0	I_NxN	1	Intra_8x8	na	Equation 7-33	Equation 7-33
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0
7	I_16x16_2_1_0	na	Intra_16x16	2	1	0
8	I_16x16_3_1_0	na	Intra_16x16	3	1	0
9	I_16x16_0_2_0	na	Intra_16x16	0	2	0
10	I_16x16_1_2_0	na	Intra_16x16	1	2	0
11	I_16x16_2_2_0	na	Intra_16x16	2	2	0
12	I_16x16_3_2_0	na	Intra_16x16	3	2	0
13	I_16x16_0_0_1	na	Intra_16x16	0	0	15
14	I_16x16_1_0_1	na	Intra_16x16	1	0	15
15	I_16x16_2_0_1	na	Intra_16x16	2	0	15
16	I_16x16_3_0_1	na	Intra_16x16	3	0	15
17	I_16x16_0_1_1	na	Intra_16x16	0	1	15
18	I_16x16_1_1_1	na	Intra_16x16	1	1	15
19	I_16x16_2_1_1	na	Intra_16x16	2	1	15
20	I_16x16_3_1_1	na	Intra_16x16	3	1	15
21	I_16x16_0_2_1	na	Intra_16x16	0	2	15
22	I_16x16_1_2_1	na	Intra_16x16	1	2	15
23	I_16x16_2_2_1	na	Intra_16x16	2	2	15
24	I_16x16_3_2_1	na	Intra_16x16	3	2	15
25	I_PCM	na	na	na	na	na

The following semantics are assigned to the macroblock types in Table 7-11.

I_NxN: A mnemonic name for mb_type equal to 0 with MbPartPredMode(mb_type, 0) equal to Intra_4x4 or Intra_8x8.

I_16x16_0_0_0, I_16x16_1_0_0, I_16x16_2_0_0, I_16x16_3_0_0, I_16x16_0_1_0, I_16x16_1_1_0, I_16x16_2_1_0, I_16x16_3_1_0, I_16x16_0_2_0, I_16x16_1_2_0, I_16x16_2_2_0, I_16x16_3_2_0, I_16x16_0_0_1, I_16x16_1_0_1, I_16x16_2_0_1, I_16x16_3_0_1, I_16x16_0_1_1, I_16x16_1_1_1, I_16x16_2_1_1, I_16x16_3_1_1, I_16x16_0_2_1, I_16x16_1_2_1, I_16x16_2_2_1, I_16x16_3_2_1: the macroblock is coded as an Intra_16x16 prediction mode macroblock.

To each Intra_16x16 prediction macroblock, an Intra16x16PredMode is assigned, which specifies the Intra_16x16 prediction mode. CodedBlockPatternChroma contains the coded block pattern value for chroma as specified in Table 7-15. When chroma_format_idc is equal to 0, CodedBlockPatternChroma shall be equal to 0. CodedBlockPatternLuma specifies whether, for the luma component, non-zero AC transform coefficient levels are present. CodedBlockPatternLuma equal to 0 specifies that all AC transform coefficient levels in the luma component of the macroblock are equal to 0. CodedBlockPatternLuma equal to 15 specifies that at least one of the AC transform coefficient levels in the luma component of the macroblock is non-zero, requiring scanning of AC transform coefficient levels for all 16 of the 4x4 blocks in the 16x16 block.

Intra_4x4 specifies the macroblock prediction mode and specifies that the Intra_4x4 prediction process is invoked as specified in subclause 8.3.1. Intra_4x4 is an Intra macroblock prediction mode.

Intra_8x8 specifies the macroblock prediction mode and specifies that the Intra_8x8 prediction process is invoked as specified in subclause 8.3.2. Intra_8x8 is an Intra macroblock prediction mode.

Intra_16x16 specifies the macroblock prediction mode and specifies that the Intra_16x16 prediction process is invoked as specified in subclause 8.3.3. Intra_16x16 is an Intra macroblock prediction mode.

For a macroblock coded with mb_type equal to I_PCM, the Intra macroblock prediction mode shall be inferred.

A macroblock type that may be referred to as SI macroblock type is specified in Table 7-12.

The macroblock types for SI slices are specified in Table 7-12 and 7-11. The mb_type value 0 is specified in Table 7-12 and the mb_type values 1 to 26 are specified in Table 7-11, indexed by subtracting 1 from the value of mb_type.

Table 7-12 – Macroblock type with value 0 for SI slices

mb_type	Name of mb_type	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	SI	Intra_4x4	na	Equation 7-33	Equation 7-33

The following semantics are assigned to the macroblock type in Table 7-12. The SI macroblock is coded as Intra_4x4 prediction macroblock.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for P and SP slices are specified in Table 7-13 and 7-11. mb_type values 0 to 4 are specified in Table 7-13 and mb_type values 5 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type.

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

The following semantics are assigned to the macroblock types in Table 7-13.

– P_L0_16x16: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples.

– P_L0_L0_MxN, with MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively.

– P_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– P_8x8ref0: has the same semantics as P_8x8 but no syntax element for the reference index (ref_idx_l0) is present in the bitstream and ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0 for all sub-macroblocks of the macroblock (with indices mbPartIdx equal to 0..3).

– P_Skip: no further data is present for the macroblock in the bitstream.

The following semantics are assigned to the macroblock prediction modes (MbPartPredMode( )) in Table 7-13.

– Pred_L0: specifies that the inter prediction process is invoked using list 0 prediction. Pred_L0 is an Inter macroblock prediction mode.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for B slices are specified in Table 7-14 and 7-11. The mb_type values 0 to 22 are specified in Table 7-14 and the mb_type values 23 to 48 are specified in Table 7-11, indexed by subtracting 23 from the value of mb_type.

Table 7-14 – Macroblock type values 0 to 22 for B slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	B_Direct_16x16	na	Direct	na	8	8
1	B_L0_16x16	1	Pred_L0	na	16	16
2	B_L1_16x16	1	Pred_L1	na	16	16
3	B_Bi_16x16	1	BiPred	na	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8
9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	na	na	8	8
inferred	B_Skip	na	Direct	na	8	8

The following semantics are assigned to the macroblock types in Table 7-14:

– B_Direct_16x16: no motion vector differences or reference indices are present for the macroblock in the bitstream. The functions MbPartWidth(B_Direct_16x16), and MbPartHeight(B_Direct_16x16) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_16x16 with X being replaced by L0, L1, or Bi: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples. For a macroblock with type B_X_16x16 with X being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream for the macroblock. For a macroblock with type B_X_16x16 with X being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock.

– B_X0_X1_MxN, with X0, X1 referring to the first and second macroblock partition and being replaced by L0, L1, or Bi, and MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively. For a macroblock partition X0 or X1 with X0 or X1 being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream. For a macroblock partition X0 or X1 with X0 or X1 being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock partition.

– B_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– B_Skip: no further data is present for the macroblock in the bitstream. The functions MbPartWidth(B_Skip), and MbPartHeight(B_Skip) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

The following semantics are assigned to the macroblock prediction modes (MbPartPredMode( )) in Table 7-14.

– Direct: no motion vector differences or reference indices are present for the macroblock (in case of B_Skip or B_Direct_16x16) in the bitstream. Direct is an Inter macroblock prediction mode.

– Pred_L0: see semantics for Table 7-13.

– Pred_L1: specifies that the Inter prediction process is invoked using list 1 prediction. Pred_L1 is an Inter macroblock prediction mode.

– BiPred: specifies that the Inter prediction process is invoked using list 0 and list 1 prediction. BiPred is an Inter macroblock prediction mode.

pcm_alignment_zero_bit is a bit equal to 0.

pcm_sample_luma[i] is a sample value. The first pcm_sample_luma[i] values represent luma sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is $BitDepth_Y$. When profile_idc is not equal to 100, 110, 122, or 144, pcm_sample_luma[i] shall not be equal to 0.

pcm_sample_chroma[i] is a sample value. The first MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cb sample values in the raster scan within the macroblock and the remaining MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cr sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is $BitDepth_C$. When profile_idc is not equal to 100, 110, 122, or 144, pcm_sample_chroma[i] shall not be equal to 0.

coded_block_pattern specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient levels. For macroblocks with prediction mode not equal to Intra_16x16, coded_block_pattern is present in the bitstream and the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as follows.

CodedBlockPatternLuma = coded_block_pattern % 16
CodedBlockPatternChroma = coded_block_pattern / 16 (7-33)

When coded_block_pattern is present, CodedBlockPatternLuma specifies, for each of the four 8x8 luma blocks of the macroblock, one of the following cases.

– All transform coefficient levels of the four 4x4 luma blocks in the 8x8 luma block are equal to zero

– One or more transform coefficient levels of one or more of the 4x4 luma blocks in the 8x8 luma block shall be non-zero valued.

The meaning of CodedBlockPatternChroma is specified in Table 7-15.

Table 7-15 – Specification of CodedBlockPatternChroma values

CodedBlockPatternChroma	Description
0	All chroma transform coefficient levels are equal to 0.
1	One or more chroma DC transform coefficient levels shall be non-zero valued. All chroma AC transform coefficient levels are equal to 0.
2	Zero or more chroma DC transform coefficient levels are non-zero valued. One or more chroma AC transform coefficient levels shall be non-zero valued.

mb_qp_delta can change the value of QP_Y in the macroblock layer. The decoded value of mb_qp_delta shall be in the range of $-(26 + QpBdOffset_Y / 2)$ to $+(25 + QpBdOffset_Y / 2)$, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = ((QP_{Y,PREV} + mb_qp_delta + 52 + 2 * QpBdOffset_Y) \% (52 + QpBdOffset_Y)) - QpBdOffset_Y \qquad (7\text{-}34)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y, of the previous macroblock in decoding order in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ derived in Equation 7-27 at the start of each slice.

The value of QP'_Y is derived as

$$QP'_Y = QP_Y + QpBdOffset_Y \qquad (7\text{-}35)$$

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction modes are derived using the following syntax elements.

prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]** and **rem_intra4x4_pred_mode[** luma4x4BlkIdx **]** specify the Intra_4x4 prediction of the 4x4 luma block with index luma4x4BlkIdx = 0..15.

prev_intra8x8_pred_mode_flag[luma8x8BlkIdx] and **rem_intra8x8_pred_mode**[luma8x8BlkIdx] specify the Intra_8x8 prediction of the 8x8 luma block with index luma8x8BlkIdx = 0..3.

intra_chroma_pred_mode specifies the type of spatial prediction used for chroma in macroblocks using Intra_4x4 or Intra_16x16 prediction, as shown in Table 7-16. The value of intra_chroma_pred_mode shall be in the range of 0 to 3, inclusive.

Table 7-16 – Relationship between intra_chroma_pred_mode and spatial prediction modes

intra_chroma_pred_mode	Intra Chroma Prediction Mode
0	DC
1	Horizontal
2	Vertical
3	Plane

ref_idx_l0[mbPartIdx **]** when present, specifies the index in reference picture list 0 of the reference picture to be used for prediction.

The range of ref_idx_l0[mbPartIdx], the index in list 0 of the reference picture, and, if applicable, the parity of the field within the reference picture used for prediction are specified as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), the value of ref_idx_l0[mbPartIdx] shall be in the range of 0 to 2 * num_ref_idx_l0_active_minus1 + 1, inclusive.

When only one reference picture is used for inter prediction, the values of ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

mvd_l0[mbPartIdx **][** 0 **][** compIdx **]** specifies the difference between a vector component to be used and its prediction. The index mbPartIdx specifies to which macroblock partition mvd_l0 is assigned. The partitioning of the macroblock is specified by mb_type. The horizontal motion vector component difference is decoded first in decoding order and is assigned CompIdx = 0. The vertical motion vector component is decoded second in decoding order and is assigned CompIdx = 1. The range of the components of mvd_l0[mbPartIdx][0][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in Annex A.

mvd_l1[mbPartIdx **][** 0 **][** compIdx **]** has the same semantics as mvd_l0, with l0 and L0 replaced by l1 and L1, respectively.

7.4.5.2 Sub-macroblock prediction semantics

sub_mb_type[mbPartIdx **]** specifies the sub-macroblock types.

Tables and semantics are specified for the various sub-macroblock types for P, and B macroblock types. Each table presents the value of sub_mb_type, the name of sub_mb_type, the number of sub-macroblock partitions used (given by the NumSubMbPart(sub_mb_type) function), and the prediction mode of the sub-macroblock (given by the SubMbPredMode(sub_mb_type) function). In the text, the value of sub_mb_type may be referred to by “sub-macroblock type”. In the text, the value of SubMbPredMode( ) may be referred to by “sub-macroblock prediction mode”.

The interpretation of sub_mb_type[mbPartIdx] for P macroblock types is specified in Table 7-17, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present.

Table 7-17 – Sub-macroblock types in P macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	na	na	na	na	na
0	P_L0_8x8	1	Pred_L0	8	8
1	P_L0_8x4	2	Pred_L0	8	4
2	P_L0_4x8	2	Pred_L0	4	8
3	P_L0_4x4	4	Pred_L0	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-17.

– P_L0_MxN, with MxN being replaced by 8x8, 8x4, 4x8, or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, two luma partitions of size MxN equal to 8x4, or two luma

partitions of size MxN equal to 4x8, or four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively.

The following semantics are assigned to the sub-macroblock prediction modes (SubMbPredMode( )) in Table 7-17.

– Pred_L0: see semantics for Table 7-13.

The interpretation of sub_mb_type[mbPartIdx] for B macroblock types is specified in Table 7-18, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present, and the inferred value "mb_type" specifies that the name of sub_mb_type[mbPartIdx] is the same as the name of mb_type for this case.

Table 7-18 – Sub-macroblock types in B macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	mb_type	4	Direct	4	4
0	B_Direct_8x8	4	Direct	4	4
1	B_L0_8x8	1	Pred_L0	8	8
2	B_L1_8x8	1	Pred_L1	8	8
3	B_Bi_8x8	1	BiPred	8	8
4	B_L0_8x4	2	Pred_L0	8	4
5	B_L0_4x8	2	Pred_L0	4	8
6	B_L1_8x4	2	Pred_L1	8	4
7	B_L1_4x8	2	Pred_L1	4	8
8	B_Bi_8x4	2	BiPred	8	4
9	B_Bi_4x8	2	BiPred	4	8
10	B_L0_4x4	4	Pred_L0	4	4
11	B_L1_4x4	4	Pred_L1	4	4
12	B_Bi_4x4	4	BiPred	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-18:

– B_Skip and B_Direct_16x16: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth( ) and SubMbPartHeight( ) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_Direct_8x8: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth(B_Direct_8x8) and SubMbPartHeight(B_Direct_8x8) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

– B_X_MxN, with X being replaced by L0, L1, or Bi, and MxN being replaced by 8x8, 8x4, 4x8 or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 8x4, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 4x8, or the samples of the sub-macroblock are predicted using four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively. All sub-macroblock partitions share the same reference index. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_X_MxN with X being replaced by either L0 or L1, one motion vector difference is present in the bitstream. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_Bi_MxN, two motion vector difference are present in the bitstream.

The following semantics are assigned to the sub-macroblock prediction modes (SubMbPredMode( )) in Table 7-18.

– Direct: see semantics for Table 7-14.

– Pred_L0: see semantics for Table 7-13.

– Pred_L1: see semantics for Table 7-14.

– BiPred: see semantics for Table 7-14.

ref_idx_l0[mbPartIdx **]** has the same semantics as ref_idx_l0 in subclause 7.4.5.1.

ref_idx_l1[mbPartIdx **]** has the same semantics as ref_idx_l1 in subclause 7.4.5.1.

mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]** has the same semantics as mvd_l0 in subclause 7.4.5.1, except that it is applied to the sub-macroblock partition index with subMbPartIdx. The indices mbPartIdx and subMbPartIdx specify to which macroblock partition and sub-macroblock partition mvd_l0 is assigned.

mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]** has the same semantics as mvd_l1 in subclause 7.4.5.1.

7.4.5.3 Residual data semantics

The syntax structure residual_block( ), which is used for parsing the transform coefficient levels, is assigned as follows.

– If entropy_coding_mode_flag is equal to 0, residual_block is set equal to residual_block_cavlc, which is used for parsing the syntax elements for transform coefficient levels.

– Otherwise (entropy_coding_mode_flag is equal to 1), residual_block is set equal to residual_block_cabac, which is used for parsing the syntax elements for transform coefficient levels.

Depending on mb_type, luma or chroma, and chroma format, the syntax structure residual_block(coeffLevel, maxNumCoeff) is used with the arguments coeffLevel, which is a list containing the maxNumCoeff transform coefficient levels that are parsed in residual_block( ) and maxNumCoeff as follows.

– Depending on MbPartPredMode(mb_type, 0), the following applies.

  – If MbPartPredMode(mb_type, 0) is equal to Intra_16x16, the transform coefficient levels are parsed into the list Intra16x16DCLevel and into the 16 lists Intra16x16ACLevel[i]. Intra16x16DCLevel contains the 16 transform coefficient levels of the DC transform coefficient levels for each 4x4 luma block. For each of the 16 4x4 luma blocks indexed by i = 0..15, the 15 AC transform coefficients levels of the i-th block are parsed into the i-th list Intra16x16ACLevel[i].

  – Otherwise (MbPartPredMode(mb_type, 0) is not equal to Intra_16x16), the following applies.

    – If transform_size_8x8_flag is equal to 0, for each of the 16 4x4 luma blocks indexed by i = 0..15, the 16 transform coefficient levels of the i-th block are parsed into the i-th list LumaLevel[i].

    – Otherwise (transform_size_8x8_flag is equal to 1), for each of the 4 8x8 luma blocks indexed by i8x8 = 0..3, the following applies.

      – If entropy_coding_mode_flag is equal to 0, first for each of the 4 4x4 luma blocks indexed by i4x4 = 0..3, the 16 transform coefficient levels of the i4x4-th block are parsed into the (i8x8 * 4 + i4x4)-th list LumaLevel[i8x8 * 4 + i4x4]. Then, the 64 transform coefficient levels of the i8x8-th 8x8 luma block which are indexed by 4 * i + i4x4, where i = 0..15 and i4x4 = 0..3, are derived as LumaLevel8x8[i8x8][4 * i + i4x4] = LumaLevel[i8x8 * 4 + i4x4][i].

NOTE – The 4x4 luma blocks with luma4x4BlkIdx = i8x8 * 4 + i4x4 containing every fourth transform coefficient level of the corresponding i8x8-th 8x8 luma block with offset i4x4 are assumed to represent spatial locations given by the inverse 4x4 luma block scanning process in subclause 6.4.3.

– Otherwise (entropy_coding_mode_flag is equal to 1), the 64 transform coefficient levels of the i8x8-th block are parsed into the i8x8-th list LumaLevel8x8[i8x8].

– For each chroma component, indexed by iCbCr = 0..1, the DC transform coefficient levels of the 4 * NumC8x8 4x4 chroma blocks are parsed into the iCbCr-th list ChromaDCLevel[iCbCr].

– For each of the 4x4 chroma blocks, indexed by i4x4 = 0..3 and i8x8 = 0..NumC8x8 – 1, of each chroma component, indexed by iCbCr = 0..1, the 15 AC transform coefficient levels are parsed into the (i8x8*4 + i4x4)-th list of the iCbCr-th chroma component ChromaACLevel[iCbCr][i8x8*4 + i4x4].

7.4.5.3.1 Residual block CAVLC semantics

The function TotalCoeff(coeff_token) that is used in subclause 7.3.5.3.1 returns the number of non-zero transform coefficient levels derived from coeff_token.

The function TrailingOnes(coeff_token) that is used in subclause 7.3.5.3.1 returns the trailing ones derived from coeff_token.

coeff_token specifies the total number of non-zero transform coefficient levels and the number of trailing one transform coefficient levels in a transform coefficient level scan. A trailing one transform coefficient level is one of up to three consecutive non-zero transform coefficient levels having an absolute value equal to 1 at the end of a scan of non-zero transform coefficient levels. The range of coeff_token is specified in subclause 9.2.1.

trailing_ones_sign_flag specifies the sign of a trailing one transform coefficient level as follows.

– If trailing_ones_sign_flag is equal to 0, the corresponding transform coefficient level is decoded as +1.

– Otherwise (trailing_ones_sign_flag equal to 1), the corresponding transform coefficient level is decoded as -1.

level_prefix and **level_suffix** specify the value of a non-zero transform coefficient level. The range of level_prefix and level_suffix is specified in subclause 9.2.2.

total_zeros specifies the total number of zero-valued transform coefficient levels that are located before the position of the last non-zero transform coefficient level in a scan of transform coefficient levels. The range of total_zeros is specified in subclause 9.2.3.

run_before specifies the number of consecutive transform coefficient levels in the scan with zero value before a non-zero valued transform coefficient level. The range of run_before is specified in subclause 9.2.3.

coeffLevel contains maxNumCoeff transform coefficient levels for the current list of transform coefficient levels.

7.4.5.3.2 Residual block CABAC semantics

coded_block_flag specifies whether the block contains non-zero transform coefficient levels as follows.

– If coded_block_flag is equal to 0, the block contains no non-zero transform coefficient levels.

– Otherwise (coded_block_flag is equal to 1), the block contains at least one non-zero transform coefficient level.

significant_coeff_flag[i **]** specifies whether the transform coefficient level at scanning position i is non-zero as follows.

– If significant_coeff_flag[i] is equal to 0, the transform coefficient level at scanning position i is set equal to 0;

– Otherwise (significant_coeff_flag[i] is equal to 1), the transform coefficient level at scanning position i has a non-zero value.

last_significant_coeff_flag[i **]** specifies for the scanning position i whether there are non-zero transform coefficient levels for subsequent scanning positions i + 1 to maxNumCoeff – 1 as follows.

– If last_significant_coeff_flag[i] is equal to 1, all following transform coefficient levels (in scanning order) of the block have value equal to 0.

– Otherwise (last_significant_coeff_flag[i] is equal to 0), there are further non-zero transform coefficient levels along the scanning path.

coeff_abs_level_minus1[i **]** is the absolute value of a transform coefficient level minus 1. The value of coeff_abs_level_minus1 is constrained by the limits in subclause 8.5.

coeff_sign_flag[i **]** specifies the sign of a transform coefficient level as follows.

– If coeff_sign_flag is equal to 0, the corresponding transform coefficient level has a positive value.

– Otherwise (coeff_sign_flag is equal to 1), the corresponding transform coefficient level has a negative value.

coeffLevel contains maxNumCoeff transform coefficient levels for the current list of transform coefficient levels.

8 Decoding process

Outputs of this process are decoded samples of the current picture (sometimes referred to by the variable CurrPic).

This clause describes the decoding process, given syntax elements and upper-case variables from clause 7.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Each picture referred to in this clause is a primary picture. Each slice referred to in this clause is a slice of a primary picture. Each slice data partition referred to in this clause is a slice data partition of a primary picture.

An overview of the decoding process is given as follows.

– The decoding of NAL units is specified in subclause 8.1.

– The processes in subclause 8.2 specify decoding processes using syntax elements in the slice layer and above.

  – Variables and functions relating to picture order count are derived in subclause 8.2.1. (only needed to be invoked for one slice of a picture)

  – Variables and functions relating to the macroblock to slice group map are derived in subclause 8.2.2. (only needed to be invoked for one slice of a picture)

  – The method of combining the various partitions when slice data partitioning is used is described in subclause 8.2.3.

  – When the frame_num of the current picture is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num is performed according to subclause 8.2.5.2 prior to the decoding of any slices of the current picture.

  – At the beginning of the decoding process for each P, SP, or B slice, the decoding process for reference picture lists construction specified in 8.2.4 performed for derivation of reference picture list 0 (RefPicList0), and when decoding a B slice, reference picture list 1 (RefPicList1).

  – When the current picture is a reference picture and after all slices of the current picture have been decoded, the decoded reference picture marking process in subclause 8.2.5 specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.

– The processes in subclauses 8.3, 8.4, 8.5, 8.6, and 8.7 specify decoding processes using syntax elements in the macroblock layer and above.

  – The intra prediction process for I and SI macroblocks, except for I_PCM macroblocks as specified in subclause 8.3, has intra prediction samples as its output. For I_PCM macroblocks subclause 8.3 directly specifies a picture construction process. The output are the constructed samples prior to the deblocking filter process.

  – The inter prediction process for P and B macroblocks is specified in subclause 8.4 with inter prediction samples being the output.

  – The transform coefficient decoding process and picture construction process prior to deblocking filter process are specified in subclause 8.5. That process derives samples for I and B macroblocks and for P macroblocks in P slices. The output are constructed samples prior to the deblocking filter process.

– The decoding process for P macroblocks in SP slices or SI macroblocks is specified in subclause 8.6. That process derives samples for P macroblocks in SP slices and for SI macroblocks. The output are constructed samples prior to the deblocking filter process.

– The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter as specified in subclause 8.7 with the output being the decoded samples.

8.1 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.2 describes the decoding process for NAL units with nal_unit_type equal to 1 through 5.

Subclauses 8.3 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1, 2, and 5.

Subclause 8.4 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 2.

Subclause 8.5 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.6 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.7 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 to 5.

NAL units with nal_unit_type equal to 7 and 8 contain sequence parameter sets and picture parameter sets, respectively. Picture parameter sets are used in the decoding processes of other NAL units as determined by reference to a picture parameter set within the slice headers of each picture. Sequence parameter sets are used in the decoding processes of other NAL units as determined by reference to a sequence parameter set within the picture parameter sets of each sequence.

No normative decoding process is specified for NAL units with nal_unit_type equal to 6, 9, 10, 11, and 12.

8.2 Slice decoding process

8.2.1 Decoding process for picture order count

Outputs of this process are TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable).

Picture order counts are used to determine initial picture orderings for reference pictures in the decoding of B slices (see subclauses 8.2.4.2.3 and 8.2.4.2.4), to represent picture order differences between frames or fields for motion vector derivation in temporal direct mode (see subclause 8.4.1.2.3), for implicit mode weighted prediction in B slices (see subclause 8.4.2.3.2), and for decoder conformance checking (see subclause C.4).

Picture order count information is derived for every frame, field (whether decoded from a coded field or as a part of a decoded frame), or complementary field pair as follows:

– Each coded frame is associated with two picture order counts, called TopFieldOrderCnt and BottomFieldOrderCnt for its top field and bottom field, respectively.

– Each coded field is associated with a picture order count, called TopFieldOrderCnt for a coded top field and BottomFieldOrderCnt for a bottom field.

– Each complementary field pair is associated with two picture order counts, which are the TopFieldOrderCnt for its coded top field and the BottomFieldOrderCnt for its coded bottom field, respectively.

TopFieldOrderCnt and BottomFieldOrderCnt indicate the picture order of the corresponding top field or bottom field relative to the first output field of the previous IDR picture or the previous reference picture including a memory_management_control_operation equal to 5 in decoding order.

TopFieldOrderCnt and BottomFieldOrderCnt are derived by invoking one of the decoding processes for picture order count type 0, 1, and 2 in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3, respectively. When the current picture includes a memory management control operation equal to 5, after the decoding of the current picture, tempPicOrderCnt is set equal to PicOrderCnt(CurrPic), TopFieldOrderCnt of the current picture (if any) is set equal to TopFieldOrderCnt - tempPicOrderCnt, and BottomFieldOrderCnt of the current picture (if any) is set equal to BottomFieldOrderCnt - tempPicOrderCnt.

The bitstream shall not contain data that results in Min(TopFieldOrderCnt, BottomFieldOrderCnt) not equal to 0 for a coded IDR frame, TopFieldOrderCnt not equal to 0 for a coded IDR top field, or BottomFieldOrderCnt not equal to 0 for a coded IDR bottom field. Thus, at least one of TopFieldOrderCnt and BottomFieldOrderCnt shall be equal to 0 for the fields of a coded IDR frame.

When the current picture is not an IDR picture, the following applies.

- Consider the list variable listD containing as elements the TopFieldOrderCnt and BottomFieldOrderCnt values associated with the list of pictures including all of the following
  - the first picture in the list is the previous picture of any of the following types
    - an IDR picture
    - a picture containing a memory_management_control_operation equal to 5
  - the following additional pictures.
    - If pic_order_cnt_type is equal to 0, all other pictures that follow in decoding order after the first picture in the list and are not "non-existing" frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2 and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is equal to 0 and the current picture is not a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.
    - Otherwise (pic_order_cnt_type is not equal to 0), all other pictures that follow in decoding order after the first picture in the list and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is not equal to 0, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.
- Consider the list variable listO which contains the elements of listD sorted in ascending order. listO shall not contain any of the following.
  - a pair of TopFieldOrderCnt and BottomFieldOrderCnt for a frame or complementary field pair that are not at consecutive positions in listO.
  - a TopFieldOrderCnt that has a value equal to another TopFieldOrderCnt.
  - a BottomFieldOrderCnt that has a value equal to another BottomFieldOrderCnt.
  - a BottomFieldOrderCnt that has a value equal to a TopFieldOrderCnt unless the BottomFieldOrderCnt and TopFieldOrderCnt belong to the same coded frame or complementary field pair.

The bitstream shall not contain data that results in values of TopFieldOrderCnt, BottomFieldOrderCnt, PicOrderCntMsb, or FrameNumOffset used in the decoding process as specified in subclauses 8.2.1.1 to 8.2.1.3 that exceed the range of values from -2^{31} to $2^{31}-1$, inclusive.

The function PicOrderCnt(picX) is specified as follows:

```
if( picX is a frame or a complementary field pair )
    PicOrderCnt( picX ) = Min( TopFieldOrderCnt, BottomFieldOrderCnt ) of the frame or complementary field
pair picX
else if( picX is a top field )
    PicOrderCnt( picX ) = TopFieldOrderCnt of field picX                                        (8-1)
else if( picX is a bottom field )
    PicOrderCnt( picX ) = BottomFieldOrderCnt of field picX
```

Then DiffPicOrderCnt(picA, picB) is specified as follows:

DiffPicOrderCnt(picA, picB) = PicOrderCnt(picA) - PicOrderCnt(picB) (8-2)

The bitstream shall not contain data that results in values of DiffPicOrderCnt(picA, picB) used in the decoding process that exceed the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE 1 – Let X be the current picture and Y and Z be two other pictures in the same sequence, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

NOTE 2 – Many applications assign PicOrderCnt(X) proportional to the sampling time of the picture X relative to the sampling time of an IDR picture.

When the current picture includes a memory_management_control_operation equal to 5, PicOrderCnt(CurrPic) shall be greater than PicOrderCnt(any other picture in listD).

8.2.1.1 Decoding process for picture order count type 0

This process is invoked when pic_order_cnt_type is equal to 0.

Input to this process is PicOrderCntMsb of the previous reference picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The variables prevPicOrderCntMsb and prevPicOrderCntLsb are derived as follows.

- If the current picture is an IDR picture, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.
- Otherwise (the current picture is not an IDR picture), the following applies.
  - If the previous reference picture in decoding order included a memory_management_control_operation equal to 5, the following applies.
    - If the previous reference picture in decoding order is not a bottom field, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to the value of TopFieldOrderCnt for the previous reference picture in decoding order.
    - Otherwise (the previous reference picture in decoding order is a bottom field), prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.
  - Otherwise (the previous reference picture in decoding order did not include a memory_management_control_operation equal to 5), prevPicOrderCntMsb is set equal to PicOrderCntMsb of the previous reference picture in decoding order and prevPicOrderCntLsb is set equal to the value of pic_order_cnt_lsb of the previous reference picture in decoding order.

PicOrderCntMsb of the current picture is derived as follows:

```
if( ( pic_order_cnt_lsb  <  prevPicOrderCntLsb )  &&
    ( ( prevPicOrderCntLsb – pic_order_cnt_lsb )  >=  ( MaxPicOrderCntLsb / 2 ) ) )
     PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb                    (8-3)
else if( ( pic_order_cnt_lsb  >  prevPicOrderCntLsb )  &&
         ( ( pic_order_cnt_lsb – prevPicOrderCntLsb )  >  ( MaxPicOrderCntLsb / 2 ) ) )
     PicOrderCntMsb = prevPicOrderCntMsb – MaxPicOrderCntLsb
else
     PicOrderCntMsb = prevPicOrderCntMsb
```

When the current picture is not a bottom field, TopFieldOrderCnt is derived as follows:

```
if( !field_pic_flag  ||  !bottom_field_flag )
     TopFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb                      (8-4)
```

When the current picture is not a top field, BottomFieldOrderCnt is derived as follows:

```
if( !field_pic_flag )
    BottomFieldOrderCnt = TopFieldOrderCnt + delta_pic_order_cnt_bottom
else if( bottom_field_flag )                                                    (8-5)
    BottomFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb
```

8.2.1.2 Decoding process for picture order count type 1

This process is invoked when pic_order_cnt_type is equal to 1.

Input to this process is FrameNumOffset of the previous picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are derived as specified in this subclause. Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows.

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The derivation proceeds in the following ordered steps.

1. The variable FrameNumOffset is derived as follows:

```
if( nal_unit_type  = =  5 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )                                             (8-6)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

2. The variable absFrameNum is derived as follows:

```
if( num_ref_frames_in_pic_order_cnt_cycle  !=  0 )
    absFrameNum = FrameNumOffset + frame_num
else                                                                            (8-7)
    absFrameNum = 0
if( nal_ref_idc  = =  0  &&  absFrameNum  >  0 )
    absFrameNum = absFrameNum – 1
```

3. When absFrameNum > 0, picOrderCntCycleCnt and frameNumInPicOrderCntCycle are derived as follows:

```
if( absFrameNum  >  0 ) {
    picOrderCntCycleCnt = ( absFrameNum – 1 ) / num_ref_frames_in_pic_order_cnt_cycle
    frameNumInPicOrderCntCycle = ( absFrameNum – 1 ) % num_ref_frames_in_pic_order_cnt_cycle     (8-8)
}
```

4. The variable expectedDeltaPerPicOrderCntCycle is derived as follows:

```
expectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    expectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]               (8-9)
```

5. The variable expectedPicOrderCnt is derived as follows:

```
if( absFrameNum > 0 ){
    expectedPicOrderCnt = picOrderCntCycleCnt * expectedDeltaPerPicOrderCntCycle
    for( i = 0; i <= frameNumInPicOrderCntCycle; i++ )
        expectedPicOrderCnt = expectedPicOrderCnt + offset_for_ref_frame[ i ]
} else
    expectedPicOrderCnt = 0
if( nal_ref_idc  = =  0 )                                                    (8-10)
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic
```

6. The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
    BottomFieldOrderCnt = TopFieldOrderCnt +
            offset_for_top_to_bottom_field + delta_pic_order_cnt[ 1 ]        (8-11)
} else if( !bottom_field_flag )
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
else
    BottomFieldOrderCnt = expectedPicOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 0 ]
```

8.2.1.3 Decoding process for picture order count type 2

This process is invoked when pic_order_cnt_type is equal to 2.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows.

– If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.

– Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The variable FrameNumOffset is derived as follows.

```
if( nal_unit_type  = =  5 )
    FrameNumOffset = 0
else if( prevFrameNum  >  frame_num )                                        (8-12)
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset
```

The variable tempPicOrderCnt is derived as follows:

```
if( nal_unit_type  = =  5 )
    tempPicOrderCnt = 0
else if( nal_ref_idc  = =  0 )                                               (8-13)
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num ) – 1
else
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num )
```

The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

```
if( !field_pic_flag ) {
    TopFieldOrderCnt = tempPicOrderCnt
    BottomFieldOrderCnt = tempPicOrderCnt                                    (8-14)
} else if( bottom_field_flag )
    BottomFieldOrderCnt = tempPicOrderCnt
else
    TopFieldOrderCnt = tempPicOrderCnt
```

NOTE 2 – Picture order count type 2 cannot be used in a coded video sequence that contains consecutive non-reference pictures that would result in more than one of these pictures having the same value of TopFieldOrderCnt or more than one of these pictures having the same value of BottomFieldOrderCnt.

NOTE 3 – Picture order count type 2 results in an output order that is the same as the decoding order.

8.2.2 Decoding process for macroblock to slice group map

Inputs to this process are the active picture parameter set and the slice header of the slice to be decoded.

Output of this process is a macroblock to slice group map MbToSliceGroupMap.

This process is invoked at the start of every slice.

NOTE – The output of this process is equal for all slices of a picture.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 3, 4, or 5, slice groups 0 and 1 have a size and shape determined by slice_group_change_direction_flag as shown in Table 8-1 and specified in subclauses 8.2.2.4 to 8.2.2.6.

Table 8-1 – Refined slice group map type

slice_group_map_type	slice_group_change_direction_flag	refined slice group map type
3	0	Box-out clockwise
3	1	Box-out counter-clockwise
4	0	Raster scan
4	1	Reverse raster scan
5	0	Wipe right
5	1	Wipe left

In such a case, MapUnitsInSliceGroup0 slice group map units in the specified growth order are allocated for slice group 0 and the remaining PicSizeInMapUnits – MapUnitsInSliceGroup0 slice group map units of the picture are allocated for slice group 1.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 4 or 5, the variable sizeOfUpperLeftGroup is defined as follows:

```
sizeOfUpperLeftGroup = ( slice_group_change_direction_flag ?
                         ( PicSizeInMapUnits – MapUnitsInSliceGroup0 ) : MapUnitsInSliceGroup0 )    (8-15)
```

The variable mapUnitToSliceGroupMap is derived as follows.

– If num_slice_groups_minus1 is equal to 0, the map unit to slice group map is generated for all i ranging from 0 to PicSizeInMapUnits – 1, inclusive, as specified by:

```
mapUnitToSliceGroupMap[ i ] = 0                                              (8-16)
```

– Otherwise (num_slice_groups_minus1 is not equal to 0), mapUnitToSliceGroupMap is derived as follows.

  – If slice_group_map_type is equal to 0, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.1 applies.

- Otherwise, if slice_group_map_type is equal to 1, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.2 applies.
- Otherwise, if slice_group_map_type is equal to 2, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.3 applies.
- Otherwise, if slice_group_map_type is equal to 3, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.4 applies.
- Otherwise, if slice_group_map_type is equal to 4, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.5 applies.
- Otherwise, if slice_group_map_type is equal to 5, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.6 applies.
- Otherwise (slice_group_map_type is equal to 6), the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.7 applies.

After derivation of the mapUnitToSliceGroupMap, the process specified in subclause 8.2.2.8 is invoked to convert the map unit to slice group map mapUnitToSliceGroupMap to the macroblock to slice group map MbToSliceGroupMap. After derivation of the macroblock to slice group map as specified in subclause 8.2.2.8, the function NextMbAddress(n) is defined as the value of the variable nextMbAddress derived as specified by:

```
i = n + 1
while( i < PicSizeInMbs  &&  MbToSliceGroupMap[ i ]  !=  MbToSliceGroupMap[ n ] )
    i++;
nextMbAddress = i                                                          (8-17)
```

8.2.2.1 Specification for interleaved slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 0.

The map unit to slice group map is generated as specified by:

```
i = 0
do
    for( iGroup = 0; iGroup <= num_slice_groups_minus1 && i < PicSizeInMapUnits;
          i += run_length_minus1[ iGroup++ ] + 1 )
         for( j = 0; j <= run_length_minus1[ iGroup ] && i + j < PicSizeInMapUnits; j++ )
              mapUnitToSliceGroupMap[ i + j ] = iGroup                     (8-18)
while( i < PicSizeInMapUnits )
```

8.2.2.2 Specification for dispersed slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 1.

The map unit to slice group map is generated as specified by:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = ( ( i % PicWidthInMbs ) +
                                      ( ( ( i / PicWidthInMbs ) * ( num_slice_groups_minus1 + 1 ) ) / 2 ) )
                                    % ( num_slice_groups_minus1 + 1 )      (8-19)
```

8.2.2.3 Specification for foreground with left-over slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 2.

The map unit to slice group map is generated as specified by:

```
for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = num_slice_groups_minus1
for( iGroup = num_slice_groups_minus1 – 1; iGroup >= 0; iGroup-- ) {
    yTopLeft = top_left[ iGroup ] / PicWidthInMbs
```

```
        xTopLeft = top_left[ iGroup ] % PicWidthInMbs
        yBottomRight = bottom_right[ iGroup ] / PicWidthInMbs
        xBottomRight = bottom_right[ iGroup ] % PicWidthInMbs
        for( y = yTopLeft; y <= yBottomRight; y++ )
            for( x = xTopLeft; x <= xBottomRight; x++ )
                mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = iGroup                    (8-20)
    }
```

NOTE – The rectangles may overlap. Slice group 0 contains the macroblocks that are within the rectangle specified by top_left[0] and bottom_right[0]. A slice group having slice group ID greater than 0 and less than num_slice_groups_minus1 contains the macroblocks that are within the specified rectangle for that slice group that are not within the rectangle specified for any slice group having a smaller slice group ID. The slice group with slice group ID equal to num_slice_groups_minus1 contains the macroblocks that are not in the other slice groups.

8.2.2.4 Specification for box-out slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 3.

The map unit to slice group map is generated as specified by:

```
    for( i = 0; i < PicSizeInMapUnits; i++ )
        mapUnitToSliceGroupMap[ i ] = 1
    x = ( PicWidthInMbs – slice_group_change_direction_flag ) / 2
    y = ( PicHeightInMapUnits – slice_group_change_direction_flag ) / 2
    ( leftBound, topBound ) = ( x, y )
    ( rightBound, bottomBound ) = ( x, y )
    ( xDir, yDir ) = ( slice_group_change_direction_flag – 1, slice_group_change_direction_flag )
    for( k = 0; k < MapUnitsInSliceGroup0; k += mapUnitVacant ) {
        mapUnitVacant = ( mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ]  = =  1 )
        if( mapUnitVacant )
            mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = 0                             (8-21)
        if( xDir  = =  –1  &&  x  = =  leftBound ) {
            leftBound = Max( leftBound – 1, 0 )
            x = leftBound
            ( xDir, yDir ) = ( 0, 2 * slice_group_change_direction_flag – 1 )
        } else if( xDir  = =  1  &&  x  = =  rightBound ) {
            rightBound = Min( rightBound + 1, PicWidthInMbs – 1 )
            x = rightBound
            ( xDir, yDir ) = ( 0, 1 – 2 * slice_group_change_direction_flag )
        } else if( yDir  = =  –1  &&  y  = =  topBound ) {
            topBound = Max( topBound – 1, 0 )
            y = topBound
            ( xDir, yDir ) = ( 1 – 2 * slice_group_change_direction_flag, 0 )
        } else if( yDir  = =  1  &&  y  = =  bottomBound ) {
            bottomBound  =  Min( bottomBound + 1, PicHeightInMapUnits – 1 )
            y = bottomBound
            ( xDir, yDir ) = ( 2 * slice_group_change_direction_flag – 1, 0 )
        } else
            ( x, y ) = ( x + xDir, y + yDir )
    }
```

8.2.2.5 Specification for raster scan slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 4.

The map unit to slice group map is generated as specified by:

```
    for( i = 0; i < PicSizeInMapUnits; i++ )
        if( i < sizeOfUpperLeftGroup )
            mapUnitToSliceGroupMap[ i ] = slice_group_change_direction_flag
        else                                                                                (8-22)
            mapUnitToSliceGroupMap[ i ] = 1 – slice_group_change_direction_flag
```

8.2.2.6 Specification for wipe slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 5.

The map unit to slice group map is generated as specified by:

```
k = 0;
for( j = 0; j < PicWidthInMbs; j++ )
    for( i = 0; i < PicHeightInMapUnits; i++ )
        if( k++ < sizeOfUpperLeftGroup )
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = slice_group_change_direction_flag
        else                                                                                    (8-23)
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = 1 – slice_group_change_direction_flag
```

8.2.2.7 Specification for explicit slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 6.

The map unit to slice group map is generated as specified by:

```
mapUnitToSliceGroupMap[ i ] = slice_group_id[ i ]                                               (8-24)
```

for all i ranging from 0 to PicSizeInMapUnits – 1, inclusive.

8.2.2.8 Specification for conversion of map unit to slice group map to macroblock to slice group map

For each value of i ranging from 0 to PicSizeInMbs – 1, inclusive, the macroblock to slice group map is specified as follows.

– If frame_mbs_only_flag is equal to 1 or field_pic_flag is equal to 1, the macroblock to slice group map is specified by:

```
MbToSliceGroupMap[ i ] = mapUnitToSliceGroupMap[ i ]                                            (8-25)
```

– Otherwise, if MbaffFrameFlag is equal to 1, the macroblock to slice group map is specified by:

```
MbToSliceGroupMap[ i ] = mapUnitToSliceGroupMap[ i / 2 ]                                        (8-26)
```

– Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and field_pic_flag is equal to 0), the macroblock to slice group map is specified by:

```
MbToSliceGroupMap[ i ] = mapUnitToSliceGroupMap[ ( i / ( 2 * PicWidthInMbs ) ) * PicWidthInMbs
                                                 + ( i % PicWidthInMbs ) ]                      (8-27)
```

8.2.3 Decoding process for slice data partitioning

Inputs to this process are

– a slice data partition A layer RBSP,

– when syntax elements of category 3 are present in the slice data, a slice data partition B layer RBSP having the same slice_id as in the slice data partition A layer RBSP, and

– when syntax elements of category 4 are present in the slice data, a slice data partition C layer RBSP having the same slice_id as in the slice data partition A layer RBSP.

NOTE 1 – The slice data partition B layer RBSP and slice data partition C layer RBSP need not be present.

Output of this process is a coded slice.

When slice data partitioning is not used, coded slices are represented by a slice layer without partitioning RBSP that contains a slice header followed by a slice data syntax structure that contains all the syntax elements of categories 2, 3, and 4 (see category column in subclause 7.3) of the macroblock data for the macroblocks of the slice.

When slice data partitioning is used, the macroblock data of a slice is partitioned into one to three partitions contained in separate NAL units. Partition A contains a slice data partition A header, and all syntax elements of category 2. Partition B, when present, contains a slice data partition B header and all syntax elements of category 3. Partition C, when present, contains a slice data partition C header and all syntax elements of category 4.

When slice data partitioning is used, the syntax elements of each category are parsed from a separate NAL unit, which need not be present when no symbols of the respective category exist. The decoding process shall process the slice data partitions of a coded slice in a manner equivalent to processing a corresponding slice layer without partitioning RBSP by extracting each syntax element from the slice data partition in which the syntax element appears depending on the slice data partition assignment in the syntax tables in subclause 7.3.

NOTE 2 – Syntax elements of category 3 are relevant to the decoding of residual data of I and SI macroblock types. Syntax elements of category 4 are relevant to the decoding of residual data of P and B macroblock types. Category 2 encompasses all other syntax elements related to the decoding of macroblocks, and their information is often denoted as header information. The slice data partition A header contains all the syntax elements of the slice header, and additionally a slice_id that are used to associate the slice data partitions B and C with the slice data partition A. The slice data partition B and C headers contain the slice_id syntax element that establishes their association with the slice data partition A of the slice.

8.2.4 Decoding process for reference picture lists construction

This process is invoked at the beginning of decoding of each P, SP~~, or~~, B, EP, or EB slice. The decoding of EP and EB slices is specified in Annex G.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause 8.2.5. Short-term reference pictures are identified by the value of frame_num. Long-term reference pictures are assigned a long-term frame index as specified by the bitstream and specified in subclause 8.2.5.

Subclause 8.2.4.1 is invoked to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variable LongTermPicNum to each of the long-term reference pictures.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When decoding a P or SP slice, there is a single reference picture list RefPicList0. When decoding a B slice, there is a second independent reference picture list RefPicList1 in addition to RefPicList0.

At the beginning of decoding of each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as follows.

– An initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.4.2.

– The initial reference picture list RefPicList0 and for B slices RefPicList1 are modified as specified in subclause 8.2.4.3.

NOTE – The reordering process for reference picture lists specified in subclause 8.2.4.3 allows the contents of RefPicList0 and for B slices RefPicList1 to be modified in a flexible fashion. In particular, it is possible for a picture that is currently marked "used for reference" to be inserted into RefPicList0 and for B slices RefPicList1 even when the picture is not in the initial reference picture list derived as specified in subclause 8.2.4.2.

The number of entries in the modified reference picture list RefPicList0 is num_ref_idx_l0_active_minus1 + 1, and for B slices the number of entries in the modified reference picture list RefPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists RefPicList0 or RefPicList1.

8.2.4.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause 8.2.4 or the decoded reference picture marking process specified in subclause 8.2.5 is invoked.

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.4.2, the modification process for reference picture lists in subclause 8.2.4.3, and for the decoded reference picture marking process in subclause 8.2.5.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows. First, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) of the corresponding short-term reference picture. Then the variable FrameNumWrap is derived as

```
if( FrameNum > frame_num )
    FrameNumWrap = FrameNum – MaxFrameNum                    (8-28)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-28 is the frame_num in the slice header(s) for the current picture.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5).

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current picture and they are set as follows.

– If field_pic_flag is equal to 0, the following applies.
  – For each short-term reference frame or complementary reference field pair:

    PicNum = FrameNumWrap (8-29)

  – For each long-term reference frame or long-term complementary reference field pair:

    LongTermPicNum = LongTermFrameIdx (8-30)

    NOTE – When decoding a frame the value of MbaffFrameFlag has no influence on the derivations in subclauses 8.2.4.2, 8.2.4.3, and 8.2.5.

– Otherwise (field_pic_flag is equal to 1), the following applies.
  – For each short-term reference field the following applies.
    – If the reference field has the same parity as the current field

      PicNum = 2 * FrameNumWrap + 1 (8-31)

    – Otherwise (the reference field has the opposite parity of the current field),

      PicNum = 2 * FrameNumWrap (8-32)

  – For each long-term reference field the following applies.
    – If the reference field has the same parity as the current field

      LongTermPicNum = 2 * LongTermFrameIdx + 1 (8-33)

    – Otherwise (the reference field has the opposite parity of the current field),

      LongTermPicNum = 2 * LongTermFrameIdx (8-34)

8.2.4.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P, SP, ~~or~~ B, EP, or EB slice header. The decoding of EP and EB slices is specified in Annex G.

RefPicList0 and RefPicList1 have initial entries as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is greater than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1,

respectively, the extra entries past position num_ref_idx_l0_active_minus1 or num_ref_idx_l1_active_minus1 are discarded from the initial reference picture list.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.4.2.1 through 8.2.4.2.5 is less than num_ref_idx_l0_active_minus1 + 1 or num_ref_idx_l1_active_minus1 + 1, respectively, the remaining entries in the initial reference picture list are set equal to "no reference picture".

8.2.4.2.1 Initialisation process for the reference picture list for P and SP slices in frames

This initialisation process is invoked when decoding a P ~~or~~ SP, or EP slice in a coded frame. The decoding of EP slices is specified in Annex G.

When this process is invoked, there shall be at least one reference frame or complementary reference field pair that is currently marked as "used for short-term reference" or "used for long-term reference".

The reference picture list RefPicList0 is ordered so that short-term reference frames and short-term complementary reference field pairs have lower indices than long-term reference frames and long-term complementary reference field pairs.

The short-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the highest PicNum value and proceeding through in descending order to the frame or complementary field pair with the lowest PicNum value.

The long-term reference frames and complementary reference field pairs are ordered starting with the frame or complementary field pair with the lowest LongTermPicNum value and proceeding through in ascending order to the frame or complementary field pair with the highest LongTermPicNum value.

NOTE – A non-paired reference field is not used for inter prediction for decoding a frame, regardless of the value of MbaffFrameFlag.

For example, when three reference frames are marked as "used for short-term reference" with PicNum equal to 300, 302, and 303 and two reference frames are marked as "used for long-term reference" with LongTermPicNum equal to 0 and 3, the initial index order is:

– RefPicList0[0] is set equal to the short-term reference picture with PicNum = 303,

– RefPicList0[1] is set equal to the short-term reference picture with PicNum = 302,

– RefPicList0[2] is set equal to the short-term reference picture with PicNum = 300,

– RefPicList0[3] is set equal to the long-term reference picture with LongTermPicNum = 0, and

– RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

8.2.4.2.2 Initialisation process for the reference picture list for P and SP slices in fields

This initialisation process is invoked when decoding a P, ~~or~~ SP, or EP slice in a coded field. The decoding of EP slices is specified in Annex G.

Each field included in the reference picture list RefPicList0 has a separate index in the reference picture list RefPicList0.

NOTE – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Two ordered lists of reference frames, refFrameList0ShortTerm and refFrameList0LongTerm, are derived as follows. For purposes of the formation of this list of frames, decoded reference frames, complementary reference field pairs, non-paired reference fields and reference frames in which a single field is marked "used for short-term reference" or "used for long-term reference" are all considered reference frames.

– All frames having one or more fields marked "used for short-term reference" are included in the list of short-term reference frames refFrameList0ShortTerm. When the current field is the second field (in decoding order) of a complementary reference field pair and the first field is marked as "used for short-term reference", the first field is included in the list of short-term reference frames refFrameList0ShortTerm. refFrameList0ShortTerm is ordered starting with the reference frame with the highest FrameNumWrap value and proceeding through in descending order to the reference frame with the lowest FrameNumWrap value.

– All frames having one or more fields marked "used for long-term reference" are included in the list of long-term reference frames refFrameList0LongTerm. When the current field is the second field (in decoding order) of a

complementary reference field pair and the first field is marked as "used for long-term reference, the first field is included in the list of long-term reference frames refFrameList0LongTerm. refFrameList0LongTerm is ordered starting with the reference frame with the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference frame with the highest LongTermFrameIdx value.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameList0LongTerm given as input and the output is assigned to RefPicList0.

8.2.4.2.3 Initialisation process for reference picture lists for B slices in frames

This initialisation process is invoked when decoding a B or EB slice in a coded frame. The decoding of EB slices is specified in Annex G.

For purposes of the formation of the reference picture lists RefPicList0 and RefPicList1 the term reference entry refers in the following to decoded reference frames or complementary reference field pairs.

When this process is invoked, there shall be at least one reference entry that is currently marked as "used for short-term reference" or "used for long-term reference".

For B slices, the order of short-term reference entries in the reference picture lists RefPicList0 and RefPicList1 depends on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

The reference picture list RefPicList0 is ordered such that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList0 in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList0 in ascending order of PicOrderCnt(entryShortTerm).

– The long-term reference entries are ordered starting with the long-term reference entry that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

The reference picture list RefPicList1 is ordered so that short-term reference entries have lower indices than long-term reference entries. It is ordered as follows.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refPicList1 in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refPicList1 in descending order of PicOrderCnt(entryShortTerm).

– Long-term reference entries are ordered starting with the long-term reference frame or complementary reference field pair that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.

– When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

NOTE 2 – A non-paired reference field is not used for inter prediction of frames (independent of the value of MbaffFrameFlag).

8.2.4.2.4 Initialisation process for reference picture lists for B slices in fields

This initialisation process is invoked when decoding a B or EB slice in a coded field. The decoding of EP and EB slices is specified in Annex G.

When decoding a field, each field of a stored reference frame is identified as a separate reference picture with a unique index. The order of short-term reference pictures in the reference picture lists RefPicList0 and RefPicList1 depend on output order, as given by PicOrderCnt( ). When pic_order_cnt_type is equal to 0, reference pictures that are marked as "non-existing" as specified in subclause 8.2.5.2 are not included in either RefPicList0 or RefPicList1.

NOTE 1 – When gaps_in_frame_num_value_allowed_flag is equal to 1, encoders should use reference picture list reordering to ensure proper operation of the decoding process (particularly when pic_order_cnt_type is equal to 0, in which case PicOrderCnt( ) is not inferred for "non-existing" frames).

NOTE 2 – When decoding a field, there are effectively at least twice as many pictures available for referencing as there would be when decoding a frame at the same position in decoding order.

Three ordered lists of reference frames, refFrameList0ShortTerm, refFrameList1ShortTerm and refFrameListLongTerm, are derived as follows. For purposes of the formation of these lists of frames the term reference entry refers in the following to decoded reference frames, complementary reference field pairs, or non-paired reference fields. When pic_order_cnt_type is equal to 0, the term reference entry does not refer to frames that are marked as "non-existing" as specified in subclause 8.2.5.2.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) less than or equal to PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList0ShortTerm in descending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList0ShortTerm in ascending order of PicOrderCnt(entryShortTerm).

  NOTE 3 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList0ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

– Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference". When some values of entryShortTerm are present having PicOrderCnt(entryShortTerm) greater than PicOrderCnt(CurrPic), these values of entryShortTerm are placed at the beginning of refFrameList1ShortTerm in ascending order of PicOrderCnt(entryShortTerm). All of the remaining values of entryShortTerm (when present) are then appended to refFrameList1ShortTerm in descending order of PicOrderCnt(entryShortTerm).

  NOTE 4 – When the current field follows in decoding order a coded field fldPrev with which together it forms a complementary reference field pair, fldPrev is included into the list refFrameList1ShortTerm using PicOrderCnt(fldPrev) and the ordering method described in the previous sentence is applied.

– refFrameListLongTerm is ordered starting with the reference entry having the lowest LongTermFrameIdx value and proceeding through in ascending order to the reference entry having highest LongTermFrameIdx value.

NOTE 5 – When the complementary field of the current picture is marked "used for long-term reference" it is included into the list refFrameListLongTerm. A reference entry in which only one field is marked as “used for long-term reference” is included into the list refFrameListLongTerm.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList0ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList0.

The process specified in subclause 8.2.4.2.5 is invoked with refFrameList1ShortTerm and refFrameListLongTerm given as input and the output is assigned to RefPicList1.

When the reference picture list RefPicList1 has more than one entry and RefPicList1 is identical to the reference picture list RefPicList0, the first two entries RefPicList1[0] and RefPicList1[1] are switched.

8.2.4.2.5 Initialisation process for reference picture lists in fields

Inputs of this process are the reference frame lists refFrameListXShortTerm (with X may be 0 or 1) and refFrameListLongTerm.

The reference picture list RefPicListX is a list ordered such that short-term reference fields have lower indices than long-term reference fields. Given the reference frame lists refFrameListXShortTerm and refFrameListLongTerm, it is derived as follows.

– Short-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListXShortTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as “used for short-term reference”, the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListXShortTerm is inserted into RefPicListX. When there are no more short-term reference fields of the alternate parity in the ordered list of frames refFrameListXShortTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListXShortTerm.

– Long-term reference fields are ordered by selecting reference fields from the ordered list of frames refFrameListLongTerm by alternating between fields of differing parity, starting with a field that has the same parity as the current field (when present). When one field of a reference frame was not decoded or is not marked as "used for long-term reference", the missing field is ignored and instead the next available stored reference field of the chosen parity from the ordered list of frames refFrameListLongTerm is inserted into RefPicListX. When there are no more long-term reference fields of the alternate parity in the ordered list of frames refFrameListLongTerm, the next not yet indexed fields of the available parity are inserted into RefPicListX in the order in which they occur in the ordered list of frames refFrameListLongTerm.

8.2.4.3 Reordering process for reference picture lists

When ref_pic_list_reordering_flag_l0 is equal to 1, the following applies.

– Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.

– The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.

  – If reordering_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

  – Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

  – Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList0 is finished.

When ref_pic_list_reordering_flag_l1 is equal to 1, the following applies.

– Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.

– The corresponding syntax elements reordering_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.

  – If reordering_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.4.3.1 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.

  – Otherwise, if reordering_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.4.3.2 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.

  – Otherwise (reordering_of_pic_nums_idc is equal to 3), the reordering process for reference picture list RefPicList1 is finished.

8.2.4.3.1 Reordering process of reference picture lists for short-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The variable picNumLXNoWrap is derived as follows.

– If reordering_of_pic_nums_idc is equal to 0

```
if( picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) < 0 )
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 ) + MaxPicNum        (8-35)
else
    picNumLXNoWrap = picNumLXPred – ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (reordering_of_pic_nums_idc is equal to 1),

```
if( picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )  >=  MaxPicNum )
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 ) – MaxPicNum        (8-36)
else
    picNumLXNoWrap = picNumLXPred + ( abs_diff_pic_num_minus1 + 1 )
```

picNumLXPred is the prediction value for the variable picNumLXNoWrap. When the process specified in this subclause is invoked the first time for a slice (that is, for the first occurrence of reordering_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_reordering( ) syntax), picNumL0Pred and picNumL1Pred are initially set equal to CurrPicNum. After each assignment of picNumLXNoWrap, the value of picNumLXNoWrap is assigned to picNumLXPred.

The variable picNumLX is derived as follows

```
if( picNumLXNoWrap > CurrPicNum )
    picNumLX = picNumLXNoWrap – MaxPicNum                    (8-37)
else
    picNumLX = picNumLXNoWrap
```

picNumLX shall be equal to the PicNum of a reference picture that is marked as “used for short-term reference” and shall not be equal to the PicNum of a short-term reference picture that is marked as "non-existing".

The following procedure is conducted to place the picture with short-term picture number picNumLX into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = short-term reference picture with PicNum equal to picNumLX
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )    (8-38)
    if( PicNumF( RefPicListX[ cIdx ] ) != picNumLX )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function PicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for short-term reference", PicNumF(RefPicListX[cIdx]) is the PicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for short-term reference"), PicNumF(RefPicListX[cIdx]) is equal to MaxPicNum.

NOTE 1 – A value of MaxPicNum can never be equal to picNumLX.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.4.3.2 Reordering process of reference picture lists for long-term reference pictures

Input to this process is an index refIdxLX (with X being 0 or 1).

Output of this process is an incremented index refIdxLX.

The following procedure is conducted to place the picture with long-term picture number long_term_pic_num into the index position refIdxLX, shift the position of any other remaining pictures to later in the list, and increment the value of refIdxLX.

```
for( cIdx = num_ref_idx_lX_active_minus1 + 1; cIdx > refIdxLX; cIdx-- )
    RefPicListX[ cIdx ] = RefPicListX[ cIdx – 1]
RefPicListX[ refIdxLX++ ] = long-term reference picture with LongTermPicNum equal to long_term_pic_num
nIdx = refIdxLX
for( cIdx = refIdxLX; cIdx <= num_ref_idx_lX_active_minus1 + 1; cIdx++ )    (8-39)
    if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the function LongTermPicNumF(RefPicListX[cIdx]) is derived as follows:

– If the picture RefPicListX[cIdx] is marked as "used for long-term reference", LongTermPicNumF(RefPicListX[cIdx]) is the LongTermPicNum of the picture RefPicListX[cIdx].

– Otherwise (the picture RefPicListX[cIdx] is not marked as "used for long-term reference"), LongTermPicNumF(RefPicListX[cIdx]) is equal to 2 * (MaxLongTermFrameIdx + 1).

NOTE 1 – A value of 2 * (MaxLongTermFrameIdx + 1) can never be equal to long_term_pic_num.

NOTE 2 – Within this pseudo-code procedure, the length of the list RefPicListX is temporarily made one element longer than the length needed for the final list. After the execution of this procedure, only elements 0 through num_ref_idx_lX_active_minus1 of the list need to be retained.

8.2.5 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

NOTE – The decoding process for gaps in frame_num that is specified in subclause 8.2.5.2 may also be invoked when nal_ref_idc is equal to 0, as specified in clause 8.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as “used for short-term reference” or "used for long-term reference". For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity.

Frames or complementary field pairs marked as “used for short-term reference” or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as “unused for reference”. A field marked as “used for short-term reference” or as "used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as “unused for reference”.

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and a long-term reference picture is identified for use in the decoding process by its long-term picture number LongTermPicNum. When the current picture is not an IDR picture, subclause 8.2.4.1 is invoked to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum.

8.2.5.1 Sequence of operations for decoded reference picture marking process

Decoded reference picture marking proceeds in the following ordered steps.

1. All slices of the current picture are decoded.
2. Depending on whether the current picture is an IDR picture, the following applies.
   - If the current picture is an IDR picture, the following applies.
     - All reference pictures are marked as "unused for reference"
     - Depending on long_term_reference_flag, the following applies.
       - If long_term_reference_flag is equal to 0, the IDR picture is marked as "used for short-term reference" and MaxLongTermFrameIdx is set equal to “no long-term frame indices”.
       - Otherwise (long_term_reference_flag is equal to 1), the IDR picture is marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture is set equal to 0, and MaxLongTermFrameIdx is set equal to 0.
   - Otherwise (the current picture is not an IDR picture), the following applies.
     - If adaptive_ref_pic_marking_mode_flag is equal to 0, the process specified in subclause 8.2.5.3 is invoked.
     - Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.5.4 is invoked.
3. When the current picture is not an IDR picture and it was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

After marking the current decoded reference picture, the total number of frames with at least one field marked as “used for reference”, plus the number of complementary field pairs with at least one field marked as “used for reference”, plus the number of non-paired fields marked as “used for reference” shall not be greater than Max(num_ref_frames, 1).

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE 1 – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this Recommendation | International Standard are historical.

NOTE 2 – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

When this process is invoked, a set of values of frame_num pertaining to “non-existing” pictures is derived as all values taken on by UnusedShortTermFrameNum in Equation 7-21 except the value of frame_num for the current picture.

The decoding process generates and marks a frame for each of the values of frame_num pertaining to “non-existing” pictures, in the order in which the values of UnusedShortTermFrameNum are generated by Equation 7-21, using the “sliding window” picture marking process as specified in subclause 8.2.5.3. The generated frames are also marked as “non-existing” and “used for short-term reference”. The sample values of the generated frames may be set to any value. The bitstream shall not contain data that results in a reference to these generated frames which are marked as “non-existing” in the inter prediction process, a reference to these frames in the reordering commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or a reference to these frames in the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3).

When pic_order_cnt_type is not equal to 0, TopFieldOrderCnt and BottomFieldOrderCnt are derived for each of the "non-existing" frames by invoking the decoding process for picture order count in subclause 8.2.1. When invoking the process in subclause 8.2.1 for a particular "non-existing" frame, the current picture is considered to be a picture considered having frame_num inferred to be equal to UnusedShortTermFrameNum, nal_ref_idc inferred to be not equal to 0, nal_unit_type inferred to be not equal to 5, field_pic_flag inferred to be equal to 0, adaptive_ref_pic_marking_mode_flag inferred to be equal to 0, delta_pic_order_cnt[0] (if needed) inferred to be equal to 0, and delta_pic_order_cnt[1] (if needed) inferred to be equal to 0.

NOTE 3 – The decoding process should infer an unintentional picture loss when any of these values of frame_num pertaining to “non-existing” pictures is referred to in the inter prediction process, is referred to in the reordering commands for reference picture lists for short-term reference pictures (subclause 8.2.4.3.1), or is referred to in the assignment process of a LongTermFrameIdx to a short-term reference picture (subclause 8.2.5.4.3). The decoding process should not infer an unintentional picture loss when a memory management control operation not equal to 3 is applied to a frame marked as “non-existing”.

8.2.5.3 Sliding window decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 0.

Depending on the properties of the current picture as specified below, the following applies.

- If the current picture is a coded field that is the second field in decoding order of a complementary reference field pair, and the first field has been marked as “used for short-term reference”, the current picture is also marked as “used for short-term reference”.
- Otherwise, the following applies.
  - Let numShortTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as “used for short-term reference”. Let numLongTerm be the total number of reference frames, complementary reference field pairs and non-paired reference fields for which at least one field is marked as “used for long-term reference”.
  - When numShortTerm + numLongTerm is equal to Max(num_ref_frames, 1), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference frame, complementary reference field pair or non-paired reference field that has the smallest value of FrameNumWrap is marked as “unused for reference”. When it is a frame or a complementary field pair, both of its fields are also marked as “unused for reference”.

8.2.5.4 Adaptive memory control decoded reference picture marking process

This process is invoked when adaptive_ref_pic_marking_mode_flag is equal to 1.

The memory_management_control_operation commands with values of 1 to 6 are processed in the order they occur in the bitstream after the current picture has been decoded. For each of these memory_management_control_operation commands, one of the processes specified in subclauses 8.2.5.4.1 to 8.2.5.4.5 is invoked depending on the value of memory_management_control_operation. The memory_management_control_operation command with value of 0 specifies the end of memory_management_control_operation commands.

Memory management control operations are applied to pictures as follows.

– If field_pic_flag is equal to 0, memory_management_control_operation commands are applied to the frames or complementary reference field pairs specified.

– Otherwise (field_pic_flag is equal to 1), memory_management_control_operation commands are applied to the individual reference fields specified.

8.2.5.4.1 Marking process of a short-term reference picture as “unused for reference”

This process is invoked when memory_management_control_operation is equal to 1.

Let picNumX be specified by

$$picNumX = CurrPicNum - (difference_of_pic_nums_minus1 + 1). \quad (8\text{-}40)$$

Depending on field_pic_flag the value of picNumX is used to mark a short-term reference picture as “unused for reference” as follows.

– If field_pic_flag is equal to 0, the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are marked as “unused for reference”.

– Otherwise (field_pic_flag is equal to 1), the short-term reference field specified by picNumX is marked as “unused for reference”. When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.5.4.2 Marking process of a long-term reference picture as “unused for reference”

This process is invoked when memory_management_control_operation is equal to 2.

Depending on field_pic_flag the value of LongTermPicNum is used to mark a long-term reference picture as “unused for reference” as follows.

– If field_pic_flag is equal to 0, the long-term reference frame or long-term complementary reference field pair having LongTermPicNum equal to long_term_pic_num and both of its fields are marked as “unused for reference”.

– Otherwise (field_pic_flag is equal to 1), the long-term reference field specified by LongTermPicNum equal to long_term_pic_num is marked as “unused for reference”. When that reference field is part of a reference frame or a complementary reference field pair, the frame or complementary field pair is also marked as "unused for reference", but the marking of the other field is not changed.

8.2.5.4.3 Assignment process of a LongTermFrameIdx to a short-term reference picture

This process is invoked when memory_management_control_operation is equal to 3.

Given the syntax element difference_of_pic_nums_minus1, the variable picNumX is obtained as specified in subclause 8.2.5.4.1. picNumX shall refer to a frame or complementary reference field pair or non-paired reference field marked as "used for short-term reference" and not marked as "non-existing".

When LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the picture specified by picNumX, that field is marked as “unused for reference”.

Depending on field_pic_flag the value of LongTermFrameIdx is used to mark a picture from "used for short-term reference" to "used for long-term reference" as follows.

– If field_pic_flag is equal to 0, the marking of the short-term reference frame or short-term complementary reference field pair specified by picNumX and both of its fields are changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

– Otherwise (field_pic_flag is equal to 1), the marking of the short-term reference field specified by picNumX is changed from "used for short-term reference" to "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx. When the field is part of a reference frame or a complementary reference field pair, and the other field of the same reference frame or complementary reference field pair is also marked as "used for long-term reference", the reference frame or complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

8.2.5.4.4 Decoding process for MaxLongTermFrameIdx

This process is invoked when memory_management_control_operation is equal to 4.

All pictures for which LongTermFrameIdx is greater than max_long_term_frame_idx_plus1 – 1 and that are marked as "used for long-term reference" are marked as “unused for reference”.

The variable MaxLongTermFrameIdx is derived as follows.

– If max_long_term_frame_idx_plus1 is equal to 0, MaxLongTermFrameIdx is set equal to “no long-term frame indices”.

– Otherwise (max_long_term_frame_idx_plus1 is greater than 0), MaxLongTermFrameIdx is set equal to max_long_term_frame_idx_plus1 – 1.

NOTE – The memory_management_control_operation command equal to 4 can be used to mark long-term reference pictures as “unused for reference”. The frequency of transmitting max_long_term_frame_idx_plus1 is not specified by this Recommendation | International Standard. However, the encoder should send a memory_management_control_operation command equal to 4 upon receiving an error message, such as an intra refresh request message.

8.2.5.4.4.1 Marking process of all reference pictures as “unused for reference” and setting MaxLongTermFrameIdx to “no long-term frame indices”

This process is invoked when memory_management_control_operation is equal to 5.

All reference pictures are marked as “unused for reference” and the variable MaxLongTermFrameIdx is set equal to “no long-term frame indices”.

8.2.5.4.5 Process for assigning a long-term frame index to the current picture

This process is invoked when memory_management_control_operation is equal to 6.

When a variable LongTermFrameIdx equal to long_term_frame_idx is already assigned to a long-term reference frame or a long-term complementary reference field pair, that frame or complementary field pair and both of its fields are marked as "unused for reference". When LongTermFrameIdx is already assigned to a non-paired reference field, and the field is not the complementary field of the current picture, that field is marked as “unused for reference”.

The current picture is marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 0, both its fields are also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

When field_pic_flag is equal to 1 and the current picture is the second field (in decoding order) of a complementary reference field pair, and the first field of the complementary reference field pair is also currently marked as "used for long-term reference), the complementary reference field pair is also marked as "used for long-term reference" and assigned LongTermFrameIdx equal to long_term_frame_idx.

After marking the current decoded reference picture, the total number of frames with at least one field marked as “used for reference”, plus the number of complementary field pairs with at least one field marked as “used for reference”, plus the number of non-paired fields marked as “used for reference” shall not be greater than Max(num_ref_frames, 1).

NOTE – Under some circumstances, the above statement may impose a constraint on the order in which a memory_management_control_operation syntax element equal to 6 can appear in the decoded reference picture marking syntax relative to a memory_management_control_operation syntax element equal to 1, 2, or 4.

8.3 Intra prediction process

This process is invoked for I and SI macroblock types.

Inputs to this process are constructed samples prior to the deblocking filter process and, for Intra_NxN prediction modes (where NxN is equal to 4x4 or 8x8), the values of IntraNxNPredMode from neighbouring macroblocks.

Outputs of this process are specified as follows.

– If the macroblock prediction mode is Intra_4x4 or Intra_8x8, the outputs are constructed luma samples prior to the deblocking filter process and (when chroma_format_idc is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise, if mb_type is not equal to I_PCM, the outputs are luma prediction samples of the macroblock $pred_L$ and (when chroma_format_idc is not equal to 0) chroma prediction samples of the macroblock $pred_C$, where C is equal to Cb and Cr.

– Otherwise (mb_type is equal to I_PCM), the outputs are constructed luma and (when chroma_format_idc is not equal to 0) chroma samples prior to the deblocking filter process.

The variable MvCnt is set equal to 0.

Depending on the value of mb_type the following applies.

– If mb_type is equal to I_PCM, the sample construction process for I_PCM macroblocks as specified in subclause 8.3.5 is invoked with S'_L and, when chromat_format_idc is not equal to 0, S'_{Cb} and S'_{Cr} as inputs.

– Otherwise (mb_type is not equal to I_PCM), the following applies.

  – The decoding processes for Intra prediction modes are described for the luma component as follows.

    – If the macroblock prediction mode is equal to Intra_4x4, the ~~specification~~ Intra_4x4 prediction process for luma samples as specified in subclause 8.3.1 ~~applies~~is invoked.

    – Otherwise, if the macroblock prediction mode is equal to Intra_8x8, the Intra_8x8 prediction process for luma samples ~~specification~~ as specified in subclause 8.3.2 ~~applies~~is invoked.

    – Otherwise (the macroblock prediction mode is equal to Intra_16x16), the ~~specification~~ Intra_16x16 prediction process for luma samples as specified in subclause 8.3.3 is invoked with S'_L as the input ~~applies~~ and $pred_L$ as the output.

  – When chroma_format_idc is not equal to 0, ~~The~~ the ~~decoding processes for~~ Intra prediction process for chroma samples as specified ~~modes for the chroma components are described~~ in subclause 8.3.4 is invoked with S'_{Cb} and S'_{Cr} as inputs and $pred_{Cb}$ and $pred_{Cr}$ as outputs. ~~This process is only invoked when chroma_format_idc is not equal to 0 (monochrome).~~

带格式的: 项目符号和编号

Samples used in the Intra prediction process are the sample values prior to alteration by any deblocking filter operation.

8.3.1 Intra_4x4 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_4x4.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from neighbouring macroblocks or macroblock pairs.

The luma component of a macroblock consists of 16 blocks of 4x4 luma samples. These blocks are inverse scanned using the 4x4 luma block inverse scanning process as specified in subclause 6.4.3.

For all 4x4 luma blocks of the luma component of a macroblock with luma4x4BlkIdx = 0..15, the derivation process for the Intra4x4PredMode as specified in subclause 8.3.1.1 is invoked with luma4x4BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra4x4PredMode[luma4x4BlkIdx] as the output.

For each luma block of 4x4 samples indexed using luma4x4BlkIdx = 0..15,

1. The Intra_4x4 sample prediction process in subclause 8.3.1.2 is invoked with luma4x4BlkIdx, Intra4x4PredMode, and the array S'_L of constructed luma samples prior ~~(in decoding order)~~ to the deblocking filter process ~~from adjacent luma blocks~~ as the input and the output are the Intra_4x4 luma prediction samples $pred4x4_L$[x, y] with x, y = 0..3.

2. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO) and x, y = 0..3.

$$pred_L[\ xO + x, yO + y\ ] = pred4x4_L[\ x, y\ ] \quad (8\text{-}41)$$

3. The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma4x4BlkIdx as the input and the constructed samples for the current 4x4 luma block S'_L as the output.

8.3.1.1 Derivation process for the Intra4x4PredMode

Inputs to this process are the index of the 4x4 luma block luma4x4BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra4x4PredMode[luma4x4BlkIdx].

Table 8-2 specifies the values for Intra4x4PredMode[luma4x4BlkIdx] and the associated names.

Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names

Intra4x4PredMode[luma4x4BlkIdx]	Name of Intra4x4PredMode[luma4x4BlkIdx]
0	Intra_4x4_Vertical (prediction mode)
1	Intra_4x4_Horizontal (prediction mode)
2	Intra_4x4_DC (prediction mode)
3	Intra_4x4_Diagonal_Down_Left (prediction mode)
4	Intra_4x4_Diagonal_Down_Right (prediction mode)
5	Intra_4x4_Vertical_Right (prediction mode)
6	Intra_4x4_Horizontal_Down (prediction mode)
7	Intra_4x4_Vertical_Left (prediction mode)
8	Intra_4x4_Horizontal_Up (prediction mode)

Intra4x4PredMode[luma4x4BlkIdx] labelled 0, 1, 3, 4, 5, 6, 7, and 8 represent directions of predictions as illustrated in Figure 8-1.

Figure 8-1 – Intra_4x4 prediction mode directions (informative)

Intra4x4PredMode[luma4x4BlkIdx] is derived as follows.

– The process specified in subclause 6.4.8.3 is invoked with luma4x4BlkIdx given as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.

– The variables intra4x4PredModeN and intra8x8PredModeN, with N being replaced by A or B, are derived as follows.

  – If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode, intra4x4PredModeN is set equal to the variable array Intra4x4PredMode which is assigned to the macroblock mbAddrN, and intra8x8PredModeN is marked as "not available".

  – Otherwise, if the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode, intra8x8PredModeN is set equal to the variable array Intra8x8PredMode which is assigned to the macroblock mbAddrN, and intra4x4PredModeN is marked as "not available".

  – Otherwise (the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode), intra4x4PredModeN and intra8x8PredModeN are marked as "not available".

– The derivation process for Intra_4x4 prediction modes as specified in subclause 8.3.1.1.1 is invoked with mbAddrA, mbAddrB, luma4x4BlkIdxA, luma4x4BlkIdxB, intra4x4PredModeA, intra4x4PredModeB, intra8x8PredModeA, and intra8x8PredModeB as inputs and the output is assigned to Intra4x4PredMode[luma4x4BlkIdx].

8.3.1.1.1 Derivation process for Intra_4x4 prediction modes

Inputs to this process are

– two macroblock addresses mbAddrA and mbAddrB

– two luma 4x4 block indices luma4x4BlkIdxA and luma4x4BlkIdxB

– two lists of intra 4x4 prediction modes intra4x4PredModeA and intra4x4PredModeB

– two lists of intra 4x4 prediction modes intra8x8PredModeA and intra8x8PredModeB

Output of this process is the variable currIntra4x4PredMode

The variable currIntra4x4PredMode is derived as follows.

– The variable dcPredModePredictedFlag is derived as follows.

  – If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1

    – the macroblock with address mbAddrA is not available

    – the macroblock with address mbAddrB is not available

带格式的: 项目符号和编号

带格式的: 项目符号和编号

  - the macroblock with address mbAddrA is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
  - the macroblock with address mbAddrB is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1
- Otherwise, dcPredModePredictedFlag is set equal to 0.

– For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows.

- If dcPredModePredictedFlag is equal to 1 or the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode, intraMxMPredModeN is set equal to 2 (Intra_4x4_DC prediction mode).
- Otherwise (dcPredModePredictedFlag is equal to 0 and (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode or the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode)), the following applies.
  - If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock mode, intraMxMPredModeN is set equal to ~~Intra4x4PredMode~~intra4x4PredModeN[luma4x4BlkIdxN]~~, where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN~~.
  - Otherwise (the macroblock with address mbAddrN is coded in Intra_8x8 macroblock mode), intraMxMPredModeN is set equal to ~~Intra8x8PredMode~~intra8x8PredModeN[luma4x4BlkIdxN >> 2]~~, where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN~~.

– The variable currIntra4x4PredMode~~[luma4x4BlkIdx]~~ is derived by applying the following procedure.

```
predIntra4x4PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra4x4_pred_mode_flag[ luma4x4BlkIdx ] )
    currIntra4x4PredMode[ luma4x4BlkIdx ] = predIntra4x4PredMode
else                                                                    (8-42)
    if( rem_intra4x4_pred_mode[ luma4x4BlkIdx ] < predIntra4x4PredMode )
        currIntra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ]
    else
        currIntra4x4PredMode[ luma4x4BlkIdx ] = rem_intra4x4_pred_mode[ luma4x4BlkIdx ] + 1
```

8.3.1.2 Intra_4x4 sample prediction

This process is invoked for each 4x4 luma block of a macroblock with prediction mode equal to Intra_4x4 followed by the transform decoding process and picture construction process prior to deblocking for each 4x4 luma block.

Inputs to this process ~~is~~ are

– the index of a 4x4 luma block luma4x4BlkIdx~~.~~

– an array Intra4x4PredMode specifying the Intra_4x4 prediction modes for the current macroblock

– an (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring blocks

Output of this process are the prediction samples pred4x4$_L$[x, y], with x, y = 0..3 for the 4x4 luma block with index luma4x4BlkIdx.

The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

The 13 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..3 and x = 0..7, y = -1, are derived as follows.

– The luma location (xN, yN) is specified by

$$xN = xO + x \qquad (8\text{-}43)$$

带格式的: 项目符号和编号

带格式的: 项目符号和编号

$$yN = yO + y \tag{8-44}$$

– The derivation process for neighbouring locations in subclause 6.4.9 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..3 and x = 0..7, y = -1 is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_4x4 prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1.

    – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI.

    – x is greater than 3 and luma4x4BlkIdx is equal to 3 or 11

  – Otherwise, the sample p[x, y] is marked as "available for Intra_4x4 prediction" and ~~the luma sample at luma location (xW, yW) inside the macroblock mbAddrN is assigned to p[x, y].~~the value of the sample p[x, y] is derived as follows.

    – The position of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as input and the output being assigned to (xM, yM).

    – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

      – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[x, y] = cS_I[xM + xW, yM + 2 * yW] \tag{8-45}$$

      – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[x, y] = cS_I[xM + xW, yM + yW] \tag{8-46}$$

带格式的: 项目符号和编号

带格式的: 项目符号和编号

When samples p[x, -1], with x = 4..7 are marked as "not available for Intra_4x4 prediction," and the sample p[3, -1] is marked as "available for Intra_4x4 prediction," the sample value of p[3, -1] is substituted for sample values p[x, -1], with x = 4..7 and samples p[x, -1], with x = 4..7 are marked as "available for Intra_4x4 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

Depending on Intra4x4PredMode[luma4x4BlkIdx], one of the Intra_4x4 prediction modes specified in subclauses 8.3.1.2.1 to 8.3.1.2.9 is invoked.

8.3.1.2.1 Specification of Intra_4x4_Vertical prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, -1] with x = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = p[x, -1], \text{ with } x, y = 0..3 \tag{8-47}$$

8.3.1.2.2 Specification of Intra_4x4_Horizontal prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 1.

This mode shall be used only when the samples p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = p[-1, y], \text{ with } x,y = 0..3 \quad (8\text{-}48)$$

8.3.1.2.3 Specification of Intra_4x4_DC prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 2.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If all samples p[x, -1], with x = 0..3 and p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 4) >> 3 \quad (8\text{-}49)$$

– Otherwise, if any samples p[x, -1], with x = 0..3 are marked as "not available for Intra_4x4 prediction" and all samples p[-1, y], with y = 0..3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (p[-1, 0] + p[-1, 1] + p[-1, 2] + p[-1, 3] + 2) >> 2 \quad (8\text{-}50)$$

– Otherwise, if any samples p[-1, y], with y = 0..3 are marked as "not available for Intra_4x4 prediction" and all samples p[x, -1], with x = 0 .. 3 are marked as "available for Intra_4x4 prediction", the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0 .. 3 are derived by

$$pred4x4_L[x, y] = (p[0, -1] + p[1, -1] + p[2, -1] + p[3, -1] + 2) >> 2 \quad (8\text{-}51)$$

– Otherwise (some samples p[x, -1], with x = 0..3 and some samples p[-1, y], with y = 0..3 are marked as "not available for Intra_4x4 prediction"), the values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived by

$$pred4x4_L[x, y] = (1 << (BitDepth_Y - 1)) \quad (8\text{-}52)$$

NOTE – A 4x4 luma block can always be predicted using this mode.

8.3.1.2.4 Specification of Intra_4x4_Diagonal_Down_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If x is equal to 3 and y is equal to 3,

$$pred4x4_L[x, y] = (p[6, -1] + 3 * p[7, -1] + 2) >> 2 \quad (8\text{-}53)$$

– Otherwise (x is not equal to 3 or y is not equal to 3),

$$pred4x4_L[x, y] = (p[x + y, -1] + 2 * p[x + y + 1, -1] + p[x + y + 2, -1] + 2) >> 2 \quad (8\text{-}54)$$

8.3.1.2.5 Specification of Intra_4x4_Diagonal_Down_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If x is greater than y,

$$pred4x4_L[x, y] = (p[x - y - 2, -1] + 2 * p[x - y - 1, -1] + p[x - y, -1] + 2) >> 2 \quad (8\text{-}55)$$

– Otherwise if x is less than y,

$$pred4x4_L[x, y] = (p[-1, y - x - 2] + 2 * p[-1, y - x - 1] + p[-1, y - x] + 2) >> 2 \quad (8\text{-}56)$$

– Otherwise (x is equal to y),

$$pred4x4_L[x, y] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 \quad (8\text{-}57)$$

8.3.1.2.6 Specification of Intra_4x4_Vertical_Right prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zVR be set equal to 2 * x – y.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3 are derived as follows.

– If zVR is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 1) >> 1 \quad (8\text{-}58)$$

– Otherwise, if zVR is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[x - (y >> 1) - 2, -1] + 2 * p[x - (y >> 1) - 1, -1] + p[x - (y >> 1), -1] + 2) >> 2 \quad (8\text{-}59)$$

– Otherwise, if zVR is equal to -1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}60)$$

– Otherwise (zVR is equal to -2 or -3),

$$pred4x4_L[x, y] = (p[-1, y - 1] + 2 * p[-1, y - 2] + p[-1, y - 3] + 2) >> 2 \quad (8\text{-}61)$$

8.3.1.2.7 Specification of Intra_4x4_Horizontal_Down prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, -1] with x = 0..3 and p[-1, y] with y = -1..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHD be set equal to 2 * y – x.

The values of the prediction samples $pred4x4_L$[x, y], with x, y = 0..3 are derived as follows.

– If zHD is equal to 0, 2, 4, or 6,

$$pred4x4_L[x, y] = (p[-1, y -(x >> 1) - 1] + p[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}62)$$

– Otherwise, if zHD is equal to 1, 3, or 5,

$$pred4x4_L[x, y] = (p[-1, y - (x >> 1) - 2] + 2 * p[-1, y - (x >> 1) - 1] + p[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}63)$$

– Otherwise, if zHD is equal to -1,

$$pred4x4_L[x, y] = (p[-1, 0] + 2 * p[-1, -1] + p[0, -1] + 2) >> 2 \quad (8\text{-}64)$$

– Otherwise (zHD is equal to -2 or -3),

$$pred4x4_L[x, y] = (p[x - 1, -1] + 2 * p[x - 2, -1] + p[x - 3, -1] + 2) >> 2 \quad (8\text{-}65)$$

8.3.1.2.8 Specification of Intra_4x4_Vertical_Left prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_4x4 prediction".

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows.

– If y is equal to 0 or 2,

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + p[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}66)$$

– Otherwise (y is equal to 1 or 3),

$$pred4x4_L[x, y] = (p[x + (y >> 1), -1] + 2 * p[x + (y >> 1) + 1, -1] + p[x + (y >> 1) + 2, -1] + 2) >> 2 \quad (8\text{-}67)$$

8.3.1.2.9 Specification of Intra_4x4_Horizontal_Up prediction mode

This Intra_4x4 prediction mode is invoked when Intra4x4PredMode[luma4x4BlkIdx] is equal to 8.

This mode shall be used only when the samples p[-1, y] with y = 0..3 are marked as "available for Intra_4x4 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred4x4_L[x, y]$, with x, y = 0..3 are derived as follows:

– If zHU is equal to 0, 2, or 4

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + p[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}68)$$

– Otherwise, if zHU is equal to 1 or 3

$$pred4x4_L[x, y] = (p[-1, y + (x >> 1)] + 2 * p[-1, y + (x >> 1) + 1] + p[-1, y + (x >> 1) + 2] + 2) >> 2 \quad (8\text{-}69)$$

– Otherwise, if zHU is equal to 5,

$$pred4x4_L[x, y] = (p[-1, 2] + 3 * p[-1, 3] + 2) >> 2 \quad (8\text{-}70)$$

– Otherwise (zHU is greater than 5),

$$pred4x4_L[x, y] = p[-1, 3] \quad (8\text{-}71)$$

8.3.2 Intra_8x8 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_8x8.

Inputs to this process are the values of Intra4x4PredMode (if available) or Intra8x8PredMode (if available) from the neighbouring macroblocks or macroblock pairs.

Outputs of this process are 8x8 luma sample arrays as part of the 16x16 luma array of prediction samples of the macroblock $pred_L$.

The luma component of a macroblock consists of 4 blocks of 8x8 luma samples. These blocks are inverse scanned using the inverse 8x8 luma block scanning process as specified in subclause 6.4.4.

For all 8x8 luma blocks of the luma component of a macroblock with luma8x8BlkIdx = 0..3, the derivation process for Intra8x8PredMode as specified in subclause 8.3.2.1 is invoked with luma8x8BlkIdx as well as Intra4x4PredMode and Intra8x8PredMode that are previously (in decoding order) derived for adjacent macroblocks as the input and the variable Intra8x8PredMode[luma8x8BlkIdx] as the output.

For each luma block of 8x8 samples indexed using luma8x8BlkIdx = 0..3, the following applies.

– The Intra_8x8 sample prediction process in subclause 8.3.2.2 is invoked with luma8x8BlkIdx, Intra8x8PredMode, and the array S'_L of constructed samples prior ~~(in decoding order)~~ to the deblocking filter process ~~from adjacent luma blocks~~ as the input and the output are the Intra_8x8 luma prediction samples $pred8x8_L$[x, y] with x, y = 0..7.

– The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.4 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO) and x, y = 0..7.

$$pred_L[\ xO + x, yO + y\ ] = pred8x8_L[\ x, y\ ] \qquad (8\text{-}72)$$

– The transform coefficient decoding process and picture construction process prior to deblocking filter process in subclause 8.5 is invoked with $pred_L$ and luma8x8BlkIdx as the input and the constructed samples for the current 8x8 luma block S'_L as the output.

8.3.2.1 Derivation process for Intra8x8PredMode

Inputs to this process are the index of the 8x8 luma block luma8x8BlkIdx and variable arrays Intra4x4PredMode (if available) and Intra8x8PredMode (if available) that are previously (in decoding order) derived for adjacent macroblocks.

Output of this process is the variable Intra8x8PredMode[luma8x8BlkIdx].

Table 8-3 specifies the values for Intra8x8PredMode[luma8x8BlkIdx] and the associated mnemonic names.

Table 8-3 – Specification of Intra8x8PredMode[luma8x8BlkIdx] and associated names

Intra8x8PredMode[luma8x8BlkIdx]	Name of Intra8x8PredMode[luma8x8BlkIdx]
0	Intra_8x8_Vertical (prediction mode)
1	Intra_8x8_Horizontal (prediction mode)
2	Intra_8x8_DC (prediction mode)
3	Intra_8x8_Diagonal_Down_Left (prediction mode)
4	Intra_8x8_Diagonal_Down_Right (prediction mode)
5	Intra_8x8_Vertical_Right (prediction mode)
6	Intra_8x8_Horizontal_Down (prediction mode)
7	Intra_8x8_Vertical_Left (prediction mode)
8	Intra_8x8_Horizontal_Up (prediction mode)

Intra8x8PredMode[luma8x8BlkIdx] is derived as follows.

– The process specified in subclause 6.4.8.2 is invoked with luma8x8BlkIdx given as input and the output is assigned to mbAddrA, luma8x8BlkIdxA, mbAddrB, and luma8x8BlkIdxB.

– The variables intra4x4PredModeN and intra8x8PredModeN, with N being replaced by A or B, are derived as follows.

  – If the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode, intra4x4PredModeN is set equal to the variable array Intra4x4PredMode which is assigned to the macroblock mbAddrN, and intra8x8PredModeN is marked as "not available".

  – Otherwise, if the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode, intra8x8PredModeN is set equal to the variable array Intra8x8PredMode which is assigned to the macroblock mbAddrN, and intra4x4PredModeN is marked as "not available".

  – Otherwise (the macroblock with address mbAddrN is not coded in Intra_4x4 or Intra_8x8 macroblock prediction mode), intra4x4PredModeN and intra8x8PredModeN are marked as "not available".

– The derivation process for Intra_8x8 prediction modes as specified in subclause 8.3.2.1.1 is invoked with mbAddrA, mbAddrB, luma8x8BlkIdxA, luma8x8BlkIdxB, intra4x4PredModeA, intra4x4PredModeB, intra8x8PredModeA, and intra8x8PredModeB as inputs and the output is assigned to Intra8x8PredMode[luma8x8BlkIdx].

8.3.2.1.1 Derivation process for Intra_8x8 prediction modes

Inputs to this process are

– two macroblock addresses mbAddrA and mbAddrB

– two 8x8 block indices luma8x8BlkIdxA and luma8x8BlkIdxB

– two lists of intra 4x4 prediction modes intra4x4PredModeA and intra4x4PredModeB

– two lists of intra 8x8 prediction modes intra8x8PredModeA and intra8x8PredModeB

Output of this process is the variable currIntra8x8PredMode.

The variable currIntra8x8PredMode is derived as follows.

– The variable dcPredModePredictedFlag is derived as follows.

  – If any of the following conditions are true, dcPredModePredictedFlag is set equal to 1

    – the macroblock with address mbAddrA is not available

    – the macroblock with address mbAddrB is not available

    – the macroblock with address mbAddrA is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1

    – the macroblock with address mbAddrB is available and coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1

  – Otherwise, dcPredModePredictedFlag is set equal to 0.

– For N being either replaced by A or B, the variables intraMxMPredModeN are derived as follows.

  – If dcPredModePredictedFlag is equal to 1 or (the macroblock with address mbAddrN is not coded in Intra_4x4 macroblock prediction mode and the macroblock with address mbAddrN is not coded in Intra_8x8 macroblock prediction mode), intraMxMPredModeN is set equal to 2 (Intra_8x8_DC prediction mode).

  – Otherwise (dcPredModePredictedFlag is equal to 0 and (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock prediction mode or the macroblock with address mbAddrN is coded in Intra_8x8 macroblock prediction mode)), the following applies.

    – If the macroblock with address mbAddrN is coded in Intra_8x8 macroblock mode, intraMxMPredModeN is set equal to ~~Intra8x8PredMode~~intra8x8PredModeN[luma8x8BlkIdxN]~~, where Intra8x8PredMode is the variable array assigned to the macroblock mbAddrN~~.

    – Otherwise (the macroblock with address mbAddrN is coded in Intra_4x4 macroblock mode), intraMxMPredModeN is derived by the following procedure~~, where Intra4x4PredMode is the variable array assigned to the macroblock mbAddrN~~.

带格式的: 项目符号和编号

带格式的: 项目符号和编号

带格式的: 项目符号和编号

intraMxMPredModeN = ~~Intra4x4PredMode~~intra4x4PredModeN[luma8x8BlkIdxN * 4 + n] (8-73)

where the variable n is derived as follows

– If N is equal to A, depending on the variable MbaffFrameFlag, the variable luma8x8BlkIdx, the current macroblock, and the macroblock mbAddrN, the following applies.

  – If MbaffFrameFlag is equal to 1, the current macroblock is a frame coded macroblock, the macroblock mbAddrN is a field coded macroblock, and luma8x8BlkIdx is equal to 2, n is set equal to 3.

  – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a field coded macroblock or the macroblock mbAddrN is a frame coded macroblock or luma8x8BlkIdx is not equal to 2), n is set equal to 1.

– Otherwise (N is equal to B), n is set equal to 2.

– Finally, given intraMxMPredModeA and intraMxMPredModeB, the variable currIntra8x8PredMode~~[luma8x8BlkIdx]~~ is derived by applying the following procedure.

```
predIntra8x8PredMode = Min( intraMxMPredModeA, intraMxMPredModeB )
if( prev_intra8x8_pred_mode_flag[ luma8x8BlkIdx ] )
    currIntra8x8PredMode[ luma8x8BlkIdx ] = predIntra8x8PredMode
else                                                                    (8-74)
    if( rem_intra8x8_pred_mode[ luma8x8BlkIdx ] < predIntra8x8PredMode )
        currIntra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ]
    else
        currIntra8x8PredMode[ luma8x8BlkIdx ] = rem_intra8x8_pred_mode[ luma8x8BlkIdx ] + 1
```

8.3.2.2 Intra_8x8 sample prediction

This process is invoked for each 8x8 luma block of a macroblock with prediction mode equal to Intra_8x8 followed by the transform decoding process and picture construction process prior to deblocking for each 8x8 luma block.

Inputs to this process ~~is~~ are

– the index of an 8x8 luma block luma8x8BlkIdx

– an array Intra8x8PredMode specifying the Intra_8x8 prediction modes for the current macroblock

– an (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$)~~,~~ array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring blocks

带格式的: 项目符号和编号

Output of this process are the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 for the 8x8 luma block with index luma8x8BlkIdx.

The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the current macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.4 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).

The 25 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..7 and x = 0..15, y = -1, are derived as follows.

– The luma location (xN, yN) is specified by

xN = xO + x (8-75)

yN = yO + y (8-76)

– The derivation process for neighbouring locations in subclause 6.4.9 is invoked for luma locations with (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 is derived as follows.

– If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_8x8 prediction"

  – mbAddrN is not available,

  – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1,

– Otherwise, the sample p[x, y] is marked as "available for Intra_8x8 prediction" and the value of ~~luma~~ the sample ~~at luma location (xW, yW) inside the macroblock mbAddrN is assigned to~~ p[x, y] is derived as follows.

  – The position of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as input and the output being assigned to (xM, yM).

  – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

    – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

      $p[x, y] = cS_L[xM + xW, yM + 2 * yW]$ (8-77)

    – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

      $p[x, y] = cS_L[xM + xW, yM + yW]$ (8-78)

带格式的: 项目符号和编号

带格式的: 项目符号和编号

When samples p[x, -1], with x = 8..15 are marked as "not available for Intra_8x8 prediction," and the sample p[7, -1] is marked as "available for Intra_8x8 prediction," the sample value of p[7, -1] is substituted for sample values p[x, -1], with x = 8..15 and samples p[x, -1], with x = 8..15 are marked as "available for Intra_8x8 prediction".

NOTE – Each block is assumed to be constructed into a picture array prior to decoding of the next block.

The reference sample filtering process for Intra_8x8 sample prediction in subclause 8.3.2.2.1 is invoked with the samples p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 (if available) as input and p'[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 as output.

Depending on Intra8x8PredMode[luma8x8BlkIdx], one of the Intra_8x8 prediction modes specified in subclauses 8.3.2.2.2 to 8.3.2.2.10 is invoked.

8.3.2.2.1 Reference sample filtering process for Intra_8x8 sample prediction

Inputs to this process are the reference samples p[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 (if available) for Intra_8x8 sample prediction.

Outputs of this process are the filtered reference samples p'[x, y] with x = -1, y = -1..7 and x = 0..15, y = -1 for Intra_8x8 sample prediction.

When all samples p[x, -1] with x = 0..7 are marked as "available for Intra_8x8 prediction", the following applies.

– The value of p'[0, -1] is derived as follows.

  – If p[-1, -1] is marked as "available for Intra_8x8 prediction", p'[0, -1] is derived by

    $p'[0, -1] = (p[-1, -1] + 2 * p[0, -1] + p[1, -1] + 2) >> 2$ (8-79)

  – Otherwise (p[-1, -1] is marked as "not available for Intra_8x8 prediction"), p'[0, -1] is derived by

    $p'[0, -1] = (3 * p[0, -1] + p[1, -1] + 2) >> 2$ (8-80)

– The values of p'[x, -1], with x = 1..7 are derived by

  $p'[x, -1] = (p[x-1, -1] + 2 * p[x, -1] + p[x+1, -1] + 2) >> 2$ (8-81)

When all samples p[x, -1] with x = 7..15 are marked as "available for Intra_8x8 prediction", the following applies.

– The values of p'[x, -1], with x = 8..14 are derived by

p'[x, -1] = (p[x-1, -1] + 2 * p[x, -1] + p[x+1, -1] + 2) >> 2 (8-82)

– The value of p'[15, -1] is derived by

p'[15, -1] = (p[14, -1] + 3 * p[15, -1] + 2) >> 2 (8-83)

When the sample p[-1, -1] is marked as "available for Intra_8x8 prediction", the value of p'[-1, -1] is derived as follows.

– If the sample p[0, -1] is marked as "not available for Intra_8x8 prediction" or the sample p[-1, 0] is marked as "not available for Intra_8x8 prediction", the following applies.

  – If the sample p[0, -1] is marked as "available for Intra_8x8 prediction", p'[-1, -1] is derived by

  p'[-1, -1] = (3 * p[-1, -1] + p[0, -1] + 2) >> 2 (8-84)

  – Otherwise (the sample p[0, -1] is marked as "not available for Intra_8x8 prediction" and the sample p[-1, 0] is marked as "available for Intra_8x8 prediction"), p'[-1, -1] is derived by

  p'[-1, -1] = (3 * p[-1, -1] + p[-1, 0] + 2) >> 2 (8-85)

– Otherwise (the sample p[0, -1] is marked as "available for Intra_8x8 prediction" and the sample p[-1, 0] is marked as "available for Intra_8x8 prediction"), p'[-1, -1] is derived by

p'[-1, -1] = (p[0, -1] + 2 * p[-1, -1] + p[-1, 0] + 2) >> 2 (8-86)

When all samples p[-1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction", the following applies.

– The value of p'[-1, 0] is derived as follows.

  – If p[-1, -1] is marked as "available for Intra_8x8 prediction", p'[-1, 0] is derived by

  p'[-1, 0] = (p[-1, -1] + 2 * p[-1, 0] + p[-1, 1] + 2) >> 2 (8-87)

  – Otherwise (if p[-1, -1] is marked as "not available for Intra_8x8 prediction"), p'[-1, 0] is derived by

  p'[-1, 0] = (3 * p[-1, 0] + p[-1, 1] + 2) >> 2 (8-88)

– The values of p'[-1, y], with y = 1..6 are derived by

p'[-1, y] = (p[-1, y-1] + 2 * p[-1, y] + p[-1, y+1] + 2) >> 2 (8-89)

– The value of p'[-1, 7] is derived by

p'[-1, 7] = (p[-1, 6] + 3 * p[-1, 7] + 2) >> 2 (8-90)

8.3.2.2.2 Specification of Intra_8x8_Vertical prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 0.

This mode shall be used only when the samples p[x, -1] with x = 0..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = p'[\ x, -1\ ], \text{ with } x, y = 0..7 \tag{8-91}$$

8.3.2.2.3 Specification of Intra_8x8_Horizontal prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 1.

This mode shall be used only when the samples p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = p'[\ -1, y\ ], \text{ with } x, y = 0..7 \tag{8-92}$$

8.3.2.2.4 Specification of Intra_8x8_DC prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 2.

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived as follows.

– If all samples p[x, -1], with x = 0..7 and p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction," the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = (\sum_{x'=0}^{7} p'[x', -1] + \sum_{y'=0}^{7} p'[-1, y'] + 8) >> 4 \tag{8-93}$$

– Otherwise, if any samples p[x, -1], with x = 0..7 are marked as "not available for Intra_8x8 prediction" and all samples p[-1, y], with y = 0..7 are marked as "available for Intra_8x8 prediction", the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = (\sum_{y'=0}^{7} p'[-1, y'] + 4) >> 3 \tag{8-94}$$

– Otherwise, if any samples p[-1, y], with y = 0..7 are marked as "not available for Intra_8x8 prediction" and all samples p[x, -1], with x = 0..7 are marked as "available for Intra_8x8 prediction", the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = (\sum_{x'=0}^{7} p'[x', -1] + 4) >> 3 \tag{8-95}$$

– Otherwise (some samples p[x, -1], with x = 0..7 and some samples p[-1, y], with y = 0..7 are marked as "not available for Intra_8x8 prediction"), the values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived by

$$pred8x8_L[\ x, y\ ] = (\ 1 << (\ BitDepth_Y - 1\ )\ ) \tag{8-96}$$

NOTE – An 8x8 luma block can always be predicted using this mode.

8.3.2.2.5 Specification of Intra_8x8_Diagonal_Down_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 3.

This mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L$[x, y], with x, y = 0..7 are derived as follows.

– If x is equal to 7 and y is equal to 7,

$$pred8x8_L[\ x, y\ ] = (\ p'[\ 14, -1\ ] + 3 * p'[15, -1\ ] + 2\ ) >> 2 \tag{8-97}$$

– Otherwise (x is not equal to 7 or y is not equal to 7),

$$pred8x8_L[\ x, y\ ] = (\ p'[\ x + y, -1\ ] + 2 * p'[\ x + y + 1, -1\ ] + p'[\ x + y + 2, -1\ ] + 2\ ) >> 2 \quad (8\text{-}98)$$

8.3.2.2.6 Specification of Intra_8x8_Diagonal_Down_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 4.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[\ x, y\ ]$, with x, y = 0..7 are derived as follows.

– If x is greater than y,

$$pred8x8_L[\ x, y\ ] = (\ p'[\ x - y - 2, -1] + 2 * p'[\ x - y - 1, -1\ ] + p'[\ x - y, -1\ ] + 2\ ) >> 2 \quad (8\text{-}99)$$

– Otherwise if x is less than y,

$$pred8x8_L[\ x, y\ ] = (\ p'[\ -1, y - x - 2\ ] + 2 * p'[\ -1, y - x - 1\ ] + p'[\ -1, y - x\ ] + 2\ ) >> 2 \quad (8\text{-}100)$$

– Otherwise (x is equal to y),

$$pred8x8_L[\ x, y\ ] = (\ p'[\ 0, -1\ ] + 2 * p'[\ -1, -1\ ] + p'[\ -1, 0\ ] + 2\ ) >> 2 \quad (8\text{-}101)$$

8.3.2.2.7 Specification of Intra_8x8_Vertical_Right prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 5.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zVR be set equal to 2 * x – y.

The values of the prediction samples $pred8x8_L[\ x, y\ ]$, with x, y = 0..7 are derived as follows.

– If zVR is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$pred8x8_L[\ x, y\ ] = (\ p'[\ x - (\ y >> 1\ ) - 1, -1\ ] + p'[\ x - (\ y >> 1\ ), -1\ ] + 1\ ) >> 1 \quad (8\text{-}102)$$

– Otherwise, if zVR is equal to 1, 3, 5, 7, 9, 11, or 13

$$pred8x8_L[\ x, y\ ] = (\ p'[\ x - (\ y >> 1\ ) - 2, -1] + 2 * p'[\ x - (\ y >> 1\ ) - 1, -1\ ] + p'[\ x - (\ y >> 1\ ), -1\ ] + 2\ ) >> 2 \quad (8\text{-}103)$$

– Otherwise, if zVR is equal to -1,

$$pred8x8_L[\ x, y\ ] = (\ p'[\ -1, 0\ ] + 2 * p'[\ -1, -1\ ] + p'[\ 0, -1\ ] + 2\ ) >> 2 \quad (8\text{-}104)$$

– Otherwise (zVR is equal to -2, -3, -4, -5, -6, or -7),

$$pred8x8_L[\ x, y\ ] = (\ p'[\ -1, y - 2*x - 1\ ] + 2 * p'[\ -1, y - 2*x - 2\ ] + p'[\ -1, y - 2*x - 3\ ] + 2\ ) >> 2 \quad (8\text{-}105)$$

8.3.2.2.8 Specification of Intra_8x8_Horizontal_Down prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 6.

This mode shall be used only when the samples p[x, -1] with x = 0..7 and p[-1, y] with y = -1..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHD be set equal to 2 * y – x.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If zHD is equal to 0, 2, 4, 6, 8, 10, 12, or 14

$$pred8x8_L[x, y] = (p'[-1, y -(x >> 1) - 1] + p'[-1, y - (x >> 1)] + 1) >> 1 \quad (8\text{-}106)$$

– Otherwise, if zHD is equal to 1, 3, 5, 7, 9, 11, or 13

$$pred8x8_L[x, y] = (p'[-1, y - (x >> 1) - 2] + 2 * p'[-1, y - (x >> 1) - 1] + p'[-1, y - (x >> 1)] + 2) >> 2 \quad (8\text{-}107)$$

– Otherwise, if zHD is equal to -1,

$$pred8x8_L[x, y] = (p'[-1, 0] + 2 * p'[-1, -1] + p'[0, -1] + 2) >> 2 \quad (8\text{-}108)$$

– Otherwise (zHD is equal to –2, -3, -4, -5, -6, -7),

$$pred8x8_L[x, y] = (p'[x - 2*y - 1, -1] + 2 * p'[x - 2*y - 2, -1] + p'[x - 2*y - 3, -1] + 2) >> 2 \quad (8\text{-}109)$$

8.3.2.2.9 Specification of Intra_8x8_Vertical_Left prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 7.

This mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_8x8 prediction".

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows.

– If y is equal to 0, 2, 4 or 6

$$pred8x8_L[x, y] = (p'[x + (y >> 1), -1] + p'[x + (y >> 1) + 1, -1] + 1) >> 1 \quad (8\text{-}110)$$

– Otherwise (y is equal to 1, 3, 5, 7),

$$pred8x8_L[x, y] = (p'[x + (y >> 1), -1] + 2 * p'[x + (y >> 1) + 1, -1] + p'[x + (y >> 1) + 2, -1] + 2) >>2 \quad (8\text{-}111)$$

8.3.2.2.10 Specification of Intra_8x8_Horizontal_Up prediction mode

This Intra_8x8 prediction mode is invoked when Intra8x8PredMode[luma8x8BlkIdx] is equal to 8.

This mode shall be used only when the samples p[-1, y] with y = 0..7 are marked as "available for Intra_8x8 prediction".

Let the variable zHU be set equal to x + 2 * y.

The values of the prediction samples $pred8x8_L[x, y]$, with x, y = 0..7 are derived as follows:

– If zHU is equal to 0, 2, 4, 6, 8, 10, or 12

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + p'[-1, y + (x >> 1) + 1] + 1) >> 1 \quad (8\text{-}112)$$

– Otherwise, if zHU is equal to 1, 3, 5, 7, 9, or 11

$$pred8x8_L[x, y] = (p'[-1, y + (x >> 1)] + 2 * p'[-1, y + (x >> 1) + 1] + p'[-1, y + (x >> 1) + 2] + 2) >>2 \quad (8\text{-}113)$$

– Otherwise, if zHU is equal to 13,

$$pred8x8_L[x, y] = (p'[-1, 6] + 3 * p'[-1, 7] + 2) >> 2 \quad (8\text{-}114)$$

– Otherwise (zHU is greater than 13),

$pred8x8_L[\ x, y\ ] = p'[\ -1, 7\ ]$ (8-115)

8.3.3 Intra_16x16 prediction process for luma samples

This process is invoked when the macroblock prediction mode is equal to Intra_16x16. It specifies how the Intra prediction luma samples for the current macroblock are derived.

Input to this process is an $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ array cS_L containing constructed luma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction luma samples for the current macroblock $pred_L[\ x, y\ ]$.

The 33 neighbouring samples p[x, y] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..15 and with x = 0..15, y = -1, are derived as follows.

– The derivation process for neighbouring locations in subclause 6.4.9 is invoked for luma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.

– Each sample p[x, y] with x = -1, y = -1..15 and with x = 0..15, y = -1 is derived as follows.

  – If any of the following conditions is true, the sample p[x, y] is marked as "not available for Intra_16x16 prediction"

    – mbAddrN is not available,

    – the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1.

    – the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1.

  – Otherwise, the sample p[x, y] is marked as "available for Intra_16x16 prediction" and the value of the ~~luma~~ sample ~~at luma location (xW, yW) inside the macroblock mbAddrN is assigned to~~ p[x, y] is derived as follows.

    – The position of the upper-left sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process ain subclause 6.4.1 with mbAddrN as input and the output being assigned to (xM, yM).

    – Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

      – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

        $p[\ x, y\ ] = cS_L[\ xM + xW, yM + 2 * yW\ ]$ (8-116)

      – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

        $p[\ x, y\ ] = cS_L[\ xM + xW, yM + yW\ ]$ (8-117)

带格式的: 项目符号和编号

带格式的: 项目符号和编号

Let $pred_L[\ x, y\ ]$ with x, y = 0..15 denote the prediction samples for the 16x16 luma block samples.

Intra_16x16 prediction modes are specified in Table 8-4.

Table 8-4 – Specification of Intra16x16PredMode and associated names

Intra16x16PredMode	Name of Intra16x16PredMode
0	Intra_16x16_Vertical (prediction mode)
1	Intra_16x16_Horizontal (prediction mode)
2	Intra_16x16_DC (prediction mode)
3	Intra_16x16_Plane (prediction mode)

Depending on Intra16x16PredMode, one of the Intra_16x16 prediction modes specified in subclauses 8.3.3.1 to 8.3.3.4 is invoked.

8.3.3.1 Specification of Intra_16x16_Vertical prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, -1] with x = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[\ x, y\ ] = p[\ x, -1\ ], \text{ with } x, y = 0..15 \tag{8-118}$$

8.3.3.2 Specification of Intra_16x16_Horizontal prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[-1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[\ x, y\ ] = p[\ -1, y\ ], \text{ with } x, y = 0..15 \tag{8-119}$$

8.3.3.3 Specification of Intra_16x16_DC prediction mode

This Intra_16x16 prediction mode operates, depending on whether the neighbouring samples are marked as "available for Intra_16x16 prediction", as follows.

– If all neighbouring samples p[x, -1], with x = 0..15 and p[-1, y], with y = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\ x, y\ ] = (\sum_{x'=0}^{15} p[x', -1] + \sum_{y'=0}^{15} p[-1, y'] + 16) >> 5, \text{ with } x, y = 0..15 \tag{8-120}$$

– Otherwise, if any of the neighbouring samples p[x, -1], with x = 0..15 are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[-1, y], with y = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\ x, y\ ] = (\sum_{y'=0}^{15} p[-1, y'] + 8) >> 4, \text{ with } x, y = 0..15 \tag{8-121}$$

– Otherwise, if any of the neighbouring samples p[-1, y], with y = 0..15 are marked as "not available for Intra_16x16 prediction" and all of the neighbouring samples p[x, -1], with x = 0..15 are marked as "available for Intra_16x16 prediction", the prediction for all luma samples in the macroblock is given by:

$$pred_L[\ x, y\ ] = (\sum_{x'=0}^{15} p[x', -1] + 8) >> 4, \text{ with } x, y = 0..15 \tag{8-122}$$

– Otherwise (some of the neighbouring samples p[x, -1], with x = 0..15 and some of the neighbouring samples p[-1, y], with y = 0..15 are marked as "not available for Intra_16x16 prediction"), the prediction for all luma samples in the macroblock is given by:

$$pred_L[\ x, y\ ] = (\ 1 << (\ BitDepth_Y - 1\ )\ ), \text{ with } x, y = 0..15 \tag{8-123}$$

8.3.3.4 Specification of Intra_16x16_Plane prediction mode

This Intra_16x16 prediction mode shall be used only when the samples p[x, -1] with x = -1..15 and p[-1, y] with y = 0..15 are marked as "available for Intra_16x16 prediction".

$$pred_L[\ x, y\ ] = Clip1_Y(\ (\ a + b * (\ x - 7\ ) + c * (\ y - 7\ ) + 16\ ) >> 5\ ), \text{ with } x, y = 0..15, \tag{8-124}$$

where:

$$a = 16 * (p[-1, 15] + p[15, -1]) \quad (8\text{-}125)$$

$$b = (5 * H + 32) >> 6 \quad (8\text{-}126)$$

$$c = (5 * V + 32) >> 6 \quad (8\text{-}127)$$

and H and V are specified in Equations 8-128 and 8-129.

$$H = \sum_{x'=0}^{7} (x'+1) * (p[8 + x', -1] - p[6 - x', -1]) \quad (8\text{-}128)$$

$$V = \sum_{y'=0}^{7} (y'+1) * (p[-1, 8 + y'] - p[-1, 6 - y']) \quad (8\text{-}129)$$

8.3.4 Intra prediction process for chroma samples

This process is invoked for I and SI macroblock types. It specifies how the Intra prediction chroma samples for the current macroblock are derived.

Inputs to this process are two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays cS_{Cb} and cS_{Cr} containing constructed chroma samples prior to the deblocking filter process of neighbouring macroblocks.

Outputs of this process are Intra prediction chroma samples for the current macroblock $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$.

Both chroma blocks (Cb and Cr) of the macroblock use the same prediction mode. The prediction mode is applied to each of the chroma blocks separately. The process specified in this subclause is invoked for each chroma block. In the remainder of this subclause, chroma block refers to one of the two chroma blocks and the subscript C is used as a replacement of the subscript Cb or Cr.

The neighbouring samples p[x, y] that are constructed chroma samples prior to the deblocking filter process, with x = -1, y = -1..MbHeightC - 1 and with x = 0..MbWidthC - 1, y = -1, are derived as follows.

- The derivation process for neighbouring locations in subclause 6.4.9 is invoked for chroma locations with (x, y) assigned to (xN, yN) as input and mbAddrN and (xW, yW) as output.
- Each sample p[x, y] is derived as follows.
  - If any of the following conditions is true, the sample p[x, y] is marked as “not available for Intra chroma prediction”
    - mbAddrN is not available,
    - the macroblock mbAddrN is coded in Inter prediction mode and constrained_intra_pred_flag is equal to 1.
    - the macroblock mbAddrN has mb_type equal to SI and constrained_intra_pred_flag is equal to 1 and the current macroblock does not have mb_type equal to SI.
  - Otherwise, the sample p[x, y] is marked as “available for Intra chroma prediction” and the value of the chroma sample p[x, y] of component C (with C being either Cb or Cr) ~~at chroma location (xW, yW) inside the macroblock mbAddrN is assigned to p[x, y]~~is derived as follows.
    - The position of the upper-left luma sample of the macroblock mbAddrN is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddrN as input and the output being assigned to (xL, yL).
    - The position (xM, yM) of the upper-left chroma sample of the macroblock mbAddr is derived by

$$xM = (xL * MbWidthC) >> 4 \quad (8\text{-}130)$$
$$yM = ((yL * MbHeightC) >> 4) + (yL \% 2) \quad (8\text{-}131)$$

带格式的: 项目符号和编号

– Depending on the variable MbaffFrameFlag and the macroblock mbAddrN, the sample value p[x, y] is derived as follows.

  – If MbaffFrameFlag is equal to 1 and the macroblock mbAddrN is a field macroblock,

$$p[\ x, y\ ] = cS_C[\ xM + xW, yM + 2 * yW\ ] \quad (8\text{-}132)$$

  – Otherwise (MbaffFrameFlag is equal to 0 or the macroblock mbAddrN is a frame macroblock),

$$p[\ x, y\ ] = cS_C[\ xM + xW, yM + yW\ ] \quad (8\text{-}133)$$

带格式的: 项目符号和编号

带格式的: 项目符号和编号

Let $pred_C$[x, y] with x = 0..MbWidthC - 1, y = 0..MbHeightC - 1 denote the prediction samples for the chroma block samples.

Intra chroma prediction modes are specified in Table 8-5.

Table 8-5 – Specification of Intra chroma prediction modes and associated names

intra_chroma_pred_mode	Name of intra_chroma_pred_mode
0	Intra_Chroma_DC (prediction mode)
1	Intra_Chroma_Horizontal (prediction mode)
2	Intra_Chroma_Vertical (prediction mode)
3	Intra_Chroma_Plane (prediction mode)

Depending on intra_chroma_pred_mode, one of the Intra chroma prediction modes specified in subclauses 8.3.4.1 to 8.3.4.4 is invoked.

8.3.4.1 Specification of Intra_Chroma_DC prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 0.

For each chroma block of 4x4 samples indexed by chroma4x4BlkIdx = 0..(1 << (chroma_format_idc + 1)) – 1, the following applies.

– Depending on chroma_format_idc, the position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx is derived as follows

  – If chroma_format_idc is equal to 1 or 2, the following applies

$$xO = InverseRasterScan(\ chroma4x4BlkIdx, 4, 4, 8, 0\ ) \quad (8\text{-}134)$$

$$yO = InverseRasterScan(\ chroma4x4BlkIdx, 4, 4, 8, 1\ ) \quad (8\text{-}135)$$

  – Otherwise (chroma_format_idc is equal to 3), the following applies

$$xO = InverseRasterScan(\ chroma4x4BlkIdx / 4, 8, 8, 16, 0\ ) + InverseRasterScan(\ chroma4x4BlkIdx \% 4, 4, 4, 8, 0\ ) \quad (8\text{-}136)$$

$$yO = InverseRasterScan(\ chroma4x4BlkIdx / 4, 8, 8, 16, 1\ ) + InverseRasterScan(\ chroma4x4BlkIdx \% 4, 4, 4, 8, 1\ ) \quad (8\text{-}137)$$

– If (xO, yO) is equal to (0, 0) or xO and yO are greater than 0, the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows.

– If all samples p[x + xO, -1], with x = 0..3 and p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + \sum_{y'=0}^{3} p[-1,y'+yO] + 4 \right) >> 3 \text{, with x, y = 0..3.} \quad (8\text{-}138)$$

– Otherwise, if any samples p[x + xO, -1], with x = 0..3 are marked as "not available for Intra chroma prediction" and all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = \left(\sum_{y'=0}^{3} p[-1,y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}139)$$

– Otherwise, if any samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction" and all samples p[x + xO, -1], with x = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}140)$$

– Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = (\,1 << (\,BitDepth_C - 1\,)\,)\text{, with x, y = 0..3.} \quad (8\text{-}141)$$

– Otherwise, if xO is greater than 0 and yO is equal to 0, the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows.

– If all samples p[x + xO, -1], with x = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = \left(\sum_{x'=0}^{3} p[x'+xO,-1] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}142)$$

– Otherwise, if all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = \left(\sum_{y'=0}^{3} p[-1,y'+yO] + 2 \right) >> 2 \text{, with x, y = 0..3.} \quad (8\text{-}143)$$

– Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as "not available for Intra chroma prediction"), the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$pred_C[\,x + xO,\ y + yO\,] = (\,1 << (\,BitDepth_C - 1\,)\,)\text{, with x, y = 0..3.} \quad (8\text{-}144)$$

– Otherwise (xO is equal to 0 and yO is greater than 0), the values of the prediction samples $pred_C$[x + xO, y + yO] with x, y = 0..3 are derived as follows.

– If all samples p[-1, y +yO], with y = 0..3 are marked as "available for Intra chroma prediction", the values of the prediction samples $pred_C$[x + xO, y + yO], with x, y = 0..3 are derived as

$$\text{pred}_C[\,x + xO,\ y + yO\,] = \left(\sum_{y'=0}^{3} p[-1, y' + yO] + 2 \right) >> 2 \text{ , with x, y = 0..3.} \quad (8\text{-}145)$$

– Otherwise, if all samples p[x + xO, -1], with x = 0..3 are marked as “available for Intra chroma prediction”, the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3 are derived as

$$\text{pred}_C[\,x + xO,\ y + yO\,] = \left(\sum_{x'=0}^{3} p[x' + xO, -1] + 2 \right) >> 2 \text{ , with x, y = 0..3.} \quad (8\text{-}146)$$

– Otherwise (some samples p[x + xO, -1], with x = 0..3 and some samples p[-1, y +yO], with y = 0..3 are marked as “not available for Intra chroma prediction”), the values of the prediction samples pred_C[x + xO, y + yO], with x, y = 0..3 are derived as

$$\text{pred}_C[\,x + xO,\ y + yO\,] = (\,1 << (\,\text{BitDepth}_C - 1\,)\,)\text{, with x, y = 0..3.} \quad (8\text{-}147)$$

8.3.4.2 Specification of Intra_Chroma_Horizontal prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 1.

This mode shall be used only when the samples p[-1, y] with y = 0..MbHeightC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

$$\text{pred}_C[\,x, y\,] = p[\,-1, y\,]\text{, with x = 0..MbWidthC - 1 and y = 0..MbHeightC - 1} \quad (8\text{-}148)$$

8.3.4.3 Specification of Intra_Chroma_Vertical prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 2.

This mode shall be used only when the samples p[x, -1] with x = 0..MbWidthC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

$$\text{pred}_C[\,x, y\,] = p[\,x, -1\,]\text{, with x = 0..MbWidthC - 1 and y = 0..MbHeightC - 1} \quad (8\text{-}149)$$

8.3.4.4 Specification of Intra_Chroma_Plane prediction mode

This Intra chroma prediction mode is invoked when intra_chroma_pred_mode is equal to 3.

This mode shall be used only when the samples p[x, -1], with x = 0..MbWidthC - 1 and p[-1, y], with y = -1..MbHeightC - 1 are marked as "available for Intra chroma prediction".

The values of the prediction samples pred_C[x, y] are derived as follows.

Let the variable xCF be set equal to 4 * (chroma_format_idc = = 3) and let the variable yCF be set equal to 4 * (chroma_format_idc != 1).

$$\text{pred}_C[\,x, y\,] = \text{Clip1}_C(\,(\,a + b * (\,x - 3 - xCF\,) + c * (\,y - 3 - yCF\,) + 16\,) >> 5\,),$$
$$\text{with x = 0..MbWidthC - 1 and y = 0..MbHeightC - 1} \quad (8\text{-}150)$$

where:

$$a = 16 * (\,p[\,-1, \text{MbHeightC} - 1\,] + p[\,\text{MbWidthC} - 1, -1\,]\,) \quad (8\text{-}151)$$

$$b = (\,(\,34 - 29 * (\,\text{chroma_format_idc} = = 3\,)\,) * H + 32\,) >> 6 \quad (8\text{-}152)$$

$$c = ((34 - 29 * (chroma_format_idc \ != \ 1)) * V + 32) >> 6 \qquad (8\text{-}153)$$

and H and V are specified as

$$H = \sum_{x'=0}^{3+xCF} (x'+1) * (p[4 + xCF + x', -1] - p[2 + xCF - x', -1]) \qquad (8\text{-}154)$$

$$V = \sum_{y'=0}^{3+yCF} (y'+1) * (p[-1, 4 + yCF + y'] - p[-1, 2 + yCF - y']) \qquad (8\text{-}155)$$

8.3.5 Sample construction process for I_PCM macroblocks

This process is invoked when mb_type is equal to I_PCM.

Inputs of this process are

– an (PicWidthInSamples$_L$)x(PicHeightInSamples$_L$) array cS_L relating to constructed luma samples prior to the deblocking filter process

– when chroma_format_idc is not equal to 0, two (PicWidthInSamples$_C$)x(PicHeightInSamples$_C$) array s cS_{Cb} and cS_{Cr} relating to constructed chroma samples prior to the deblocking filter process

带格式的：项目符号和编号

The variable dy is derived as follows.

– If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock, dy is set equal to 2.

– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock), dy is set equal to 1.

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

The constructed luma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < 256; i++ )
  S'L cSL[ xP + ( i % 16 ), yP + dy * ( i / 16 ) ) ] = pcm_sample_luma[ i ]      (8-156)
```

带格式的：英语(英国)

When chroma_format_idc is not equal to 0 (monochrome), the constructed chroma samples prior to the deblocking process are generated as specified by:

```
for( i = 0; i < MbWidthC * MbHeightC; i++ ) {
  S'Cb cSCb[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC – 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i ]                                            (8-157)
  S'Cr cSCr[ ( xP / SubWidthC ) + ( i % MbWidthC ),
            ( ( yP + SubHeightC – 1 ) / SubHeightC ) + dy * ( i / MbWidthC ) ] =
              pcm_sample_chroma[ i + MbWidthC * MbHeightC ]
}
```

8.4 Inter prediction process

This process is invoked when decoding P and B macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array $pred_L$ of luma samples and when chroma_format_idc is not equal to 0 (monochrome) two 8x8 arrays $pred_{Cr}$ and $pred_{Cb}$ of chroma samples, one for each of the chroma components Cb and Cr.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by mbPartIdx. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can be further partitioned into sub-macroblock partitions as specified by sub_mb_type. Each sub-macroblock partition is

referred to by subMbPartIdx. When the macroblock partitioning does not consist of sub-macroblocks, subMbPartIdx is set equal to 0.

The following steps are specified for each macroblock partition or for each sub-macroblock partition.

The functions MbPartWidth( ), MbPartHeight( ), SubMbPartWidth( ), and SubMbPartHeight( ) describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Table 7-13, 7-14, 7-17, and 7-18.

The range of the macroblock partition index mbPartIdx is derived as follows.

– If mb_type is equal to B_Skip or B_Direct_16x16, mbPartIdx proceeds over values 0..3.

– Otherwise (mb_type is not equal to B_Skip or B_Direct_16x16), mbPartIdx proceeds over values 0..NumMbPart(mb_type) – 1.

For each value of mbPartIdx, the variables partWidth and partHeight for each macroblock partition or sub-macroblock partition in the macroblock are derived as follows.

– If mb_type is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, subMbPartIdx is set equal to 0, and partWidth and partHeight are derived as

$$partWidth = MbPartWidth(mb_type) \quad (8\text{-}158)$$

$$partHeight = MbPartHeight(mb_type) \quad (8\text{-}159)$$

– Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is not equal to B_Direct_8x8, subMbPartIdx proceeds over values 0..NumSubMbPart(sub_mb_type) – 1, and partWidth and partHeight are derived as

$$partWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]) \quad (8\text{-}160)$$

$$partHeight = SubMbPartHeight(sub_mb_type[mbPartIdx]). \quad (8\text{-}161)$$

– Otherwise (mb_type is equal to B_Skip or B_Direct_16x16, or mb_type is equal to B_8x8 and sub_mb_type[mbPartIdx] is equal to B_Direct_8x8), subMbPartIdx proceeds over values 0..3, and partWidth and partHeight are derived as

$$partWidth = 4 \quad (8\text{-}162)$$

$$partHeight = 4 \quad (8\text{-}163)$$

When chroma_format_idc is not equal to 0 (monochrome) the variables partWidthC and partHeightC are derived as

$$partWidthC = partWidth / SubWidthC \quad (8\text{-}164)$$
$$partHeightC = partHeight / SubHeightC \quad (8\text{-}165)$$

Let the variable MvCnt be initially set equal to 0 before any invocation of subclause 8.4.1 for the macroblock.

The Inter prediction process for a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx consists of the following ordered steps

1. Derivation process for motion vector components and reference indices as specified in subclause 8.4.1.

   Inputs to this process are

   – a macroblock partition mbPartIdx,

   – a sub-macroblock partition subMbPartIdx.

   – the reference picture lists RefPicList0 and RefPicList1 (if available), which have been derived as specified in subclause 8.2.4

带格式的: 项目符号和编号

Outputs of this process are

- luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) the chroma motion vectors mvCL0 and mvCL1
- reference indices refIdxL0 and refIdxL1
- prediction list utilization flags predFlagL0 and predFlagL1
- the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.
3. Decoding process for Inter prediction samples as specified in subclause 8.4.2.

   Inputs to this process are

   - a macroblock partition mbPartIdx,
   - a sub-macroblock partition subMbPartIdx.
   - variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available), and partHeightC (if available)
   - luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) the chroma motion vectors mvCL0 and mvCL1
   - reference indices refIdxL0 and refIdxL1
   - prediction list utilization flags predFlagL0 and predFlagL1
   - the reference picture lists RefPicList0 and RefPicList1 (if available), which have been derived as specified in subclause 8.2.4
   - the variables $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L}, which have been derived as specified in subclause 8.4.3
   - when chroma_format_idc is not equal to 0, the variables $logWD_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C}, which have been derived as specified in subclause 8.4.3.

   Outputs of this process are

   - inter prediction samples (pred); which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples and when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays $predPart_{Cr}$, and $predPart_{Cb}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made:

MvL0[mbPartIdx][subMbPartIdx] = mvL0 (8-166)

MvL1[mbPartIdx][subMbPartIdx] = mvL1 (8-167)

RefIdxL0[mbPartIdx] = refIdxL0 (8-168)

RefIdxL1[mbPartIdx] = refIdxL1 (8-169)

PredFlagL0[mbPartIdx] = predFlagL0 (8-170)

PredFlagL1[mbPartIdx] = predFlagL1 (8-171)

带格式的：项目符号和编号

The location of the upper-left sample of the partition relative to the upper-left sample of the macroblock is derived by invoking the inverse macroblock partition scanning process as described in subclause 6.4.2.1 with mbPartIdx as the input and (xP, yP) as the output.

The location of the upper-left sample of the macroblock sub-partition relative to the upper-left sample of the macroblock partition is derived by invoking the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 with subMbPartIdx as the input and (xS, yS) as the output.

The macroblock prediction is formed by placing the partition or sub-macroblock partition prediction samples in their correct relative positions in the macroblock, as follows.

The variable $pred_L$[xP + xS + x, yP + yS + y] with x = 0 .. partWidth – 1, y = 0 .. partHeight – 1 is derived by

$$pred_L[\ xP + xS + x,\ yP + yS + y\ ] = predPart_L[\ x,\ y\ ] \qquad (8\text{-}172)$$

When chroma_format_idc is not equal to 0 (monochrome) the variable $pred_C$ with x = 0..partWidthC – 1, y = 0..partHeightC – 1, and C in $pred_C$ and $predPart_C$ being replaced by Cb or Cr is derived by

$$pred_C[\ xP / SubWidthC + xS / SubWidthC + x,\ yP / SubHeightC + yS / SubHeightC + y\ ] = predPart_C[\ x,\ y\ ] \qquad (8\text{-}173)$$

8.4.1 Derivation process for motion vector components and reference indices

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– reference picture lists refPicList0 and refPicList1 (if available).

带格式的：项目符号和编号

Outputs of this process are

– luma motion vectors mvL0 and mvL1 as well as the chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags predFlagL0 and predFlagL1

– a sub-partition macroblock motion vector count variable subMvCnt

For the derivation of the variables mvL0 and mvL1 as well as refIdxL0 and refIdxL1, the following applies.

– If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The sub-partition motion vector count variable subMvCnt is set equal to 1.

– Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices in subclause 8.4.1.2 is invoked with mbPartIdx~~ and~~, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as the input and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, the sub-partition motion vector count subMvCnt, and the prediction utilization flags predFlagL0 and predFlagL1.

– Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX, the following applies.

1. The variables refIdxLX and predFlagLX are derived as follows.

   – If MbPartPredMode(mb_type, mbPartIdx) or SubMbPredMode(sub_mb_type[mbPartIdx]) is equal to Pred_LX or to BiPred,

$$refIdxLX = ref_idx_lX[\ mbPartIdx\ ] \qquad (8\text{-}174)$$

predFlagLX = 1 (8-175)

– Otherwise, the variables refIdxLX and predFlagLX are specified by

refIdxLX = -1 (8-176)

predFlagLX = 0 (8-177)

2. The variable subMvCnt for sub-partition motion vector count is set equal to predFlagL0 + predFlagL1.
3. The variable currSubMbType is derived as follows.
   – If the macroblock type is equal to B_8x8, currSubMbType is set equal to sub_mb_type[mbPartIdx].
   – Otherwise (the macroblock type is not equal to B_8x8), currSubMbType is set equal to "na".
4. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxLX, and currSubMbType as the inputs and the output being mvpLX. The luma motion vectors are derived by

mvLX[0] = mvpLX[0] + mvd_lX[mbPartIdx][subMbPartIdx][0] (8-178)

mvLX[1] = mvpLX[1] + mvd_lX[mbPartIdx][subMbPartIdx][1] (8-179)

For the derivation of the variables for the chroma motion vectors, the following applies. When predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.1.4 is invoked with mvLX and refIdxLX as input and the output being mvCLX.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0. (8-180)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

– The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.
– The variable mvL0 is specified as follows.
  – If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.
    – mbAddrA is not available
    – mbAddrB is not available
    – refIdxL0A is equal to 0 and both components of mvL0A are equal to 0
    – refIdxL0B is equal to 0 and both components of mvL0B are equal to 0
  – Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

8.4.1.2 Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8

This process is invoked when mb_type is equal to B_Skip or B_Direct_16x16, or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, ~~and~~ subMbPartIdx, and the reference picture lists refPicList0 and refPicList1.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The derivation process depends on the value of direct_spatial_mv_pred_flag, which is present in the bitstream in the slice header syntax as specified in subclause 7.3.3, and is specified as follows.

– If direct_spatial_mv_pred_flag is equal to 1, the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode.

– Otherwise (direct_spatial_mv_pred_flag is equal to 0), mode in which the outputs of this process are derived is referred to as temporal direct prediction mode.

Both spatial and temporal direct prediction mode use the co-located motion vectors and reference indices as specified in subclause 8.4.1.2.1 with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale as output.

The motion vectors and reference indices are derived as follows.

– If spatial direct prediction mode is used, the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.2 is used with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output~~, with subMvCnt being an output~~.

– Otherwise (temporal direct prediction mode is used), the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.3 is used with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output and the variable subMvCnt is derived as follows.

  – If subMbPartIdx is equal to 0, subMvCnt is set equal to 2.

  – Otherwise (subMbPartIdx is not equal to 0), subMvCnt is set equal to 0.

8.4.1.2.1 Derivation process for the co-located 4x4 sub-macroblock partitions

Inputs to this process are mbPartIdx, ~~and~~ subMbPartIdx, and the reference picture lists refPicList0 and refPicList1.

Outputs of this process are the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale (which can be One_To_One, Frm_To_Fld or Fld_To_Frm).

When ~~RefPicList1~~refPicList1[0] is a frame or a complementary field pair, let firstRefPicL1Top and firstRefPicL1Bottom be the top and bottom fields of ~~RefPicList1~~refPicList1[0], respectively, and let the following variables be specified as

topAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Top, CurrPic)) (8-181)

bottomAbsDiffPOC = Abs(DiffPicOrderCnt(firstRefPicL1Bottom, CurrPic)) (8-182)

The variable colPic specifies the picture that contains the co-located macroblock as specified in Table 8-6.

Table 8-6 – Specification of the variable colPic

field_pic_flag	~~RefPicList1~~refPicList1[0] is …	mb_field_decoding_flag	additional condition	colPic
1	a field of a decoded frame			the frame containing ~~RefPicList1~~refPicList1[0]
	a decoded field			~~RefPicList1~~refPicList1[0]
0	a decoded frame			~~RefPicList1~~refPicList1[0]
	a complementary field pair	0	topAbsDiffPOC < bottomAbsDiffPOC	firstRefPicL1Top
			topAbsDiffPOC >= bottomAbsDiffPOC	firstRefPicL1Bottom
		1	(CurrMbAddr & 1) = = 0	firstRefPicL1Top
			(CurrMbAddr & 1) != 0	firstRefPicL1Bottom

When direct_8x8_inference_flag is equal to 1, subMbPartIdx is set as follows.

subMbPartIdx = mbPartIdx (8-183)

Let PicCodingStruct(X) be a function with the argument X being either CurrPic or colPic. It is specified in Table 8-7.

Table 8-7 – Specification of PicCodingStruct(X)

X is coded with field_pic_flag equal to …	mb_adaptive_frame_field_flag	PicCodingStruct(X)
1		FLD
0	0	FRM
0	1	AFRM

With luma4x4BlkIdx = mbPartIdx * 4 + subMbPartIdx, the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) assigned to (xCol, yCol) as the output.

Table 8-8 specifies the co-located macroblock address mbAddrCol, yM, and the variable vertMvScale in two steps:

1. Specification of a macroblock address mbAddrX depending on PicCodingStruct(CurrPic), and PicCodingStruct(colPic).

   NOTE – It is not possible for CurrPic and colPic picture coding types to be either (FRM, AFRM) or (AFRM, FRM) because these picture coding types must be separated by an IDR picture.

2. Specification of mbAddrCol, yM, and vertMvScale depending on mb_field_decoding_flag and the variable fieldDecodingFlagX, which is derived as follows.

   – If the macroblock mbAddrX in the picture colPic is a field macroblock, fieldDecodingFlagX is set equal to 1

   – Otherwise (the macroblock mbAddrX in the picture colPic is a frame macroblock), fieldDecodingFlagX is set equal to 0.

Unspecified values in Table 8-8 indicate that the value of the corresponding variable is not relevant for the current table row.

mbAddrCol is set equal to CurrMbAddr or to one of the following values.

mbAddrCol1 = 2 * PicWidthInMbs * (CurrMbAddr / PicWidthInMbs) +
(CurrMbAddr % PicWidthInMbs) + PicWidthInMbs * (yCol / 8) (8-184)

mbAddrCol2 = 2 * CurrMbAddr + (yCol / 8) (8-185)

mbAddrCol3 = 2 * CurrMbAddr + bottom_field_flag (8-186)

mbAddrCol4 = PicWidthInMbs * (CurrMbAddr / (2 * PicWidthInMbs)) +
(CurrMbAddr % PicWidthInMbs) (8-187)

mbAddrCol5 = CurrMbAddr / 2 (8-188)

mbAddrCol6 = 2 * (CurrMbAddr / 2) + ((topAbsDiffPOC < bottomAbsDiffPOC) ? 0 : 1) (8-189)

mbAddrCol7 = 2 * (CurrMbAddr / 2) + (yCol / 8) (8-190)

Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale

PicCodingStruct(CurrPic)	PicCodingStruct(colPic)	mbAddrX	mb_field_decoding_flag	fieldDecodingFlagX	mbAddrCol	yM	vertMvScale
FLD	FLD				CurrMbAddr	yCol	One_To_One
	FRM				mbAddrCol1	(2 * yCol) % 16	Frm_To_Fld
	AFRM	2*CurrMbAddr		0	mbAddrCol2	(2 * yCol) % 16	Frm_To_Fld
				1	mbAddrCol3	yCol	One_To_One
FRM	FLD				mbAddrCol4	8 * ((CurrMbAddr / PicWidthInMbs) % 2) + 4 * (yCol / 8)	Fld_To_Frm
	FRM				CurrMbAddr	yCol	One_To_One
AFRM	FLD		0		mbAddrCol5	8 * (CurrMbAddr % 2) +4 * (yCol / 8)	Fld_To_Frm
			1		mbAddrCol5	yCol	One_To_One
	AFRM	CurrMbAddr	0	0	CurrMbAddr	yCol	One_To_One
				1	mbAddrCol6	8 * (CurrMbAddr % 2) + 4 * (yCol / 8)	Fld_To_Frm
		CurrMbAddr	1	0	mbAddrCol7	(2 * yCol) % 16	Frm_To_Fld
				1	CurrMbAddr	yCol	One_To_One

Let mbPartIdxCol be the macroblock partition index of the co-located partition and subMbPartIdxCol the sub-macroblock partition index of the co-located sub-macroblock partition. The partition in the macroblock mbAddrCol inside the picture colPic covering the sample (xCol, yM) is assigned to mbPartIdxCol and the sub-macroblock partition inside the partition mbPartIdxCol covering the sample (xCol, yM) in the macroblock mbAddrCol inside the picture colPic is assigned to subMbPartIdxCol.

The prediction utilization flags predFlagL0Col and predFlagL1Col are set equal to PredFlagL0[mbPartIdxCol] and PredFlagL1[mbPartIdxCol], respectively, which are the prediction utilization flags that have been assigned to the macroblock partition mbAddrCol\mbPartIdxCol inside the picture colPic.

The motion vector mvCol and the reference index refIdxCol are derived as follows.

– If the macroblock mbAddrCol is coded in Intra macroblock prediction mode or both prediction utilization flags, predFlagL0Col and predFlagL1Col are equal to 0, both components of mvCol are set equal to 0 and refIdxCol is set equal to –1.

– Otherwise, the following applies.

  – If predFlagL0Col is equal to 1, the motion vector mvCol and the reference index refIdxCol are set equal to MvL0[mbPartIdxCol][subMbPartIdxCol] and RefIdxL0[mbPartIdxCol], respectively, which are the motion vector mvL0 and the reference index refIdxL0 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

– Otherwise (predFlagL0Col is equal to 0 and predFlagL1Col is equal to 1), the motion vector mvCol and the reference index refIdxCol are set equal to MvL1[mbPartIdxCol][subMbPartIdxCol] and RefIdxL1[mbPartIdxCol], respectively, which are the motion vector mvL1 and the reference index refIdxL1 that have been assigned to the (sub-)macroblock partition mbAddrCol\mbPartIdxCol\subMbPartIdxCol inside the picture colPic.

8.4.1.2.2 Derivation process for spatial direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 1 and any of the following conditions is true.

– mb_type is equal to B_Skip

– mb_type is equal to B_Direct_16x16

– sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The reference indices refIdxL0 and refIdxL1 and the variable directZeroPredictionFlag are derived by applying the following ordered steps.

1. Let the variable currSubMbType be set equal to sub_mb_type[mbPartIdx].
2. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 0 as inputs and the output is assigned to the motion vectors mvL0N and the reference indices refIdxL0N with N being replaced by A, B, or C.
3. The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx = 0, subMbPartIdx = 0, currSubMbType, and listSuffixFlag = 1 as inputs and the output is assigned to the motion vectors mvL1N and the reference indices refIdxL1N with N being replaced by A, B, or C.

   NOTE 1 – The motion vectors mvL0N, mvL1N and the reference indices refIdxL0N, refIdxL1N are identical for all 4x4 sub-macroblock partitions of a macroblock.

4. The reference indices refIdxL0, refIdxL1, and directZeroPredictionFlag are derived by

$$refIdxL0 = MinPositive(refIdxL0A, MinPositive(refIdxL0B, refIdxL0C)) \quad (8\text{-}191)$$
$$refIdxL1 = MinPositive(refIdxL1A, MinPositive(refIdxL1B, refIdxL1C)) \quad (8\text{-}192)$$
$$directZeroPredictionFlag = 0 \quad (8\text{-}193)$$

where

$$MinPositive(x, y) = \begin{cases} Min(x, y) & \text{if } x >= 0 \text{ and } y >= 0 \\ Max(x, y) & \text{otherwise} \end{cases} \quad (8\text{-}194)$$

5. When both reference indices refIdxL0 and refIdxL1 are less than 0,

$$refIdxL0 = 0 \quad (8\text{-}195)$$
$$refIdxL1 = 0 \quad (8\text{-}196)$$
$$directZeroPredictionFlag = 1 \quad (8\text{-}197)$$

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx given as input and the output is assigned to refIdxCol and mvCol.

The variable colZeroFlag is derived as follows.

– If all of the following conditions are true, colZeroFlag is set equal to 1.

  – ~~RefPicList1~~refPicList1[0] is currently marked as "used for short-term reference".

  – refIdxCol is equal to 0

- both motion vector components mvCol[0] and mvCol[1] lie in the range of -1 to 1 in units specified as follows.
  - If the co-located macroblock is a frame macroblock, the units of mvCol[0] and mvCol[1] are units of quarter luma frame samples.
  - Otherwise (the co-located macroblock is a field macroblock), the units of mvCol[0] and mvCol[1] are units of quarter luma field samples.

NOTE 2 – For purposes of determining the condition above, the value mvCol[1] is not scaled to use the units of a motion vector for the current macroblock in cases when the current macroblock is a frame macroblock and the co-located macroblock is a field macroblock or when the current macroblock is a field macroblock and the co-located macroblock is a frame macroblock. This aspect differs from the use of mvCol[1] in the temporal direct mode as specified in subclause 8.4.1.2.3, which applies scaling to the motion vector of the co-located macroblock to use the same units as the units of a motion vector for the current macroblock, using Equation 8-200 or Equation 8-201 in these cases.

- Otherwise, colZeroFlag is set equal to 0.

The motion vectors mvLX (with X being 0 or 1) are derived as follows.

- If any of the following conditions is true, both components of the motion vector mvLX are set equal to 0.
  - directZeroPredictionFlag is equal to 1
  - refIdxLX is less than 0
  - refIdxLX is equal to 0 and colZeroFlag is equal to 1
- Otherwise, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxLX, and currSubMbType as inputs and the output is assigned to mvLX.

NOTE 3 – The motion vector mvLX returned from subclause 8.4.1.3 is identical for all 4x4 sub-macroblock partitions of a macroblock for which the process is invoked.

The prediction utilization flags predFlagL0 and predFlagL1 are derived as specified using Table 8-9.

Table 8-9 – Assignment of prediction utilization flags

refIdxL0	refIdxL1	predFlagL0	predFlagL1
>= 0	>= 0	1	1
>= 0	< 0	1	0
< 0	>= 0	0	1

The variable subMvCnt is derived as follows.

- If subMbPartIdx is not equal to 0 or direct_8x8_inference_flag is equal to 0, subMvCnt is set equal to 0.
- Otherwise (subMbPartIdx is equal to 0 and direct_8x8_inference_flag is equal to 1), subMvCnt is set equal to predFlagL0 + predFLagL1.

8.4.1.2.3 Derivation process for temporal direct luma motion vector and reference index prediction mode

This process is invoked when direct_spatial_mv_pred_flag is equal to 0 and any of the following conditions is true.

- mb_type is equal to B_Skip
- mb_type is equal to B_Direct_16x16
- sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, ~~and~~ subMbPartIdx, and the reference picture lists refPicList0 and refPicList1.

Outputs of this process are the motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags, predFlagL0 and predFlagL1.

The process specified in subclause 8.4.1.2.1 is invoked with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 given as input and the output is assigned to colPic, mbAddrCol, mvCol, refIdxCol, and vertMvScale.

The reference indices refIdxL0 and refIdxL1 are derived as follows.

refIdxL0 = ((refIdxCol < 0) ? 0 : MapColToList0(refIdxCol)) (8-198)

refIdxL1 = 0 (8-199)

NOTE 1 – If the current macroblock is a field macroblock, refIdxL0 and refIdxL1 index a list of fields; otherwise (the current macroblock is a frame macroblock), refIdxL0 and refIdxL1 index a list of frames or complementary reference field pairs.

Let refPicCol be a frame, a field, or a complementary field pair that was referred by the reference index refIdxCol when decoding the co-located macroblock mbAddrCol inside the picture colPic. The function MapColToList0(refIdxCol) is specified as follows.

- If vertMvScale is equal to One_To_One, the following applies.
  - If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
    - Let refIdxL0Frm be the lowest valued reference index in the current reference picture list ~~RefPicList0~~ refPicList0 that references the frame or complementary field pair that contains the field refPicCol. ~~RefPicList0~~ refPicList0 shall contain a frame or complementary field pair that contains the field refPicCol. The return value of MapColToList0( ) is specified as follows.
      - If the field referred to by refIdxCol has the same parity as the current macroblock, MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1).
      - Otherwise (the field referred by refIdxCol has the opposite parity of the current macroblock), MapColToList0(refIdxCol) returns the reference index ((refIdxL0Frm << 1) + 1).
  - Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list ~~RefPicList0~~ refPicList0 that references refPicCol. ~~RefPicList0~~ refPicList0 shall contain refPicCol.
- Otherwise, if vertMvScale is equal to Frm_To_Fld, the following applies.
  - If field_pic_flag is equal to 0, let refIdxL0Frm be the lowest valued reference index in the current reference picture list ~~RefPicList0~~ refPicList0 that references refPicCol. MapColToList0(refIdxCol) returns the reference index (refIdxL0Frm << 1). ~~RefPicList0~~ refPicList0 shall contain refPicCol.
  - Otherwise (field_pic_flag is equal to 1), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list ~~RefPicList0~~ refPicList0 that references the field of refPicCol with the same parity as the current picture CurrPic. ~~RefPicList0~~ refPicList0 shall contain the field of refPicCol with the same parity as the current picture CurrPic.
- Otherwise (vertMvScale is equal to Fld_To_Frm), MapColToList0(refIdxCol) returns the lowest valued reference index refIdxL0 in the current reference picture list ~~RefPicList0~~ refPicList0 that references the frame or complementary field pair that contains refPicCol. ~~RefPicList0~~ refPicList0 shall contain a frame or complementary field pair that contains the field refPicCol.

NOTE 2 – A decoded reference picture that was marked as "used for short-term reference" when it was referenced in the decoding process of the picture containing the co-located macroblock may have been modified to be marked as "used for long-term reference" before being used for reference for inter prediction using the direct prediction mode for the current macroblock.

Depending on the value of vertMvScale the vertical component of mvCol is modified as follows.

- If vertMvScale is equal to Frm_To_Fld

mvCol[1] = mvCol[1] / 2 (8-200)

- Otherwise, if vertMvScale is equal to Fld_To_Frm

mvCol[1] = mvCol[1] * 2 (8-201)

– Otherwise (vertMvScale is equal to One_To_One), mvCol[1] remains unchanged.

The variables currPicOrField, pic0, and pic1, are derived as follows.

– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.
  – currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.
  – pic1 is the field of RefPicList1[0] that has the same parity as the current macroblock.
  – The variable pic0 is derived as follows.
    – If refIdxL0 % 2 is equal to 0, pic0 is the field of ~~RefPicList0~~refPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.
    – –Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of ~~RefPicList0~~refPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.
– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is the decoded reference picture ~~RefPicList1~~refPicList1[0], and pic0 is the decoded reference picture ~~RefPicList0~~refPicList0[refIdxL0].

The two motion vectors mvL0 and mvL1 for each 4x4 sub-macroblock partition of the current macroblock are derived as follows:

NOTE 3 – It is often the case that many of the 4x4 sub-macroblock partitions share the same motion vectors and reference pictures. In these cases, temporal direct mode motion compensation can calculate the inter prediction sample values in larger units than 4x4 luma sample blocks. For example, when direct_8x8_inference_flag is equal to 1, at least each 8x8 luma sample quadrant of the macroblock shares the same motion vectors and reference pictures.

– If the reference index refIdxL0 refers to a long-term reference picture, or DiffPicOrderCnt(pic1, pic0) is equal to 0, the motion vectors mvL0, mvL1 for the direct mode partition are derived by

mvL0 = mvCol (8-202)

mvL1 = 0 (8-203)

– Otherwise, the motion vectors mvL0, mvL1 are derived as scaled versions of the motion vector mvCol of the co-located sub-macroblock partition as specified below (see Figure 8-2)

tx = (16 384 + Abs(td / 2)) / td (8-204)

DistScaleFactor = Clip3(-1024, 1023, (tb * tx + 32) >> 6) (8-205)

mvL0 = (DistScaleFactor * mvCol + 128) >> 8 (8-206)

mvL1 = mvL0 – mvCol (8-207)

where tb and td are derived as follows.

tb = Clip3(-128, 127, DiffPicOrderCnt(currPicOrField, pic0)) (8-208)

td = Clip3(-128, 127, DiffPicOrderCnt(pic1, pic0)) (8-209)

NOTE 4 – mvL0 and mvL1 cannot exceed the ranges specified in Annex A.

The prediction utilization flags predFlagL0 and predFlagL1 are both set equal to 1.

Figure 8-2 illustrates the temporal direct-mode motion vector inference when the current picture is temporally between the reference picture from reference picture list 0 and the reference picture from reference picture list 1.

Figure 8-2 – Example for temporal direct-mode motion vector inference (informative)

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the reference index of the current partition refIdxLX (with X being 0 or 1),

– the variable currSubMbType.

Output of this process is the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = X (with X being 0 or 1 for refIdxLX being refIdxL0 or refIdxL1, respectively) as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX,

$$mvpLX = mvLXB \tag{8-210}$$

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX,

$$mvpLX = mvLXA \tag{8-211}$$

– MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX,

$$mvpLX = mvLXA \tag{8-212}$$

– MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX,

$$mvpLX = mvLXC \tag{8-213}$$

Figure 8-3 illustrates the non-median prediction as described above.

Figure 8-3 – Directional segmentation prediction (informative)

8.4.1.3.1 Derivation process for median luma motion vector prediction

Inputs to this process are

– the neighbouring partitions mbAddrN\mbPartIdxN\subMbPartIdxN (with N being replaced by A, B, or C),

– the motion vectors mvLXN (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference indices refIdxLXN (with N being replaced by A, B, or C) of the neighbouring partitions, and

– the reference index refIdxLX of the current partition.

Output of this process is the motion vector prediction mvpLX.

The variable mvpLX is derived as follows:

– When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

$$mvLXB = mvLXA \tag{8-214}$$

$$mvLXC = mvLXA \tag{8-215}$$

$$refIdxLXB = refIdxLXA \tag{8-216}$$

$$refIdxLXC = refIdxLXA \tag{8-217}$$

– Depending on reference indices refIdxLXA, refIdxLXB, or refIdxLXC, the following applies.

  – If one and only one of the reference indices refIdxLXA, refIdxLXB, or refIdxLXC is equal to the reference index refIdxLX of the current partition, the following applies. Let refIdxLXN be the reference index that is equal to refIdxLX, the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

$$mvpLX = mvLXN \tag{8-218}$$

  – Otherwise, each component of the motion vector prediction mvpLX is given by the median of the corresponding vector components of the motion vector mvLXA, mvLXB, and mvLXC:

mvpLX[0] = Median(mvLXA[0], mvLXB[0], mvLXC[0]) (8-219)

mvpLX[1] = Median(mvLXA[1], mvLXB[1], mvLXC[1]) (8-220)

8.4.1.3.2 Derivation process for motion data of neighbouring partitions

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the current sub-macroblock type currSubMbType,

– the list suffix flag listSuffixFlag

Outputs of this process are (with N being replaced by A, B, or C)

– mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,

– the motion vectors mvLXN of the neighbouring partitions, and

– the reference indices refIdxLXN of the neighbouring partitions.

Variable names that include the string "LX" are interpreted with the X being equal to listSuffixFlag.

The partitions mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, or C are derived in the following ordered steps.

1. Let mbAddrD\mbPartIdxD\subMbPartIdxD be variables specifying an additional neighbouring partition.
2. The process in subclause 6.4.8.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being replaced by A, B, C, or D.
3. When the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies

mbAddrC = mbAddrD (8-221)

mbPartIdxC = mbPartIdxD (8-222)

subMbPartIdxC = subMbPartIdxD (8-223)

The motion vectors mvLXN and reference indices refIdxLXN (with N being A, B, or C) are derived as follows.

– If the macroblock partition or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is not available or mbAddrN is coded in Intra prediction mode or predFlagLX of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvLXN are set equal to 0 and refIdxLXN is set equal to –1.

– Otherwise, the following applies.

  – The motion vector mvLXN and reference index refIdxLXN are set equal to MvLX[mbPartIdxN][subMbPartIdxN] and RefIdxLX[mbPartIdxN], respectively, which are the motion vector mvLX and reference index refIdxLX that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.

  – The variables mvLXN[1] and refIdxLXN are further processed as follows.

    – If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock

mvLXN[1] = mvLXN[1] / 2 (8-224)

refIdxLXN = refIdxLXN * 2 (8-225)

– Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock

mvLXN[1] = mvLXN[1] * 2 (8-226)

refIdxLXN = refIdxLXN / 2 (8-227)

– Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

8.4.1.4 Derivation process for chroma motion vectors

This process is only invoked when chroma_format_idc is not equal to 0 (monochrome).

Inputs to this process are a luma motion vector mvLX and a reference index refIdxLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

The precision of the chroma motion vector components is 1 ÷ (4 * SubWidthC) horizontally and 1 ÷ (4 * SubHeightC) vertically.

NOTE – For example, when using the 4:2:0 chroma format, since the units of luma motion vectors are one-quarter luma sample units and chroma has half horizontal and vertical resolution compared to luma, the units of chroma motion vectors are one-eighth chroma sample units, i.e., a value of 1 for the chroma motion vector refers to a one-eighth chroma sample displacement. For example, when the luma vector applies to 8x16 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 4x8 chroma samples and when the luma vector applies to 4x4 luma samples, the corresponding chroma vector in 4:2:0 chroma format applies to 2x2 chroma samples.

For the derivation of the motion vector mvCLX, the following applies.

– If chroma_format_idc is not equal to 1 or the current macroblock is a frame macroblock, the horizontal and vertical components of the chroma motion vector mvCLX are derived as

mvCLX[0] = mvLX[0] (8-228)
mvCLX[1] = mvLX[1] (8-229)

– Otherwise (chroma_format_idc is equal to 1 and the current macroblock is a field macroblock), only the horizontal component of the chroma motion vector mvCLX[0] is derived using Equation 8-228. The vertical component of the chroma motion vector mvCLX[1] is dependent on the parity of the current field or the current macroblock and the reference picture, which is referred by the reference index refIdxLX. mvCLX[1] is derived from mvLX[1] according to Table 8-10.

Table 8-10 – Derivation of the vertical component of the chroma vector in field coding mode

Parity conditions		**mvCLX[1]**
Reference picture (refIdxLX)	Current field (picture/macroblock)	
Top field	Bottom field	mvLX[1] + 2
Bottom field	Top field	mvLX[1] – 2
Otherwise		mvLX[1]

8.4.2 Decoding process for Inter prediction samples

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available)

– luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags, predFlagL0 and predFlagL1

– reference picture lists refPicList0 and refPicList1 (if available)

– the variables for weighted prediction of luma samples $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L},

– when chroma_format_idc is not equal to 0 (monochrome), the variables for weighted prediction of chroma samples $logWD_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C}.

带格式的: 项目符号和编号

Outputs of this process are

– the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array $predPart_L$ of prediction luma samples, and when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$, $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

Let $predPartL0_L$ and $predPartL1_L$ be (partWidth)x(partHeight) arrays of predicted luma sample values and when chroma_format_idc is not equal to 0 (monochrome) $predPartL0_{Cb}$, $predPartL1_{Cb}$, $predPartL0_{Cr}$, and $predPartL1_{Cr}$ be (partWidthC)x(partHeightC) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, predPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies.

– The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and when chroma_format_idc is not equal to 0 (monochrome) two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.4.2.1 with refIdxLX and ~~RefPicListX~~ refPicListX given as input.

– The array $predPartLX_L$ and when chroma_format_idc is not equal to 0 (monochrome) the arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$ are derived by invoking the process specified in subclause 8.4.2.2 with the current partition specified by mbPartIdx\subMbPartIdx, the motion vectors mvLX, mvCLX (if available), and the reference arrays with $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available) given as input.

For C being replaced by L, Cb (if available), or Cr (if available), the array $predPart_C$ of the prediction samples of component C is derived by invoking the process specified in subclause 8.4.2.3 with the current partition specified by mbPartIdx and subMbPartIdx ~~and~~, the arrays $predPartL0_C$ and $predPartL1_C$, the reference picture lists refPicList0 and refPicList1 as well as the prediction uitilization flags predFlagL0 and predFlagL1 given as input.

8.4.2.1 Reference picture selection process

Inputs to this process ~~is~~ are

– a reference index refIdxLX.

– a reference picture list refPicListX

带格式的: 项目符号和编号

Output of this process is a reference picture consisting of a two-dimensional array of luma samples $refPicLX_L$ and two two-dimensional arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

Depending on field_pic_flag, the reference picture list ~~RefPicListX~~ refPicListX ~~(which has been derived as specified in subclause 8.2.4)~~ consists of the following.

– If field_pic_flag is equal to 1, each entry of ~~RefPicListX~~ refPicListX is a reference field or a field of a reference frame.

– Otherwise (field_pic_flag is equal to 0), each entry of ~~RefPicListX~~ refPicListX is a reference frame or a complementary reference field pair.

For the derivation of the reference picture, the following applies.

– If field_pic_flag is equal to 1, the reference field or field of a reference frame ~~RefPicListX~~refPicListX[refIdxLX] is the output. The output reference field or field of a reference frame consists of a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array of luma samples $refPicLX_L$ and, when chroma_format_idc is not equal to 0 (monochrome), two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

– Otherwise (field_pic_flag is equal to 0), the following applies.

  – If the current macroblock is a frame macroblock, the reference frame or complementary reference field pair ~~RefPicListX~~refPicListX[refIdxLX] is the output. The output reference frame or complementary reference field pair consists of a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array of luma samples $refPicLX_L$ and, when chroma_format_idc is not equal to 0 (monochrome), two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

  – Otherwise (the current macroblock is a field macroblock), the following applies.

    – Let refFrame be the reference frame or complementary reference field pair ~~RefPicListX~~refPicListX[refIdxLX / 2].

    – The field of refFrame is selected as follows.

      – If refIdxLX % 2 is equal to 0, the field of refFrame that has the same parity as the current macroblock is the output.

      – Otherwise (refIdxLX % 2 is equal to 1), the field of refFrame that has the opposite parity as the current macroblock is the output.

    – The output reference field or field of a reference frame consists of a ($PicWidthInSamples_L$)x($PicHeightInSamples_L$ / 2) array of luma samples $refPicLX_L$ and, when chroma_format_idc is not equal to 0 (monochrome), two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$ / 2) arrays of chroma samples $refPicLX_{Cb}$ and $refPicLX_{Cr}$.

The reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available) correspond to decoded sample arrays S_L, S_{Cb} (if available), S_{Cr} (if available) derived in subclause 8.7 for a previously-decoded reference field or reference frame or complementary reference field pair or field of a reference frame.

8.4.2.2 Fractional sample interpolation process

Inputs to this process are

– the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,

– the width and height partWidth, partHeight of this partition in luma-sample units,

– a luma motion vector mvLX given in quarter-luma-sample units,

– a chroma motion vector mvCLX given in eighth-chroma-sample units, and

– the selected reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cb}$

Outputs of this process are

– a (partWidth)x(partHeight) array $predPartLX_L$ of prediction luma sample values and

– when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays $predPartLX_{Cb}$, and $predPartLX_{Cr}$ of prediction chroma sample values.

Let (xA_L, yA_L) be the location given in full-sample units of the upper-left luma sample of the current partition given by mbPartIdx\subMbPartIdx relative to the upper-left luma sample location of the given two-dimensional array of luma samples.

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_L$, $refPicLX_{Cb}$ (if available), and $refPicLX_{Cr}$ (if available).

For each luma sample location ($0 <= x_L < partWidth$, $0 <= y_L < partHeight$) inside the prediction luma sample array $predPartLX_L$, the corresponding prediction luma sample value $predPartLX_L[x_L, y_L]$ is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by

$$xInt_L = xA_L + (mvLX[0] >> 2) + x_L \quad (8\text{-}230)$$
$$yInt_L = yA_L + (mvLX[1] >> 2) + y_L \quad (8\text{-}231)$$

$$xFrac_L = mvLX[0] \& 3 \quad (8\text{-}232)$$
$$yFrac_L = mvLX[1] \& 3 \quad (8\text{-}233)$$

– The prediction luma sample value $predPartLX_L[x_L, y_L]$ is derived by invoking the process specified in subclause 8.4.2.2.1 with ($xInt_L$, $yInt_L$), ($xFrac_L$, $yFrac_L$) and $refPicLX_L$ given as input.

When chroma_format_idc is not equal to 0 (monochrome), the following applies.

Let ($xInt_C$, $yInt_C$) be a chroma location given in full-sample units and ($xFrac_C$, $yFrac_C$) be an offset given in one-eighth sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

For each chroma sample location (0 <= x_C < partWidthC, 0 <= y_C < partHeightC) inside the prediction chroma sample arrays $predPartLX_{Cb}$ and $predPartLX_{Cr}$, the corresponding prediction chroma sample values $predPartLX_{Cb}[x_C, y_C]$ and $predPartLX_{Cr}[x_C, y_C]$ are derived as follows:

– Depending on chroma_format_idc, the variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived as follows.

  – If chroma_format_idc is equal to 1,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}234)$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 3) + y_C \quad (8\text{-}235)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8\text{-}236)$$
$$yFrac_C = mvCLX[1] \& 7 \quad (8\text{-}237)$$

  – Otherwise, if chroma_format_idc is equal to 2,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 3) + x_C \quad (8\text{-}238)$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 2) + y_C \quad (8\text{-}239)$$

$$xFrac_C = mvCLX[0] \& 7 \quad (8\text{-}240)$$
$$yFrac_C = (mvCLX[1] \& 3) << 1 \quad (8\text{-}241)$$

  – Otherwise (chroma_format_idc is equal to 3),

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 2) + x_C \quad (8\text{-}242)$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 2) + y_C \quad (8\text{-}243)$$

$$xFrac_C = (mvCLX[0] \& 3) << 1 \quad (8\text{-}244)$$
$$yFrac_C = (mvCLX[1] \& 3) << 1 \quad (8\text{-}245)$$

– The prediction sample value $predPartLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cb}$ given as input.

– The prediction sample value $predPartLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause 8.4.2.2.2 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $refPicLX_{Cr}$ given as input.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and

– the luma sample array of the selected reference picture $refPicLX_L$

Output of this process is a predicted luma sample value $predPartLX_L[\ x_L, y_L\ ]$.

Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

The variable $refPicHeightEffective_L$, which is the height of the effective reference picture luma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$ / 2.

In Figure 8-4, the positions labelled with upper-case letters within shaded blocks represent luma samples at full-sample locations inside the given two-dimensional array $refPicLX_L$ of luma samples. These samples may be used for generating the predicted luma sample value $predPartLX_L[\ x_L, y_L\ ]$. The locations (xZ_L, yZ_L) for each of the corresponding luma samples Z, where Z may be A, B, C, D, E, F, G, H, I, J, K, L, M, N, P, Q, R, S, T, or U, inside the given array $refPicLX_L$ of luma samples are derived as follows:

$$xZ_L = Clip3(\ 0, PicWidthInSamples_L - 1, xInt_L + xDZ_L\ )$$
$$yZ_L = Clip3(\ 0, refPicHeightEffective_L - 1, yInt_L + yDZ_L\ ) \qquad (8\text{-}246)$$

Table 8-11 specifies (xDZ_L, yDZ_L) for different replacements of Z.

Table 8-11 – Differential full-sample luma locations

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
xDZ_L	0	1	0	1	-2	-1	0	1	2	3	-2	-1	0	1	2	3	0	1	0	1
yDZ_L	-2	-2	-1	-1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

Given the luma samples 'A' to 'U' at full-sample locations (xA_L, yA_L) to (xU_L, yU_L), the luma samples 'a' to 's' at fractional sample positions are derived by the following rules. The luma prediction values at half sample positions are derived by applying a 6-tap filter with tap values (1, -5, 20, 20, -5, 1). The luma prediction values at quarter sample positions are derived by averaging samples at full and half sample positions. The process for each fractional position is described below.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as b_1 by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as h_1 by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}247)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}248)$$

– The final prediction values b and h are derived using:

$$b = Clip1_Y((b_1 + 16) >> 5) \quad (8\text{-}249)$$
$$h = Clip1_Y((h_1 + 16) >> 5) \quad (8\text{-}250)$$

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}251)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}252)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (8\text{-}253)$$

– The final prediction values s and m are derived from s_1 and m_1 in the same manner as the derivation of b and h, as given by:

$$s = Clip1_Y((s_1 + 16) >> 5) \quad (8\text{-}254)$$
$$m = Clip1_Y((m_1 + 16) >> 5) \quad (8\text{-}255)$$

– The samples at quarter sample positions labelled as a, c, d, n, f, i, k, and q are derived by averaging with upward rounding of the two nearest samples at integer and half sample positions using:

$$a = (G + b + 1) >> 1 \quad (8\text{-}256)$$
$$c = (H + b + 1) >> 1 \quad (8\text{-}257)$$
$$d = (G + h + 1) >> 1 \quad (8\text{-}258)$$
$$n = (M + h + 1) >> 1 \quad (8\text{-}259)$$
$$f = (b + j + 1) >> 1 \quad (8\text{-}260)$$
$$i = (h + j + 1) >> 1 \quad (8\text{-}261)$$
$$k = (j + m + 1) >> 1 \quad (8\text{-}262)$$
$$q = (j + s + 1) >> 1. \quad (8\text{-}263)$$

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}264)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}265)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}266)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}267)$$

The luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$) specifies which of the generated luma samples at full-sample and fractional-sample locations is assigned to the predicted luma sample value $predPartLX_L[x_L, y_L]$. This assignment is done according to Table 8-12. The value of $predPartLX_L[x_L, y_L]$ is the output.

Table 8-12 – Assignment of the luma prediction sample $predPartLX_L[x_L, y_L]$

$xFrac_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$yFrac_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$predPartLX_L[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

8.4.2.2.2 Chroma sample interpolation process

This process shall only be invoked when chroma_format_idc is not equal to 0 (monochrome).

Inputs to this process are

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location offset in fractional-sample units ($xFrac_C$, $yFrac_C$), and

– chroma component samples from the selected reference picture $refPicLX_C$.

Output of this process is a predicted chroma sample value $predPartLX_C[x_C, y_C]$.

In Figure 8-5, the positions labelled with A, B, C, and D represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLX_C$ of chroma samples.

Figure 8-5 – Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D

The variable $refPicHeightEffective_C$, which is the height of the effective reference picture chroma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$ / 2.

The sample coordinates specified in Equations 8-268 through 8-275 are used for generating the predicted chroma sample value $predPartLX_C[\ x_C, y_C\ ]$.

$$xA_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C\ ) \quad (8\text{-}268)$$
$$xB_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C + 1\ ) \quad (8\text{-}269)$$
$$xC_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C\ ) \quad (8\text{-}270)$$
$$xD_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C + 1\ ) \quad (8\text{-}271)$$

$$yA_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C\ ) \quad (8\text{-}272)$$
$$yB_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C\ ) \quad (8\text{-}273)$$
$$yC_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C + 1\ ) \quad (8\text{-}274)$$
$$yD_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C + 1\ ) \quad (8\text{-}275)$$

Given the chroma samples A, B, C, and D at full-sample locations specified in Equations 8-268 through 8-275, the predicted chroma sample value $predPartLX_C[\ x_C, y_C\ ]$ is derived as follows:

$$predPartLX_C[\ x_C, y_C\ ] = (\ (\ 8 - xFrac_C\ ) * (\ 8 - yFrac_C\ ) * A + xFrac_C * (\ 8 - yFrac_C\ ) * B + (\ 8 - xFrac_C\ ) * yFrac_C * C + xFrac_C * yFrac_C * D + 32\ ) >> 6 \quad (8\text{-}276)$$

8.4.2.3 Weighted sample prediction process

Inputs to this process are

– mbPartIdx: the current partition given by the partition index

– subMbPartIdx: the sub-macroblock partition index

– predFlagL0 and predFlagL1: prediction list utilization flags

– reference picture lists refPicListX (with X being replaced by 0 or 1 depending on predFlagL0 and predFlagL1)

– $predPartLX_L$: a (partWidth)x(partHeight) array of prediction luma samples (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)

– when chroma_format_idc is not equal to 0 (monochrome), $predPartLX_{Cb}$ and $predPartLX_{Cr}$: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr (with LX being replaced by L0 or L1 depending on predFlagL0 and predFlagL1)

– weighting parameters of prediction luma samples $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L},

– when chroma_format_idc is not equal to 0 (monochrome), weighting parameters of prediction chroma samples $logWD_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C},

带格式的: 项目符号和编号

带格式的: 项目符号和编号

Outputs of this process are

– $predPart_L$: a (partWidth)x(partHeight) array of prediction luma samples and

– when chroma_format_idc is not equal to 0 (monochrome), $predPart_{Cb}$, and $predPart_{Cr}$: (partWidthC)x(partHeightC) arrays of prediction chroma samples, one for each of the chroma components Cb and Cr.

For macroblocks or partitions with predFlagL0 equal to 1 in P and SP slices, the following applies.

– If weighted_pred_flag is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

– Otherwise (weighted_pred_flag is equal to 1), the explicit weighted prediction process as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

For macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 in B slices, the following applies.

– If weighted_bipred_idc is equal to 0, the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

– Otherwise, if weighted_bipred_idc is equal to 1, the explicit weighted sample prediction process as described in subclause 8.4.2.3.2, for macroblocks or partitions with predFlagL0 or predFlagL1 equal to 1 with the same inputs and outputs as the process described in this subclause.

– Otherwise (weighted_bipred_idc is equal to 2), the following applies.

  – If predFlagL0 is equal to 1 and predFlagL1 is equal to 1, the implicit weighted sample prediction as described in subclause 8.4.2.3.2 is invoked with the same inputs and outputs as the process described in this subclause.

  – Otherwise (predFlagL0 or predFlagL1 are equal to 1 but not both), the default weighted sample prediction process as described in subclause 8.4.2.3.1 is invoked with the same inputs and outputs as the process described in this subclause.

8.4.2.3.1 Default weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies.

– If the luma sample prediction values $predPart_L[x, y]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

The prediction sample values are derived as follows.

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0 for the current partition

$$predPart_C[x, y] = predPartL0_C[x, y] \quad (8\text{-}277)$$

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1 for the current partition

$$predPart_C[x, y]= predPartL1_C[x, y] \quad (8\text{-}278)$$

– Otherwise (predFlagL0 and predFlagL1 are equal to 1 for the current partition),

$$predPart_C[x, y] = (predPartL0_C[x, y] + predPartL1_C[x, y] + 1) >> 1. \quad (8\text{-}279)$$

8.4.2.3.2 Weighted sample prediction process

Input to this process are the same as specified in subclause 8.4.2.3.

Output of this process are the same as specified in subclause 8.4.2.3.

Depending on the available component for which the prediction block is derived, the following applies.

– If the luma sample prediction values $predPart_L[x, y]$ are derived, the following applies with C set equal to L, x set equal to 0 .. partWidth - 1, and y set equal to 0 .. partHeight - 1.

– Otherwise, if the chroma Cb component sample prediction values $predPart_{Cb}[x, y]$ are derived, the following applies with C set equal to Cb, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

– Otherwise (the chroma Cr component sample prediction values $predPart_{Cr}[x, y]$ are derived), the following applies with C set equal to Cr, x set equal to 0 .. partWidthC - 1, and y set equal to 0 .. partHeightC - 1.

The prediction sample values are derived as follows

– If the partition mbPartIdx\subMbPartIdx has predFlagL0 equal to 1 and predFlagL1 equal to 0, the final predicted sample values $predPart_C[x, y]$ are derived by

if(logWD >= 1)
predPart$_C$[x, y] = Clip1$_C$(((predPartL0$_C$[x, y] * w_{0C} + (1 << (~~2^logWD_C~~ – logWD$_C$ – 1))) >> logWD$_C$) + o_{0C})
else (8-280)
predPart$_C$[x, y] = Clip1$_C$(predPartL0$_C$[x, y] * w_{0C} + o_{0C})

– Otherwise, if the partition mbPartIdx\subMbPartIdx has predFlagL0 equal to 0 and predFlagL1 equal to 1, the final predicted sample values predPart$_C$[x, y] are derived by

if(logWD >= 1)
predPart$_C$[x, y] = Clip1$_C$(((predPartL1$_C$[x, y] * w_{1C} + (1 << (logWD$_C$ – 1)) ~~2^(logWD – 1)~~) >> logWD$_C$) + o_{1C})
else (8-281)
predPart$_C$[x, y] = Clip1$_C$(predPartL1$_C$[x, y] * w_{1C} + o_{1C})

– Otherwise (the partition mbPartIdx\subMbPartIdx has both predFlagL0 and predFlagL1 equal to 1), the final predicted sample values predPart$_C$[x, y] are derived by

predPart$_C$[x, y] = Clip1$_C$(((predPartL0$_C$[x, y] * w_{0C} + predPartL1$_C$[x, y] * w_{1C} + (1 << logWD$_C$) ~~2^logWD~~) >> (logWD$_C$ + 1)) + ((o_{0C} + o_{1C} + 1) >> 1)) (8-282)

8.4.3 Derivation process for prediction weights

Inputs to this process are

– the partition index mbPartIdx

– the sub-macroblock partition index subMbPartIdx

– reference picture lists refPicList0 and refPicList1

– prediction list utilization flags, predFlagL0 and predFlagL1

Outputs of this process are

– variables for weighted prediction of luma samples logWD$_L$, w_{0L}, w_{1L}, o_{0L}, o_{0L}

– when chroma_format_idc is not equal to 0 (monochrome), variables for weighted prediction of chroma samples logWD$_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C}.

The output variables of this process are derived as follows.

– If weighted_bipred_idc is equal to 2 and the slice_type is equal to B, implicit mode weighted prediction is used as follows with C set equal to L for luma, Cb and Cr for each of the chroma components.

logWD$_C$ = 5 (8-283)

o_{0C} = 0 (8-284)

o_{1C} = 0 (8-285)

and w_{0C} and w_{1C} are derived as follows.

– The variables currPicOrField, pic0, and pic1 are derived as follows:

– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.

– currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.

带格式的: 非上标/ 下标
带格式的: 非上标/ 下标
带格式的: 非上标/ 下标
带格式的: 项目符号和编号
带格式的: 项目符号和编号
带格式的: 项目符号和编号
带格式的: 项目符号和编号
带格式的: 项目符号和编号

– The variable pic0 is derived as follows.

– If refIdxL0 % 2 is equal to 0, pic0 is the field of refPicList0[refIdxL0 / 2] that has the same parity as the current macroblock.

– Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of refPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock.

– The variable pic1 is derived as follows.

– If refIdxL1 % 2 is equal to 0, pic1 is the field of refPicList1[refIdxL1 / 2] that has the same parity as the current macroblock.

– Otherwise (refIdxL1 % 2 is not equal to 0), pic1 is the field of refPicList1[refIdxL1 / 2] that has the opposite parity of the current macroblock.

– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is refPicList1[refIdxL1], and pic0 is refPicList0[refIdxL0].

– The variables tb, td, tx, and DistScaleFactor are derived from the values of currPicOrField, pic0, pic1 using Equations 8-208, 8-209, 8-204, and 8-205, respectively.

– If DiffPicOrderCnt(pic1, pic0) is equal to 0 or one or both of pic1 and pic0 is marked as "used for long-term reference" or (DistScaleFactor >> 2) < -64 or (DistScaleFactor >> 2) > 128, w_{0C} and w_{1C} are derived as

$$w_{0C} = 32 \quad (8\text{-}286)$$

$$w_{1C} = 32 \quad (8\text{-}287)$$

– Otherwise,

$$w_{0C} = 64 - (DistScaleFactor >> 2) \quad (8\text{-}288)$$

$$w_{1C} = DistScaleFactor >> 2 \quad (8\text{-}289)$$

– Otherwise (weighted_pred_flag is equal to 1 in P or SP slices or weighted_bipred_idc equal to 1 in B slices), explicit mode weighted prediction is used as follows.

– The variables refIdxL0WP and refIdxL1WP are derived as follows.

– If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$refIdxL0WP = refIdxL0 >> 1 \quad (8\text{-}290)$$

$$refIdxL1WP = refIdxL1 >> 1 \quad (8\text{-}291)$$

– Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$refIdxL0WP = refIdxL0 \quad (8\text{-}292)$$

$$refIdxL1WP = refIdxL1 \quad (8\text{-}293)$$

– The variables $logWD_C$, w_{0C}, w_{1C}, o_{0C}, and o_{1C} are derived as follows.

– If C is set equal to L for luma samples

$$logWD_C = luma_log2_weight_denom \quad (8\text{-}294)$$

带格式的: 项目符号和编号

带格式的: 项目符号和编号

带格式的: 项目符号和编号

带格式的: 项目符号和编号

w_{0C} = luma_weight_l0[refIdxL0WP] (8-295)

w_{1C} = luma_weight_l1[refIdxL1WP] (8-296)

o_{0C} = luma_offset_l0[refIdxL0WP] * (1 << ($BitDepth_Y$ – 8)) (8-297)

o_{1C} = luma_offset_l1[refIdxL1WP] * (1 << ($BitDepth_Y$ – 8)) (8-298)

– Otherwise (C is set equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

带格式的：项目符号和编号

$logWD_C$ = chroma_log2_weight_denom (8-299)

w_{0C} = chroma_weight_l0[refIdxL0WP][iCbCr] (8-300)

w_{1C} = chroma_weight_l1[refIdxL1WP][iCbCr] (8-301)

o_{0C} = chroma_offset_l0[refIdxL0WP][iCbCr] * (1 << ($BitDepth_C$ – 8)) (8-302)

o_{1C} = chroma_offset_l1[refIdxL1WP][iCbCr] * (1 << ($BitDepth_C$ – 8)) (8-303)

When explicit mode weighted prediction is used and the partition mbPartIdx\subMbPartIdx has both predFlagL0 and predFlagL1 equal to 1, the following constraint shall be obeyed

-128 <= w_{0C} + w_{1C} <= (($logWD_C$ = = 7) ? 127 : 128) (8-304)

NOTE – For implicit mode weighted prediction, weights w_{0C} and w_{1C} are each guaranteed to be in the range of -64..128 and the constraint expressed in Equation 8-304 although not explicitly imposed, will always be met. For explicit mode weighted prediction with $logWD_C$ equal to 7, when one of the two weights w_{0C} or w_{1C} is inferred to be equal to 128 (as a consequence of luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag equal to 0), the other weight (w_{1C} or w_{0C}) must have a negative value in order for the constraint expressed in Equation 8-304 to hold (and therefore the other flag luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag must be equal to 1).

~~The variables in the above derivation for the prediction samples are derived as follows.~~

~~– If weighted_bipred_idc is equal to 2 and the slice_type is equal to B, implicit mode weighted prediction is used as used as follows.~~

带格式的：项目符号和编号

~~logWD = 5 (8-283)~~

~~o_0 = 0 (8-284)~~

~~o_1 = 0 (8-285)~~

~~and w_0 and w_1 are derived as follows.~~

~~– The variables currPicOrField, pic0, and pic1 are derived as follows:~~

带格式的：项目符号和编号

~~– If field_pic_flag is equal to 0 and the current macroblock is a field macroblock, the following applies.~~

~~– currPicOrField is the field of the current picture CurrPic that has the same parity as the current macroblock.~~

~~– The variable pic0 is derived as follows.~~

~~If refIdxL0 % 2 is equal to 0, pic0 is the field of RefPicList0refPicList0[refIdxL0 / 2] that has the the same parity as the current macroblock.~~

~~Otherwise (refIdxL0 % 2 is not equal to 0), pic0 is the field of RefPicList0refPicList0[refIdxL0 / 2] that has the opposite parity of the current macroblock. macroblock.~~

~~The variable pic1 is derived as follows.~~

~~If refIdxL1 % 2 is equal to 0, pic1 is the field of RefPicList1refPicList1[refIdxL1 / 2] that has the the same parity as the current macroblock.~~

~~Otherwise (refIdxL1 % 2 is not equal to 0), pic1 is the field of RefPicList1refPicList1[refIdxL1 / 2] that has the opposite parity of the current macroblock. macroblock.~~

~~Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock), currPicOrField is the current picture CurrPic, pic1 is RefPicList1refPicList1[refIdxL1], and pic0 is RefPicList0refPicList0[refIdxL0].~~

~~The variables tb, td, tx, and DistScaleFactor are derived from the values of currPicOrField, pic0, pic1 using Equations 8-208, 8-209, 8-204, and 8-205, respectively.~~

~~If DiffPicOrderCnt(pic1, pic0) is equal to 0 or one or both of pic1 and pic0 is marked as "used for long term reference" or (DistScaleFactor >> 2) < -64 or (DistScaleFactor >> 2) > 128, w_0 and w_1 are derived as derived as~~

~~$w_0 = 32$ (8-286)~~

~~$w_1 = 32$ (8-287)~~

~~Otherwise,~~

~~$w_0 = 64 - (DistScaleFactor >> 2)$ (8-288)~~

~~$w_1 = DistScaleFactor >> 2$ (8-289)~~

~~Otherwise (weighted_pred_flag is equal to 1 in P or SP slices or weighted_bipred_idc equal to 1 in B slices), explicit explicit mode weighted prediction is used as follows.~~

~~The variables refIdxL0WP and refIdxL1WP are derived as follows.~~

~~If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock~~

~~refIdxL0WP = refIdxL0 >> 1 (8-290)~~

~~refIdxL1WP = refIdxL1 >> 1 (8-291)~~

~~Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),~~

~~refIdxL0WP = refIdxL0 (8-292)~~

~~refIdxL1WP = refIdxL1 (8-293)~~

~~The variables logWD, w_0, w_1, o_0, and o_1 are derived as follows.~~

~~If C in $predPart_C$[x, y] is replaced by L for luma samples~~

带格式的：项目符号和编号

带格式的：项目符号和编号

带格式的：项目符号和编号

带格式的：项目符号和编号

~~logWD = luma_log2_weight_denom (8-294)~~

~~w_0 = luma_weight_l0[refIdxL0WP] (8-295)~~

~~w_1 = luma_weight_l1[refIdxL1WP] (8-296)~~

~~o_0 = luma_offset_l0[refIdxL0WP] * (1 << ($BitDepth_Y$ − 8)) (8-297)~~

~~o_1 = luma_offset_l1[refIdxL1WP] * (1 << ($BitDepth_Y$ − 8)) (8-298)~~

~~Otherwise (C in $predPart_C$[x, y] is replaced by Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),~~

~~logWD = chroma_log2_weight_denom (8-299)~~

~~w_0 = chroma_weight_l0[refIdxL0WP][iCbCr] (8-300)~~

~~w_1 = chroma_weight_l1[refIdxL1WP][iCbCr] (8-301)~~

~~o_0 = chroma_offset_l0[refIdxL0WP][iCbCr] * (1 << ($BitDepth_C$ − 8)) (8-302)~~

~~o_1 = chroma_offset_l1[refIdxL1WP][iCbCr] * (1 << ($BitDepth_C$ − 8)) (8-303)~~

~~When explicit mode weighted prediction is used and the partition mbPartIdx\subMbPartIdx has both predFlagL0 and predFlagL1 equal to 1, the following constraint shall be obeyed~~

~~128 <= w_0 + w_1 <= ((logWD − − 7) ? 127 : 128) (8-304)~~

8.5 ~~NOTE – For implicit mode weighted prediction, weights w_0 and w_1 are each guaranteed to be in the range of −64..128 and the constraint expressed in Equation 8-304, although not explicitly imposed, will always be met. For explicit mode weighted prediction with logWD equal to 7, when one of the two weights w_0 or w_1 is inferred to be equal to 128 (as a consequence of luma_weight_l0_flag, luma_weight_l1_flag, chroma_weight_l0_flag, or chroma_weight_l1_flag equal to 0), the other weight (w_1 or w_0) must have a negative value in order for the constraint expressed in Equation 8-304~~Transform coefficient decoding process and picture construction process prior to deblocking filter process

Inputs to this process are Intra16x16DCLevel (if available), Intra16x16ACLevel (if available), LumaLevel (if available), LumaLevel8x8 (if available), ChromaDCLevel (if available), ChromaACLevel (if available), and available Inter or Intra prediction sample arrays for the current macroblock for the applicable components $pred_L$, $pred_{Cb}$, or $pred_{Cr}$.

NOTE 1 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) prediction mode, the luma component of the macroblock prediction array may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated.

Outputs of this process are the constructed sample arrays prior to the deblocking filter process for the applicable components S'_L, S'_{Cb}, or S'_{Cr}.

NOTE 2 – When decoding a macroblock in Intra_4x4 (or Intra_8x8) prediction mode, the luma component of the macroblock constructed sample arrays prior to the deblocking filter process may not be complete, since for each 4x4 (or 8x8) luma block, the Intra_4x4 (or Intra_8x8) prediction process for luma samples as specified in subclause 8.3.1 (or 8.3.2) and the process specified in this subclause are iterated.

带格式的: 英语(英国)
带格式的: 英语(英国)
域代码已更改
带格式的: 英语(英国)
域代码已更改
带格式的: 项目符号和编号
带格式的: 字体: 小五
带格式的: 字体: 小五

This subclause specifies transform coefficient decoding and picture construction prior to the deblocking filter process.

When the current macroblock is coded as P_Skip or B_Skip, all values of LumaLevel, LumaLevel8x8, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

When residual_colour_transform_flag is equal to 1, the residual colour transform process as specified in subclause 8.5.13 is invoked.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.
2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.
3. When residual_colour_transform_flag is equal to 1, the variable $R_{Y,ij}$ is set equal to r_{ij} with i, j = 0..3 and this process is suspended until after completion of the residual colour transform process as specified in subclause 8.5.13, after the completion of which, the variable r_{ij} is set equal to $R_{G,ij}$ with i, j = 0..3 and this process is continued.
4. The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).
5. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \mathrm{Clip1}_Y(\mathrm{pred}_L[xO + j, yO + i] + r_{ij}) \qquad (8\text{-}305)$$

   When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-305 that is not equal to $\mathrm{pred}_L[xO + j, yO + i] + r_{ij}$.
6. The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with luma4x4BlkIdx and u as the inputs.

8.5.2 Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode

When the current macroblock prediction mode is equal to Intra_16x16, the variables Intra16x16DCLevel and Intra16x16ACLevel contain the levels for the luma transform coefficients. The transform coefficient decoding proceeds in the following ordered steps:

1. The 4x4 luma DC transform coefficients of all 4x4 luma blocks of the macroblock are decoded.
   a) The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with Intra16x16DCLevel as the input and the two-dimensional array c as the output.
   b) The scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.8 is invoked with c as the input and dcY as the output.
2. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.
   a) The variable lumaList, which is a list of 16 entries, is derived. The first entry of lumaList is the corresponding value from the array dcY. Figure 8-6 shows the assignment of the indices of the array dcY to the luma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcY_{ij}, and the numbers in large squares refer to luma4x4BlkIdx.

00 0	01 1	02 4	03 5
10 2	11 3	12 6	13 7
20 8	21 9	22 12	23 13
30 10	31 11	32 14	33 15

Figure 8-6 – Assignment of the indices of dcY to luma4x4BlkIdx

The elements in lumaList with index k = 1..15 are specified as

lumaList[k] = Intra16x16ACLevel[luma4x4BlkIdx][k - 1] (8-306)

b) The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with lumaList as the input and the two-dimensional array c as the output.

c) The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

d) When residual_colour_transform_flag is equal to 1, the variable $R_{Y,ij}$ is set equal to r_{ij} with i, j = 0..3 and this process is suspended until after completion of the residual colour transform process as specified in subclause 8.5.13, after the completion of which, the variable r_{ij} is set equal to $R_{G,ij}$ with i, j = 0..3 and this process is continued.

e) The position of the upper-left sample of a 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO).

f) The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

u_{ij} = Clip1$_Y$(pred$_L$[xO + j, yO + i] + r_{ij}) (8-307)

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by 8-307 that is not equal to pred$_L$[xO + j, yO + i] + r_{ij}.

g) The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with luma4x4BlkIdx and u as the inputs.

8.5.3 Specification of transform decoding process for 8x8 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 1.

The variable LumaLevel8x8[luma8x8BlkIdx] with luma8x8BlkIdx = 0..3 contains the levels for the luma transform coefficients for the luma 8x8 block with index luma8x8BlkIdx.

For an 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following ordered steps are specified.

1. The inverse scanning process for 8x8 luma transform coefficients as described in subclause 8.5.6 is invoked with LumaLevel8x8[luma8x8BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 8x8 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

3. When residual_colour_transform_flag is equal to 1, the variable $R_{Y,ij}$ is set equal to r_{ij} with i, j = 0..7 and this process is suspended until after completion of the residual colour transform process as specified in subclause 8.5.13, after the completion of which, the variable r_{ij} is set equal to $R_{G,ij}$ with i, j = 0..7 and this process is continued.

4. The position of the upper-left sample of an 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.4 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO).

5. The 8x8 array u with elements u_{ij} for i, j = 0..7 is derived as

$$u_{ij} = \mathrm{Clip1}_Y(\mathrm{pred}_L[xO + j, yO + i] + r_{ij}) \tag{8-308}$$

When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-308 that is not equal to $\mathrm{pred}_L[xO + j, yO + i] + r_{ij}$.

6. The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with luma8x8BlkIdx and u as the inputs.

8.5.4 Specification of transform decoding process for chroma samples

This process is invoked for each chroma component Cb and Cr seperately.

For each chroma component, the variables ChromaDCLevel[iCbCr] and ChromaACLevel[iCbCr], with iCbCr set equal to 0 for Cb and iCbCr set equal to 1 for Cr, contain the levels for both components of the chroma transform coefficients.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

For each chroma component, the transform decoding proceeds separately in the following ordered steps:

1. The numChroma4x4Blks chroma DC transform coefficients of the 4x4 chroma blocks of the component indexed by iCbCr of the macroblock are decoded.

   a) Depending on the variable chroma_format_idc, the following applies.

      – If chroma_format_idc is equal to 1, the 2x2 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows

$$c = \begin{bmatrix} \mathrm{ChromaDCLevel}[iCbCr][0] & \mathrm{ChromaDCLevel}[iCbCr][1] \\ \mathrm{ChromaDCLevel}[iCbCr][2] & \mathrm{ChromaDCLevel}[iCbCr][3] \end{bmatrix} \tag{8-309}$$

      – Otherwise, if chroma_format_idc is equal to 2, the 2x4 array c is derived using the inverse raster scanning process applied to ChromaDCLevel as follows

$$c = \begin{bmatrix} \mathrm{ChromaDCLevel}[iCbCr][0] & \mathrm{ChromaDCLevel}[iCbCr][2] \\ \mathrm{ChromaDCLevel}[iCbCr][1] & \mathrm{ChromaDCLevel}[iCbCr][5] \\ \mathrm{ChromaDCLevel}[iCbCr][3] & \mathrm{ChromaDCLevel}[iCbCr][6] \\ \mathrm{ChromaDCLevel}[iCbCr][4] & \mathrm{ChromaDCLevel}[iCbCr][7] \end{bmatrix} \tag{8-310}$$

      – Otherwise (chroma_format_idc is equal to 3), the inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with ChromaDCLevel[iCbCr] as the input and the two-dimensional 4x4 array c as the output.

   b) The scaling and transformation process for chroma DC transform coefficients as specified in subclause 8.5.9 is invoked with c as the input and dcC as the output.

2. For each 4x4 chroma block indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks – 1 of the component indexed by iCbCr, the following ordered steps are specified.

   a. The variable chromaList, which is a list of 16 entries, is derived. The first entry of chromaList is the corresponding value from the array dcC. Figure 8-7 shows the assignment of the indices of the array dcC to the chroma4x4BlkIdx. The two numbers in the small squares refer to indices i and j in dcC_{ij}, and the numbers in large squares refer to chroma4x4BlkIdx.

Figure 8-7 – Assignment of the indices of dcC to chroma4x4BlkIdx:
(a) chroma_format_idc equal to 1, (b) chroma_format_idc equal to 2, (c) chroma_format_idc equal to 3

The elements in chromaList with index k = 1..15 are specified as

chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k – 1] (8-311)

b. The inverse scanning process for transform coefficients as specified in subclause 8.5.9 is invoked with chromaList as the input and the two-dimensional array c as the output.

c. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

d. Depending on the variable chroma_format_idc, the position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

– If chroma_format_idc is equal to 1 or 2, the following applies.

xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) (8-312)

yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) (8-313)

– Otherwise (chroma_format_idc is equal to 3), the following applies.

xO = InverseRasterScan(chroma4x4BlkIdx / 4, 8, 8, 16, 0) +
InverseRasterScan(chroma4x4BlkIdx % 4, 4, 4, 8, 0) (8-314)

yO =InverseRasterScan(chroma4x4BlkIdx / 4, 8, 8, 16, 1) +
InverseRasterScan(chroma4x4BlkIdx % 4, 4, 4, 8, 1) (8-315)

e. When residual_colour_transform_flag is equal to 1, the variable xO' is set equal to xO % (4 << transform_size_8x8_flag), the variable yO' is set equal to yO % (4 << transform_size_8x8_flag), and the following applies.

   – If this process is invoked for the chroma component Cb, the variable $R_{Cb,mn}$ is set equal to r_{ij} with i, j = 0..3, m = xO' + i, n = yO' + j, and this process is suspended until after completion of the residual colour transform process as specified in subclause 8.5.13, after which the variable r_{ij} is set equal to $R_{B,mn}$ with i, j = 0..3, m = xO' + i, n = yO' + j and this process is continued.

   – Otherwise (this process is invoked for the chroma component Cr), the variable $R_{Cr,mn}$ is set equal to r_{ij} with i, j = 0..3, m = xO' + i, n = yO' + j and this process is suspended until after the completion of the residual colour transform process as specified in subclause 8.5.13, after which the variable r_{ij} is set equal to $R_{R,mn}$ with i, j = 0..3, m = xO' + i, n = yO' + j and this process is continued.

f. The 4x4 array u with elements u_{ij} for i, j = 0..3 is derived as

$$u_{ij} = \mathrm{Clip1}_C(\mathrm{pred}_C[xO + j, yO + i] + r_{ij}) \qquad (8\text{-}316)$$

   When qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the bitstream shall not contain data that result in a value of u_{ij} as computed by Equation 8-316 that is not equal to $\mathrm{pred}_C[xO + j, yO + i] + r_{ij}$.

g. The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with chroma4x4BlkIdx and u as the inputs.

8.5.5 Inverse scanning process for transform coefficients

Input to this process is a list of 16 values.

Output of this process is a variable c containing a two-dimensional array of 4x4 values. In the case of transform coefficients, these 4x4 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing a 4x4 scaling matrix.

The inverse scanning process for transform coefficients maps the sequence of transform coefficient levels to the transform coefficient level positions. Table 8-13 specifies the two mappings: inverse zig-zag scan and inverse field scan. The inverse zig-zag scan is used for transform coefficients in frame macroblocks and the inverse field scan is used for transform coefficients in field macroblocks.

The inverse scanning process for scaling lists maps the sequence of scaling list entries to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-8 illustrates the scans.

Figure 8-8 – 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative)

Table 8-13 provides the mapping from the index idx of input list of 16 elements to indices i and j of the two-dimensional array c.

Table 8-13 – Specification of mapping of idx to c_{ij} for zig-zag and field scan

idx	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{31}	c_{22}	c_{13}	c_{23}	c_{32}	c_{33}
field	c_{00}	c_{10}	c_{01}	c_{20}	c_{30}	c_{11}	c_{21}	c_{31}	c_{02}	c_{12}	c_{22}	c_{32}	c_{03}	c_{13}	c_{23}	c_{33}

8.5.6 Inverse scanning process for 8x8 luma transform coefficients

Input to this process is a list of 64 values.

Output of this process is a variable c containing a two-dimensional array of 8x8 values. In the case of transform coefficients, these 8x8 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing an 8x8 scaling matrix.

The inverse scanning process for transform coefficients maps the sequence of transform coefficient levels to the transform coefficient level positions. Table 8-14 specifies the two mappings: inverse 8x8 zig-zag scan and inverse 8x8 field scan. The inverse 8x8 zig-zag scan is used for transform coefficients in frame macroblocks and the inverse 8x8 field scan is used for transform coefficients in field macroblocks.

The inverse scanning process for scaling lists maps the sequence of scaling list entries to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-9 illustrates the scans.

Figure 8-9 – 8x8 block scans. (a) 8x8 zig-zag scan. (b) 8x8 field scan (informative)

Table 8-14 provides the mapping from the index idx of the input list of 64 elements to indices i and j of the two-dimensional array c.

Table 8-14 – Specification of mapping of idx to c_{ij} for 8x8 zig-zag and 8x8 field scan

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
zig-zag	c_{00}	c_{01}	c_{10}	c_{20}	c_{11}	c_{02}	c_{03}	c_{12}	c_{21}	c_{30}	c_{40}	c_{31}	c_{22}	c_{13}	c_{04}	c_{05}
field	c_{00}	c_{10}	c_{20}	c_{01}	c_{11}	c_{30}	c_{40}	c_{21}	c_{02}	c_{31}	c_{50}	c_{60}	c_{70}	c_{41}	c_{12}	c_{03}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
zig-zag	c_{14}	c_{23}	c_{32}	c_{41}	c_{50}	c_{60}	c_{51}	c_{42}	c_{33}	c_{24}	c_{15}	c_{06}	c_{07}	c_{16}	c_{25}	c_{34}
field	c_{22}	c_{51}	c_{61}	c_{71}	c_{32}	c_{13}	c_{04}	c_{23}	c_{42}	c_{52}	c_{62}	c_{72}	c_{33}	c_{14}	c_{05}	c_{24}

Table 8-14 (continued) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
zig-zag	c_{43}	c_{52}	c_{61}	c_{70}	c_{71}	c_{62}	c_{53}	c_{44}	c_{35}	c_{26}	c_{17}	c_{27}	c_{36}	c_{45}	c_{54}	c_{63}
field	c_{43}	c_{53}	c_{63}	c_{73}	c_{34}	c_{15}	c_{06}	c_{25}	c_{44}	c_{54}	c_{64}	c_{74}	c_{35}	c_{16}	c_{26}	c_{45}

Table 8-14 (concluded) – Specification of mapping of idx to cij for 8x8 zig-zag and 8x8 field scan

idx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
zig-zag	c_{72}	c_{73}	c_{64}	c_{55}	c_{46}	c_{37}	c_{47}	c_{56}	c_{65}	c_{74}	c_{75}	c_{66}	c_{57}	c_{67}	c_{76}	c_{77}
field	c_{55}	c_{65}	c_{75}	c_{36}	c_{07}	c_{17}	c_{46}	c_{56}	c_{66}	c_{76}	c_{27}	c_{37}	c_{47}	c_{57}	c_{67}	c_{77}

8.5.7 Derivation process for the chroma quantisation parameters and scaling function

Outputs of this process are:

– QP_C: the chroma quantisation parameter for each chroma component Cb and Cr

– QS_C: the additional chroma quantisation parameter for each chroma component Cb and Cr required for decoding SP and SI slices (if applicable)

NOTE 1 – QP quantisation parameter values QP_Y and QS_Y are always in the range of $-QpBdOffset_Y$ to 51, inclusive. QP quantisation parameter values QP_C and QS_C are always in the range of $-QpBdOffset_C$ to 51, inclusive.

The value of QP_C for a chroma component is determined from the current value of QP_Y and the value of chroma_qp_index_offset (for Cb) or second_chroma_qp_index_offset (for Cr).

NOTE 2 – The scaling equations are specified such that the equivalent transform coefficient level scaling factor doubles for every increment of 6 in QP_Y. Thus, there is an increase in the factor used for scaling of approximately 12 % for each increase of 1 in the value of QP_Y.

The value of QP_C for each chroma component is determined as specified in Table 8-15 based on the index denoted as qP_I.

The variable qP_{Offset} for each chroma component is derived as follows.

– If the chroma component is the Cb component, qP_{Offset} is specified as

$$qP_{Offset} = \text{chroma_qp_index_offset} \tag{8-317}$$

– Otherwise (the chroma component is the Cr component), qP_{Offset} is specified as

$$qP_{Offset} = \text{second_chroma_qp_index_offset} \tag{8-318}$$

The value of qP_I for each chroma component is derived as

$$qP_I = \text{Clip3}(\ -\text{QpBdOffset}_C,\ 51,\ QP_Y + qP_{Offset}\ ) \tag{8-319}$$

The value of QP'_C for the chroma components is derived as

$$QP'_C = QP_C + \text{QpBdOffset}_C \tag{8-320}$$

The value of $\text{BitDepth}'_C$ for the chroma components is derived as

$$\text{BitDepth}'_C = \text{BitDepth}_C + \text{residual_colour_transform_flag} \tag{8-321}$$

Table 8-15 – Specification of QP_C as a function of qP_I

qP_I	<30	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
QP_C	$=qP_I$	29	30	31	32	32	33	34	34	35	35	36	36	37	37	37	38	38	38	39	39	39	39

When the current slice is an SP or SI slice, QS_C is derived using the above process, substituting QP_Y with QS_Y and QP_C with QS_C.

The function LevelScale(m, i, j) is specified as follows.

- The 4x4 matrix weightScale(i, j) is specified as follows.
  - The variable mbIsInterFlag is derived as follows.
    - If the current macroblock is coded using Inter macroblock prediction modes, mbIsInterFlag is set equal to 1.
    - Otherwise (the current macroblock is coded using Intra macroblock prediction modes), mbIsInterFlag is set equal to 0.
  - The variable iYCbCr derived as follows.
    - If the input array c relates to a luma residual block, iYCbCr is set equal to 0.
    - Otherwise, if the input array c relates to a chroma residual block and the chroma component is equal to Cb, iYCbCr is set equal to 1.
    - Otherwise (the input array c relates to a chroma residual block and the chroma component is equal to Cr), iYCbCr is set equal to 2.
  - The inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with ScalingList4x4[iYCbCr + ((mbIsInterFlag = = 1) ? 3 : 0)] as the input and the output is assigned to the 4x4 matrix weightScale.

$$\text{LevelScale}(\ m, i, j) = \text{weightScale}(\ i, j) * \text{normAdjust}(\ m, i, j) \tag{8-322}$$

where

$$\text{normAdjust}(m, i, j) = \begin{cases} v_{m0} & \text{for } (\ i\,\%\,2,\ j\,\%\,2\ ) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (\ i\,\%\,2,\ j\,\%\,2\ ) \text{ equal to } (1,1), \\ v_{m2} & \text{otherwise;} \end{cases} \tag{8-323}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{8-324}$$

The function LevelScale8x8(m, i, j) is specified as follows:

– The 8x8 matrix weightScale8x8(i, j) is specified as follows.

  – The variable mbIsInterFlag is derived as follows.

    – If the current macroblock is coded using Inter macroblock prediction modes, mbIsInterFlag is set equal to 1.

    – Otherwise (the current macroblock is coded using Intra macroblock prediction modes), mbIsInterFlag is set equal to 0.

  – The inverse scanning process for 8x8 luma transform coefficients as specified in subclause 8.5.6 is invoked with ScalingList8x8[mbIsInterFlag] as the input and the output is assigned to the 8x8 matrix weightScale8x8.

LevelScale8x8(m, i, j) = weightScale8x8(i, j) * normAdjust8x8(m, i, j) (8-325)

where

$$\text{normAdjust8x8}(m, i, j) = \begin{cases} v_{m0} & \text{for } (i \% 4, j \% 4) \text{ equal to } (0,0), \\ v_{m1} & \text{for } (i \% 2, j \% 2) \text{ equal to } (1,1), \\ v_{m2} & \text{for } (i \% 4, j \% 4) \text{ equal to } (2,2), \\ v_{m3} & \text{for } (i \% 4, j \% 2) \text{ equal to } (0,1) \text{ or } (i \% 2, j \% 4) \text{ equal to } (1,0), \\ v_{m4} & \text{for } (i \% 4, j \% 4) \text{ equal to } (0,2) \text{ or } (i \% 4, j \% 4) \text{ equal to } (2,0), \\ v_{m5} & \text{otherwise;} \end{cases} \tag{8-326}$$

where the first and second subscripts of v are row and column indices, respectively, of the matrix specified as:

$$v = \begin{bmatrix} 20 & 18 & 32 & 19 & 25 & 24 \\ 22 & 19 & 35 & 21 & 28 & 26 \\ 26 & 23 & 42 & 24 & 33 & 31 \\ 28 & 25 & 45 & 26 & 35 & 33 \\ 32 & 28 & 51 & 30 & 40 & 38 \\ 36 & 32 & 58 & 34 & 46 & 43 \end{bmatrix}. \tag{8-327}$$

8.5.8 Scaling and transformation process for luma DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are transform coefficient level values for luma DC transform coefficients of Intra_16x16 macroblocks as a 4x4 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 16 scaled DC values for luma 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output dcY is derived as

$$dcY_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..3 \tag{8-328}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following ~~text of this process specifies the output~~applies.

  – The transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.8.1 is invoked with c as input and the output is assigned to the 4x4 array f of DC values with elements f_{ij}.

  – The scaling process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.8.2 is invoked with f as input and the array dcY of scaled DC values for luma 4x4 blocks of Intra_16x16 macroblocks with elements dcY_{ij} as output.

带格式的: 项目符号和编号

8.5.8.1 Transformation process for luma DC transform coefficients for Intra_16x16 macroblock type

Inputs to this process are transform coefficient level values for luma DC transform coefficients of Intra_16x16 macroblocks as a 4x4 array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are 16 DC values for luma 4x4 blocks of of Intra_16x16 macroblocks as a 4x4 array f with elements f_{ij}.

The inverse transform for the 4x4 luma DC transform coefficients is specified by:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{8-329}$$

带格式的: 项目符号和编号

8.5.8.2 Scaling process for luma DC transform coefficients for Intra_16x16 macroblock type

Outputs of this process are 16 DC values for luma 4x4 blocks of of Intra_16x16 macroblocks as a 4x4 array f with elements f_{ij}.

Outputs of this process are 16 scaled DC values for luma 4x4 blocks of Intra_16x16 macroblocks as a 4x4 array dcY with elements dcY_{ij}.

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

~~After the inverse transform,~~The scaling of luma DC transform coefficients of Intra_16x16 macroblocks is performed as follows.

– If QP'_Y is greater than or equal to 36, the scaled result is derived as

$$dcY_{ij} = (f_{ij} * LevelScale(QP'_Y \% 6, 0, 0)) << (QP'_Y / 6 - 6), \quad \text{with} \quad i, j = 0..3 \tag{8-330}$$

– Otherwise (QP'_Y is less than 36), the scaled result is derived as

$$dcY_{ij} = (f_{ij} * LevelScale (QP'_Y \% 6, 0, 0) + 2^{5-QP'_Y/6}) >> (6 - QP'_Y / 6), \quad \text{with} \quad i, j = 0..3 \tag{8-331}$$

The bitstream shall not contain data that results in any element dcY_{ij} of dcY with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and QP'_Y is less than 10 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c in subclause 8.5.8.1 is not sufficient to represent the full range of values of the elements dcY_{ij} of dcY that could be necessary to form a close approximation of the content of any possible source picture by use of the Intra_16x16 macroblock type.

NOTE 2 – Since the range limit imposed on the elements dcY_{ij} of dcY is imposed after the right shift in Equation 8-331, a larger range of values must be supported in the decoder prior to the right shift.

8.5.9 Scaling and transformation process for chroma DC transform coefficients

Inputs to this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output dcC is derived as

$$dcC_{ij} = c_{ij} \text{ with } i = 0..(\text{MbWidthC} / 4) - 1 \text{ and } j = 0..(\text{MbHeightC} / 4) - 1. \quad (8\text{-}332)$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following ~~text of this process specifies the output~~applies.

  – The transformation process for chroma DC transform coefficients as specified in subclause 8.5.9.1 is invoked with c as input and the output is assigned to the (MbWidthC / 4)x(MbHeightC / 4) array f of DC values with elements f_{ij}.

  – The scaling process for chroma DC coefficients as specified in subclause 8.5.9.2 is invoked with f as input and the (MbWidthC / 4)x(MbHeightC / 4) array dcC of scaled DC values with elements dcC_{ij} as output.

8.5.9.1 Transformation process for chroma DC transform coefficients

Inputs to this process are transform coefficient level values for chroma DC transform coefficients of one chroma component of the macroblock as an (MbWidthC / 4)x(MbHeightC / 4) array c with elements c_{ij}, where i and j form a two-dimensional frequency index.

Outputs of this process are the DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Depending on the variable chroma_format_idc, the inverse transform is specified as follows.

– If chroma_format_idc is equal to 1, the inverse transform for the 2x2 chroma DC transform coefficients is specified as

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}333)$$

– Otherwise, if chroma_format_idc is equal to 2, the inverse transform for the 2x4 chroma DC transform coefficients is specified as

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \\ c_{20} & c_{21} \\ c_{30} & c_{31} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}334)$$

– Otherwise (chroma_format_idc is equal to 3), the inverse transform for the 4x4 chroma DC transform coefficients is specified as follows.

  – If residual_colour_transform_flag is equal to 1 and the current macroblock prediction mode MbPartPredMode(mb_type, 0) is Intra_4x4 or Intra_8x8, the inverse transform for the 4x4 chroma DC transform coefficients is specified as

$$f_{ij} = c_{ij} << 2 \text{ with } i, j = 0..3 \quad (8\text{-}335)$$

  – Otherwise, the inverse transform for the 4x4 chroma DC transform coefficients is specified as

带格式的: 项目符号和编号

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{8-336}$$

带格式的: 项目符号和编号

8.5.9.2 Scaling process for chroma DC transform coefficients

Inputs to this process are DC values as an (MbWidthC / 4)x(MbHeightC / 4) array f with elements f_{ij}.

Outputs of this process are scaled DC values as an (MbWidthC / 4)x(MbHeightC / 4) array dcC with elements dcC_{ij}.

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + BitDepth'_C)}$ to $2^{(7 + BitDepth'_C)}-1$, inclusive.

After the inverse transform, scaling is performed depending on the variable chroma_format_idc as follows.

– If chroma_format_idc is equal to 1, the scaled result is derived as

$$dcC_{ij} = ((f_{ij} * LevelScale(QP'_C \% 6, 0, 0)) << (QP'_C / 6)) >> 5, \quad with \quad i, j = 0, 1 \tag{8-337}$$

– If chroma_format_idc is equal to 2, the following applies.

  – The variable $QP'_{C,DC}$ is derived as

$$QP'_{C,DC} = QP'_C + 3 \tag{8-338}$$

  – Depending on the value of $QP'_{C,DC}$, the following applies.

    – If $QP'_{C,DC}$ is greater than or equal to 36, the scaled result is derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP'_{C,DC} \% 6, 0, 0)) << (QP'_{C,DC} / 6 - 6), \quad \text{with } i = 0..3, j = 0, 1 \tag{8-339}$$

    – Otherwise ($QP'_{C,DC}$ is less than 36), the scaled result is derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP'_{C,DC} \% 6, 0, 0) + 2^{5-QP'_{C,DC}/6}) >> (6 - QP'_{C,DC} / 6), \quad \text{with } i = 0..3, j = 0,1 \tag{8-340}$$

– Otherwise (chroma_format_idc is equal to 3), the following applies.

  – If QP'_C is greater than or equal to 36, the scaled result is derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP'_C \% 6, 0, 0)) << (QP'_C / 6 - 6), \quad \text{with } i, j = 0..3. \tag{8-341}$$

  – Otherwise (QP'_C is less than 36), the scaled result is derived as

$$dcC_{ij} = (f_{ij} * LevelScale(QP'_C \% 6, 0, 0) + 2^{5-QP'_C/6}) >> (6 - QP'_C / 6), \text{ with } i, j, = 0..3 \tag{8-342}$$

The bitstream shall not contain data that results in any element dcC_{ij} of dcC with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + BitDepth'_C)}$ to $2^{(7 + BitDepth'_C)}-1$, inclusive.

NOTE 1 – When entropy_coding_mode_flag is equal to 0 and QP'_C is less than 4 and profile_idc is equal to 66, 77, or 88, the range of values that can be represented for the elements c_{ij} of c in subclause 8.5.9.1 may not be sufficient to represent the full range of values of the elements dcCij of dcC that could be necessary to form a close approximation of the content of any possible source picture.

NOTE 2 – Since the range limit imposed on the elements dcCij of dcC is imposed after the right shift in Equation 8-337, 8-340, or 8-342, a larger range of values must be supported in the decoder prior to the right shift.

8.5.10 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output r is derived as

$$r_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..3 \tag{8-343}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following ~~text of this process specifies the output~~applies.

  – The scaling process for residual 4x4 blocks is invoked with c as input and the output is assigned to the 4x4 array of scaled transform coefficients d with elements d_{ij}.

  – The transformation process for residual 4x4 blocks is invoked with the 4x4 array of scaled transform coefficients d as input and residual sample values as 4x4 array r with elements r_{ij} as output.

带格式的: 项目符号和编号

8.5.10.1 Scaling process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of luma component or an array relating to a residual block of a chroma component.

Output of this process is a 4x4 array of scaled transform coefficients d with elements d_{ij}.

The variable bitDepth is derived as follows.

– If the input array c relates to a luma residual block, bitDepth is set equal to $BitDepth_Y$.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to $BitDepth'_C$.

The bitstream shall not contain data that results in any element c_{ij} of c with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

The variable sMbFlag is derived as follows.

– If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, sMbFlag is set equal to 1,

– Otherwise (mb_type not equal to SI and the macroblock prediction mode is not equal to Inter in an SP slice), sMbFlag is set equal to 0.

The variable qP is derived as follows.

– If the input array c relates to a luma residual block and sMbFlag is equal to 0

$$qP = QP'_Y \tag{8-344}$$

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1

$$qP = QS_Y \tag{8-345}$$

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0

$$qP = QP'_C \tag{8-346}$$

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$$qP = QS_C \tag{8-347}$$

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows.

– If all of the following conditions are true
  – i is equal to 0
  – j is equal to 0
  – c relates to a luma residual block coded using Intra_16x16 prediction mode or c relates to a chroma residual block

the variable d_{00} is derived by

$$d_{00} = c_{00} \tag{8-348}$$

– Otherwise, the following applies.
  – If qP is greater than or equal to 24, the scaled result is derived as follows

$$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) << (qP / 6 - 4), \text{ with } i,j = 0..3 \text{ except as noted above} \tag{8-349}$$

  – Otherwise (qP is less than 24), the scaled result is derived as follows

$$d_{ij} = (c_{ij} * \text{LevelScale} (qP \% 6, i, j) + 2^{3-qP/6}) >> (4 - qP / 6), \text{ with } i, j = 0..3 \text{ except as noted above} \tag{8-350}$$

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

8.5.10.2 Transformation process for residual 4x4 blocks

Input to this process is a 4x4 array of scaled transform coefficients d with elements d_{ij}.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

The variable bitDepth is derived as follows.

– If the input array c relates to a luma residual block, bitDepth is set equal to BitDepth_Y.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to $\text{BitDepth}'_C$.

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})}-1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$$e_{i0} = d_{i0} + d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-351}$$

$$e_{i1} = d_{i0} - d_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-352}$$

$$e_{i2} = (d_{i1} >> 1) - d_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-353}$$

$$e_{i3} = d_{i1} + (d_{i3} >> 1), \quad \text{with} \quad i = 0..3 \tag{8-354}$$

The bitstream shall not contain data that results in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$f_{i0} = e_{i0} + e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-355}$$

带格式的: 项目符号和编号

带格式的: 项目符号和编号

$$f_{i1} = e_{i1} + e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-356}$$

$$f_{i2} = e_{i1} - e_{i2}, \quad \text{with} \quad i = 0..3 \tag{8-357}$$

$$f_{i3} = e_{i0} - e_{i3}, \quad \text{with} \quad i = 0..3 \tag{8-358}$$

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$$g_{0j} = f_{0j} + f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-359}$$

$$g_{1j} = f_{0j} - f_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-360}$$

$$g_{2j} = (f_{1j} >> 1) - f_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-361}$$

$$g_{3j} = f_{1j} + (f_{3j} >> 1), \quad \text{with} \quad j = 0..3 \tag{8-362}$$

The bitstream shall not contain data that results in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$$h_{0j} = g_{0j} + g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-363}$$

$$h_{1j} = g_{1j} + g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-364}$$

$$h_{2j} = g_{1j} - g_{2j}, \quad \text{with} \quad j = 0..3 \tag{8-365}$$

$$h_{3j} = g_{0j} - g_{3j}, \quad \text{with} \quad j = 0..3 \tag{8-366}$$

The bitstream shall not contain data that results in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values is derived as

$$r_{ij} = (h_{ij} + 2^5) >> 6 \quad \text{with} \quad i, j = 0..3 \tag{8-367}$$

8.5.11 Scaling and transformation process for residual 8x8 luma blocks

Input to this process is an 8x8 array c with elements c_{ij} which is an array relating to an 8x8 residual block of the luma component.

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output r is derived as

$$r_{ij} = c_{ij} \quad \text{with} \quad i, j = 0..7 \tag{8-368}$$

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following ~~text of this process specifies the output~~applies.

- The scaling process for residual 8x8 luma blocks is invoked with c as input and the output is assigned to the 8x8 array of scaled transform coefficients d with elements d_{ij}.
- The transformation process for residual 8x8 luma blocks is invoked with the 8x8 array of scaled transform coefficients d as input and residual samples as 8x8 array r with elements r_{ij} as output.

带格式的: 缩进: 左侧: 0.71 厘米, 悬挂缩进: 3.91 字符

带格式的: 项目符号和编号

8.5.11.1 Scaling process for residual 8x8 luma blocks

Input to this process is an 8x8 array c with elements c_{ij} which is either an array relating to an 8x8 residual block of luma component.

Output of this process is an 8x8 array d with elements d_{ij} which is an array relating to an 8x8 block of scaled transform coefficients.

The bitstream shall not contain data that results in any element c_{ij} of c with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

The scaling process for 8x8 block transform coefficient levels c_{ij} proceeds as follows.

– If QP'_Y is greater than or equal to 36, the scaled result is derived as

$$d_{ij} = (c_{ij} * LevelScale8x8(QP'_Y \% 6, i, j)) << (QP'_Y / 6 - 6), \text{ with } i,j = 0..7 \quad (8\text{-}369)$$

– Otherwise (QP'_Y is less than 36), the scaled result is derived as

$$d_{ij} = (c_{ij} * LevelScale8x8(QP'_Y \% 6, i, j)) + 2^{5-QP'_Y/6}) >> (6 - QP'_Y /6), \text{ with } i,j = 0..7 \quad (8\text{-}370)$$

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

带格式的: 项目符号和编号

8.5.11.2 Transformation process for residual 8x8 luma blocks

Input to this process is an 8x8 array d with elements d_{ij} which is an array relating to an 8x8 block of scaled transform coefficients.

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..7 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

– A set of intermediate values e_{ij} is derived by

$$e_{i0} = d_{i0} + d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}371)$$

$$e_{i1} = - d_{i3} + d_{i5} - d_{i7} - (d_{i7} >> 1), \text{ with } i = 0..7 \quad (8\text{-}372)$$

$$e_{i2} = d_{i0} - d_{i4}, \text{ with } i = 0..7 \quad (8\text{-}373)$$

$$e_{i3} = d_{i1} + d_{i7} - d_{i3} - (d_{i3} >> 1), \text{ with } i = 0..7 \quad (8\text{-}374)$$

$$e_{i4} = (d_{i2} >> 1) - d_{i6}, \text{ with } i = 0..7 \quad (8\text{-}375)$$

$e_{i5} = -d_{i1} + d_{i7} + d_{i5} + (d_{i5} >> 1)$, with $i = 0..7$ (8-376)

$e_{i6} = d_{i2} + (d_{i6} >> 1)$, with $i = 0..7$ (8-377)

$e_{i7} = d_{i3} + d_{i5} + d_{i1} + (d_{i1} >> 1)$, with $i = 0..7$ (8-378)

– A second set of intermediate results f_{ij} is computed from the intermediate values e_{ij} as

$f_{i0} = e_{i0} + e_{i6}$, with $i = 0..7$ (8-379)

$f_{i1} = e_{i1} + (e_{i7} >> 2)$, with $i = 0..7$ (8-380)

$f_{i2} = e_{i2} + e_{i4}$, with $i = 0..7$ (8-381)

$f_{i3} = e_{i3} + (e_{i5} >> 2)$, with $i = 0..7$ (8-382)

$f_{i4} = e_{i2} - e_{i4}$, with $i = 0..7$ (8-383)

$f_{i5} = (e_{i3} >> 2) - e_{i5}$, with $i = 0..7$ (8-384)

$f_{i6} = e_{i0} - e_{i6}$, with $i = 0..7$ (8-385)

$f_{i7} = e_{i7} - (e_{i1} >> 2)$, with $i = 0..7$ (8-386)

– Then, the transformed result g_{ij} is computed from these intermediate values f_{ij} as

$g_{i0} = f_{i0} + f_{i7}$, with $i = 0..7$ (8-387)

$g_{i1} = f_{i2} + f_{i5}$, with $i = 0..7$ (8-388)

$g_{i2} = f_{i4} + f_{i3}$, with $i = 0..7$ (8-389)

$g_{i3} = f_{i6} + f_{i1}$, with $i = 0..7$ (8-390)

$g_{i4} = f_{i6} - f_{i1}$, with $i = 0..7$ (8-391)

$g_{i5} = f_{i4} - f_{i3}$, with $i = 0..7$ (8-392)

$g_{i6} = f_{i2} - f_{i5}$, with $i = 0..7$ (8-393)

$g_{i7} = f_{i0} - f_{i7}$, with $i = 0..7$ (8-394)

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

– A set of intermediate values h_{ij} is computed from the horizontally transformed value g_{ij} as

$$h_{0j} = g_{0j} + g_{4j}, \text{ with } j = 0..7 \quad (8\text{-}395)$$

$$h_{1j} = -g_{3j} + g_{5j} - g_{7j} - (g_{7j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}396)$$

$$h_{2j} = g_{0j} - g_{4j}, \text{ with } j = 0..7 \quad (8\text{-}397)$$

$$h_{3j} = g_{1j} + g_{7j} - g_{3j} - (g_{3j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}398)$$

$$h_{4j} = (g_{2j} >> 1) - g_{6j}, \text{ with } j = 0..7 \quad (8\text{-}399)$$

$$h_{5j} = -g_{1j} + g_{7j} + g_{5j} + (g_{5j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}400)$$

$$h_{6j} = g_{2j} + (g_{6j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}401)$$

$$h_{7j} = g_{3j} + g_{5j} + g_{1j} + (g_{1j} >> 1), \text{ with } j = 0..7 \quad (8\text{-}402)$$

– A second set of intermediate results k_{ij} is computed from the intermediate values h_{ij} as

$$k_{0j} = h_{0j} + h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}403)$$

$$k_{1j} = h_{1j} + (h_{7j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}404)$$

$$k_{2j} = h_{2j} + h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}405)$$

$$k_{3j} = h_{3j} + (h_{5j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}406)$$

$$k_{4j} = h_{2j} - h_{4j}, \text{ with } j = 0..7 \quad (8\text{-}407)$$

$$k_{5j} = (h_{3j} >> 2) - h_{5j}, \text{ with } j = 0..7 \quad (8\text{-}408)$$

$$k_{6j} = h_{0j} - h_{6j}, \text{ with } j = 0..7 \quad (8\text{-}409)$$

$$k_{7j} = h_{7j} - (h_{1j} >> 2), \text{ with } j = 0..7 \quad (8\text{-}410)$$

– Then, the transformed result m_{ij} is computed from these intermediate values k_{ij} as

$$m_{0j} = k_{0j} + k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}411)$$

$$m_{1j} = k_{2j} + k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}412)$$

$$m_{2j} = k_{4j} + k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}413)$$

$$m_{3j} = k_{6j} + k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}414)$$

$$m_{4j} = k_{6j} - k_{1j}, \text{ with } j = 0..7 \quad (8\text{-}415)$$

$$m_{5j} = k_{4j} - k_{3j}, \text{ with } j = 0..7 \quad (8\text{-}416)$$

$$m_{6j} = k_{2j} - k_{5j}, \text{ with } j = 0..7 \quad (8\text{-}417)$$

$$m_{7j} = k_{0j} - k_{7j}, \text{ with } j = 0..7 \quad (8\text{-}418)$$

The bitstream shall not contain data that results in any element e_{ij}, f_{ij}, g_{ij}, h_{ij}, or k_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_Y)}$ to $2^{(7 + \text{BitDepth}_Y)} - 1$, inclusive.

The bitstream shall not contain data that results in any element m_{ij} for i and j in the range of 0..7, inclusive, that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_Y)}$ to $2^{(7 + \text{BitDepth}_Y)} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values are derived as

$$r_{ij} = (m_{ij} + 2^5) >> 6 \text{ with } i, j = 0..7 \quad (8\text{-}419)$$

8.5.12 Picture construction process prior to deblocking filter process

Inputs to this process are

– luma4x4BlkIdx or chroma4x4BlkIdx or luma8x8BlkIdx

– a sample array u with elements u_{ij} which is either a 4x4 luma block or a 4x4 chroma block or an 8x8 luma block

The position of the upper-left luma sample of the current macroblock is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with CurrMbAddr as input and the output being assigned to (xP, yP).

When u is a luma block, for each sample u_{ij} of the luma block, the following applies.

– Depending on the size of the block u, the following applies.

  – If u is an 4x4 luma block, the position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 4.

  – Otherwise (u is an 8x8 luma block), the position of the upper-left sample of the 8x8 luma block with index luma8x8BlkIdx inside the macroblock is derived by invoking the inverse 8x8 luma block scanning process in subclause 6.4.4 with luma8x8BlkIdx as the input and the output being assigned to (xO, yO), and the variable nE is set equal to 8.

– Depending on the variable MbaffFrameFlag and the current macroblock, the following applies.

  – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_L[xP + xO + j, yP + 2 * (yO + i)] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \quad (8\text{-}420)$$

  – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_L[xP + xO + j, yP + yO + i] = u_{ij} \quad \text{with } i, j = 0..nE - 1 \quad (8\text{-}421)$$

When u is a chroma block, for each sample u_{ij} of the 4x4 chroma block, the following applies.

– The subscript C in the variable S'_C is replaced with Cb for the Cb chroma component and with Cr for the Cr chroma component.

– Depending on the variable chroma_format_idc, the position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

  – If chroma_format_idc is equal to 1 or 2, the following applies.

$$xO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 0) \quad (8\text{-}422)$$

$$yO = InverseRasterScan(chroma4x4BlkIdx, 4, 4, 8, 1) \quad (8\text{-}423)$$

  – Otherwise (chroma_format_idc is equal to 3), the following applies.

$$xO = InverseRasterScan(chroma4x4BlkIdx / 4, 8, 8, 16, 0) + InverseRasterScan(chroma4x4BlkIdx \% 4, 4, 4, 8, 0) \quad (8\text{-}424)$$

$$yO = InverseRasterScan(chroma4x4BlkIdx / 4, 8, 8, 16, 1) + InverseRasterScan(chroma4x4BlkIdx \% 4, 4, 4, 8, 1) \quad (8\text{-}425)$$

– Depending on the variable MbaffFrameFlag and the current macroblock, the following applies.

  – If MbaffFrameFlag is equal to 1 and the current macroblock is a field macroblock

$$S'_C[(xP / subWidthC) + xO + j, ((yP + SubHeightC – 1) / SubHeightC) + 2 * (yO + i)] = u_{ij} \text{ with } i, j = 0..3 \quad (8\text{-}426)$$

  – Otherwise (MbaffFrameFlag is equal to 0 or the current macroblock is a frame macroblock),

$$S'_C[(xP/ subWidthC) + xO + j, (yP / SubHeightC) + yO + i] = u_{ij} \text{ with } i, j = 0..3 \quad (8\text{-}427)$$

8.5.13 Residual colour transform process

This process is invoked when residual_colour_transform_flag is equal to 1.

After invoking, this process is suspended until the derivation of $R_{Y,ij}$, $R_{Cb,ij}$, and $R_{Cr,ij}$ has been completed for i, j = 0..ijMax, where ijMax is specified as follows.

– If transform_size_8x8_flag is equal to 0, the variable ijMax is set equal to 3.

– Otherwise (transform_size_8x8_flag is equal to 1), the variable ijMax is set equal to 7.

At the resumption of this process, all values $R_{Y,ij}$, $R_{Cb,ij}$, and $R_{Cr,ij}$ with i, j = 0..ijMax shall be available through prior invocations of the relevant processes specified in subclauses 8.5.1, 8.5.2, 8.5.3, or 8.5.4

For each i, j = 0..ijMax, the residual colour transform is computed as

$$t = R_{Y,ij} – (R_{Cb,ij} >> 1) \quad (8\text{-}428)$$

$$R_{G,ij} = t + R_{Cb,ij} \quad (8\text{-}429)$$

$$R_{B,ij} = t – (R_{Cr,ij} >> 1) \quad (8\text{-}430)$$

$$R_{R,ij} = R_{B,ij} + R_{Cr,ij} \quad (8\text{-}431)$$

NOTE – The residual colour transform is similar to the YCgCo transformation specified in Equations E-30 through E-33. However, the residual colour transform operates on the decoded residual difference data within the decoding process rather than

operating as a post-processing step that is outside the decoding process specified in this Recommendation | International Standard.

8.6 Decoding process for P macroblocks in SP slices or SI macroblocks

This process is invoked when decoding P macroblock types in an SP slice type or an SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the predicted samples for the current macroblock.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause specifies the transform coefficient decoding process and picture construction process for P macroblock types in SP slices and SI macroblock type in SI slices.

NOTE – SP slices make use of Inter predictive coding to exploit temporal redundancy in the sequence, in a similar manner to P slice coding. Unlike P slice coding, however, SP slice coding allows identical reconstruction of a slice even when different reference pictures are being used. SI slices make use of spatial prediction, in a similar manner to I slices. SI slice coding allows identical reconstruction to a corresponding SP slice. The properties of SP and SI slices aid in providing functionalities for bitstream switching, splicing, random access, fast-forward, fast reverse, and error resilience/recovery.

An SP slice consists of macroblocks coded either as I macroblock types or P macroblock types.

An SI slice consists of macroblocks coded either as I macroblock types or SI macroblock type.

The transform coefficient decoding process and picture construction process prior to deblocking filter process for I macroblock types in SI slices is invoked as specified in subclause 8.5. SI macroblock type is decoded as described below.

When the current macroblock is coded as P_Skip, all values of LumaLevel, ChromaDCLevel, ChromaACLevel are set equal to 0 for the current macroblock.

8.6.1 SP decoding process for non-switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 0.

Inputs to this process are Inter prediction samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels.

Outputs of this process are the decoded samples of the current macroblock prior to the deblocking filter process.

This subclause applies to all macroblocks in SP slices in which sp_for_switch_flag is equal to 0, except those with macroblock prediction mode equal to Intra_4x4 or Intra_16x16. It does not apply to SI slices.

8.6.1.1 Luma transform coefficient decoding process

Inputs to this process are Inter prediction luma samples for the current macroblock $pred_L$ from subclause 8.4 and the prediction residual transform coefficient levels, LumaLevel, and the index of the 4x4 luma block luma4x4BlkIdx.

The position of the upper-left sample of the 4x4 luma block with index luma4x4BlkIdx inside the current macroblock is derived by invoking the inverse 4x4 luma block scanning process in subclause 6.4.3 with luma4x4BlkIdx as the input and the output being assigned to (x, y).

Let the variable p be a 4x4 array of prediction samples with element p_{ij} being derived as follows.

$$p_{ij} = pred_L[\ x + j,\ y + i\ ] \quad \text{with } i, j = 0..3 \tag{8-432}$$

The variable p is transformed producing transform coefficients c^p according to:

$$c^p = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{8-433}$$

The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements c_{ij}^r as the output.

The prediction residual transform coefficients c^r are scaled using quantisation parameter QP_Y, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 as follows.

$$c_{ij}^s = c_{ij}^p + (((c_{ij}^r * \text{LevelScale}(QP_Y \% 6, i, j) * A_{ij}) << (QP_Y / 6)) >> 10) \quad (8\text{-}434)$$

where LevelScale(m, i, j) is specified in Equation 8-322, and where A_{ij} is specified as:

$$A_{ij} = \begin{cases} 16 & \text{for} \quad (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ 25 & \text{for} \quad (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ 20 & \text{otherwise;} \end{cases} \quad (8\text{-}435)$$

The function LevelScale2(m, i, j), used in the formulas below, is specified as:

$$\text{LevelScale2}(m, i, j) = \begin{cases} w_{m0} & \text{for} \quad (i, j) \in \{(0,0), (0,2), (2,0), (2,2)\}, \\ w_{m1} & \text{for} \quad (i, j) \in \{(1,1), (1,3), (3,1), (3,3)\}, \\ w_{m2} & \text{otherwise;} \end{cases} \quad (8\text{-}436)$$

where the first and second subscripts of w are row and column indices, respectively, of the matrix specified as:

$$w = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \quad (8\text{-}437)$$

The resulting sum, c^s, is quantised with a quantisation parameter QS_Y and with i, j = 0..3 as follows.

$$c_{ij} = \text{Sign}(c_{ij}^s) * ((\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \quad (8\text{-}438)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_Y(r_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}439)$$

The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.1.2 Chroma transform coefficient decoding process

Inputs to this process are Inter prediction chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

– The position of the upper-left sample of a 4x4 chroma block with index chroma4x4BlkIdx inside the macroblock is derived as follows.

$$x = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 0) \quad (8\text{-}440)$$

$$y = \text{InverseRasterScan}(\text{chroma4x4BlkIdx}, 4, 4, 8, 1) \quad (8\text{-}441)$$

– Let p be a 4x4 array of prediction samples with elements p_{ij} being derived as follows.

$$p_{ij} = \text{pred}_C[x + j, y + i] \quad \text{with } i, j = 0..3 \quad (8\text{-}442)$$

– The 4x4 array p is transformed producing transform coefficients c^p(chroma4x4BlkIdx) using Equation 8-433.

– The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

$$\text{chromaList}[k] = \text{ChromaACLevel}[\text{iCbCr}][\text{chroma4x4BlkIdx}][k - 1] \quad (8\text{-}443)$$

– The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with chromaList as the input and the 4x4 array c^r as the output.

– The prediction residual transform coefficients c^r are scaled using quantisation parameter QPC, and added to the transform coefficients of the prediction block c^p with i, j = 0..3 except for the combination i = 0, j = 0 as follows.

$$c_{ij}^s = c_{ij}^p(\text{chroma4x4BlkIdx}) + (((c_{ij}^r * \text{LevelScale}(QP_C \% 6, i, j) * A_{ij}) << (QP_C / 6)) >> 10) \quad (8\text{-}444)$$

– The resulting sum, c^s, is quantised with a quantisation parameter QS_C and with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}(\text{chroma4x4BlkIdx}) = (\text{Sign}(c_{ij}^s) * (\text{Abs}(c_{ij}^s) * \text{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \quad (8\text{-}445)$$

– The scaling and transformation process for residual 4x4 blocks as specified in 8.5.10 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

– The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \text{Clip1}_C(r_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}446)$$

– The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction chroma 4x4 blocks of the current component of the macroblock are assembled into a 2x2 matrix with elements c_{00}^p(chroma4x4BlkIdx) and a 2x2 transform is applied to the DC transform coefficients as follows

$$dc^p = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00}^p(0) & c_{00}^p(1) \\ c_{00}^p(2) & c_{00}^p(3) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (8\text{-}447)$$

The chroma DC prediction residual transform coefficient levels, ChromaDCLevel[iCbCr][k] with k = 0..3 are scaled using quantisation parameter QP, and added to the prediction DC transform coefficients as follows.

$$dc_{ij}^s = dc_{ij}^p + (((\text{ChromaDCLevel}[\text{iCbCr}][j * 2 + i] * \text{LevelScale}(QP_C \% 6, 0, 0) * A_{00}) << (QP_C / 6)) >> 9) \quad \text{with } i, j = 0, 1 \quad (8\text{-}448)$$

The 2x2 array dc^s, is quantised using the quantisation parameter QS_C as follows.

$$dc_{ij}^{\ r} = (Sign(dc_{ij}^{\ s}) * (Abs(dc_{ij}^{\ s}) * LevelScale2(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \qquad (8\text{-}449)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived as follows.

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} dc_{00}^{r} & dc_{01}^{r} \\ dc_{10}^{r} & dc_{11}^{r} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \qquad (8\text{-}450)$$

Scaling of the elements f_{ij} of f is performed as follows.

$$c_{00}(j * 2 + i) = ((f_{ij} * LevelScale(QS_C \% 6, 0, 0)) << (QS_C / 6)) >> 5 \quad \text{with } i, j = 0, 1 \qquad (8\text{-}451)$$

8.6.2 SP and SI slice decoding process for switching pictures

This process is invoked, when decoding P macroblock types in SP slices in which sp_for_switch_flag is equal to 1 and when decoding SI macroblock type in SI slices.

Inputs to this process are the prediction residual transform coefficient levels and the prediction sample arrays $pred_L$, $pred_{Cb}$ and $pred_{Cr}$ for the current macroblock.

8.6.2.1 Luma transform coefficient decoding process

Inputs to this process are prediction luma samples $pred_L$ and the luma prediction residual transform coefficient levels, LumaLevel.

The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.1, is transformed according to Equation 8-433 to produce transform coefficients c^p. These transform coefficients are then quantised with the quantisation parameter QS_Y, as follows:

$$c_{ij}^{\ s} = Sign(c_{ij}^{\ p}) * ((Abs(c_{ij}^{\ p}) * LevelScale2(QS_Y \% 6, i, j) + (1 << (14 + QS_Y / 6))) >> (15 + QS_Y / 6)) \quad \text{with } i, j = 0..3 \qquad (8\text{-}452)$$

The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c^r with elements $c_{ij}^{\ r}$ as the output.

The 4x4 array c with elements c_{ij} with i, j = 0..3 is derived as follows.

$$c_{ij} = c_{ij}^{\ r} + c_{ij}^{\ s} \quad \text{with } i, j = 0..3 \qquad (8\text{-}453)$$

The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = Clip1_Y(r_{ij}) \quad \text{with } i, j = 0..3 \qquad (8\text{-}454)$$

The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with luma4x4BlkIdx and u as the inputs.

8.6.2.2 Chroma transform coefficient decoding process

Inputs to this process are predicted chroma samples for the current macroblock from subclause 8.4 and the prediction residual transform coefficient levels, ChromaDCLevel and ChromaACLevel.

This process is invoked twice: once for the Cb component and once for the Cr component. The component is referred to by replacing C with Cb for the Cb component and C with Cr for the Cr component. Let iCbCr select the current chroma component.

For each 4x4 block of the current chroma component indexed using chroma4x4BlkIdx with chroma4x4BlkIdx equal to 0..3, the following applies.

– The 4x4 array p with elements p_{ij} with i, j = 0..3 is derived as in subclause 8.6.1.2, is transformed according to Equation 8-433 to produce transform coefficients c^p(chroma4x4BlkIdx). These transform coefficients are then quantised with the quantisation parameter QS_C, with i, j = 0..3 except for the combination i = 0, j = 0 as follows. The processing of c_{00}^p(chroma4x4BlkIdx) is described below in this subclause.

$$c_{ij}^s = (\mathrm{Sign}(c_{ij}^p(\mathrm{chroma4x4BlkIdx})) * (\mathrm{Abs}(c_{ij}^p(\mathrm{chroma4x4BlkIdx})) * \mathrm{LevelScale2}(QS_C \% 6, i, j) + (1 << (14 + QS_C / 6)))) >> (15 + QS_C / 6) \quad (8\text{-}455)$$

– The variable chromaList, which is a list of 16 entries, is derived. chromaList[0] is set equal to 0. chromaList[k] with index k = 1..15 are specified as follows.

$$\mathrm{chromaList}[k] = \mathrm{ChromaACLevel}[iCbCr][\mathrm{chroma4x4BlkIdx}][k - 1] \quad (8\text{-}456)$$

– The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with chromaList as the input and the two-dimensional array c^r(chroma4x4BlkIdx) with elements c_{ij}^r(chroma4x4BlkIdx) as the output.

– The 4x4 array c(chroma4x4BlkIdx) with elements c_{ij}(chroma4x4BlkIdx) with i, j = 0..3 except for the combination i = 0, j = 0 is derived as follows. The derivation of c_{00}(chroma4x4BlkIdx) is described below.

$$c_{ij}(\mathrm{chroma4x4BlkIdx}) = c_{ij}^r(\mathrm{chroma4x4BlkIdx}) + c_{ij}^s \quad (8\text{-}457)$$

– The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c(chroma4x4BlkIdx) as the input and r as the output.

– The 4x4 array u with elements u_{ij} is derived as follows.

$$u_{ij} = \mathrm{Clip1}_C(r_{ij}) \text{ with } i, j = 0..3 \quad (8\text{-}458)$$

– The picture construction process prior to deblocking filter process in subclause 8.5.12 is invoked with chroma4x4BlkIdx and u as the inputs.

The derivation of the DC transform coefficient level c_{00}(chroma4x4BlkIdx) is specified as follows. The DC transform coefficients of the 4 prediction 4x4 chroma blocks of the current component of the macroblock, c_{00}^p(chroma4x4BlkIdx), are assembled into a 2x2 matrix, and a 2x2 transform is applied to the DC transform coefficients of these blocks according to Equation 8-447 resulting in DC transform coefficients dc_{ij}^p.

These DC transform coefficients are then quantised with the quantisation parameter QS_C, as given by:

$$dc_{ij}^s = (\mathrm{Sign}(dc_{ij}^p) * (\mathrm{Abs}(dc_{ij}^p) * \mathrm{LevelScale2}(QS_C \% 6, 0, 0) + (1 << (15 + QS_C / 6)))) >> (16 + QS_C / 6) \quad \text{with } i, j = 0, 1 \quad (8\text{-}459)$$

The parsed chroma DC prediction residual transform coefficients, ChromaDCLevel[iCbCr][k] with k = 0..3 are added to these quantised DC transform coefficients of the prediction block, as given by:

$$dc_{ij}^r = dc_{ij}^s + \mathrm{ChromaDCLevel}[iCbCr][j * 2 + i] \quad \text{with } i, j = 0, 1 \quad (8\text{-}460)$$

The 2x2 array f with elements f_{ij} and i, j = 0..1 is derived using Equation 8-450.

The 2x2 array f with elements f_{ij} and i, j = 0..1 is copied as follows.

$$c_{00}(j * 2 + i) = f_{ij} \quad \text{with } i, j = 0, 1 \quad (8\text{-}461)$$

8.7 Deblocking filter process

A conditional filtering process is applied to all NxN (where N = 4 or N = 8 for luma, and N = 4 for chroma) block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc, as specified below. This filtering process is performed on a macroblock basis after the completion of the picture construction process prior to deblocking filter process (as specified in subclauses 8.5 and 8.6) for the entire decoded picture, with all macroblocks in a picture processed in order of increasing macroblock addresses.

NOTE 1 – Prior to the operation of the deblocking filter process for each macroblock, the deblocked samples of the macroblock or macroblock pair above (if any) and the macroblock or macroblock pair to the left (if any) of the current macroblock are always available because the deblocking filter process is performed after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture. However, for purposes of determining which edges are to be filtered when disable_deblocking_filter_idc is equal to 2, macroblocks in different slices are considered not available during specified steps of the operation of the deblocking filter process.

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock and each component, vertical edges are filtered first, starting with the edge on the left-hand side of the macroblock proceeding through the edges towards the right-hand side of the macroblock in their geometrical order, and then horizontal edges are filtered, starting with the edge on the top of the macroblock proceeding through the edges towards the bottom of the macroblock in their geometrical order. Figure 8-10 shows edges of a macroblock which can be interpreted as luma or chroma edges.

When interpreting the edges in Figure 8-10 as luma edges, depending on the transform_size_8x8_flag, the following applies.

– If transform_size_8x8_flag is equal to 0, both types, the solid bold and dashed bold luma edges are filtered.

– Otherwise (transform_size_8x8_flag is equal to 1), only the solid bold luma edges are filtered.

When interpreting the edges in Figure 8-10 as chroma edges, depending on chroma_format_idc, the following applies.

– If chroma_format_idc is equal to 1 (4:2:0 format), only the solid bold chroma edges are filtered.

– Otherwise, if chroma_format_idc is equal to 2 (4:2:2 format), the solid bold vertical chroma edges are filtered and both types, the solid bold and dashed bold horizontal chroma edges are filtered.

– Otherwise, if chroma_format_idc is equal to 3 (4:4:4 format), both types, the solid bold and dashed bold chroma edges are filtered.

– Otherwise (chroma_format_idc is equal to 0 (monochrome)), no chroma edges are filtered.

Figure 8-10 – Boundaries in a macroblock to be filtered

For the current macroblock address CurrMbAddr proceeding over values 0..PicSizeInMbs – 1, the following applies.

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.
2. The variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as follows.
   - The variable fieldModeMbFlag is derived as follows.
     - If any of the following conditions is true, fieldModeMbFlag is set equal to 1.
       - field_pic_flag is equal to 1
       - MbaffFrameFlag is equal to 1 and the macroblock CurrMbAddr is a field macroblock
     - Otherwise, fieldModeMbFlag is set equal to 0.
   - The variable filterInternalEdgesFlag is derived as follows.
     - If disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1, the variable filterInternalEdgesFlag is set equal to 0;
     - Otherwise (disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is not equal to 1), the variable filterInternalEdgesFlag is set equal to 1.
   - The variable filterLeftMbEdgeFlag is derived as follows.
     - If any of the following conditions is true, the variable filterLeftMbEdgeFlag is set equal to 0.
       - MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0.
       - MbaffFrameFlag is equal to 1 and (CurrMbAddr >> 1) % PicWidthInMbs is equal to 0
       - disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1
       - disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrA is not available
     - Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.
   - The variable filterTopMbEdgeFlag is derived as follows.
     - If any of the following conditions is true, the variable filterTopMbEdgeFlag is set equal to 0.
       - MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs.
       - MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, and the macroblock CurrMbAddr is a field macroblock.

- – MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and CurrMbAddr % 2 is equal to 0.
- – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1
- – disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrB is not available

– Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

3. Given the variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows.

– When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical luma edge is specified as follows.

- – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (0, k)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.

– When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified as follows.

- – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (4, k)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.
- – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (8, k)$ with k = 0..15, and S'_L as input and S'_L as output.
- – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (12, k)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.

– When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows.

- – If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr - 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.
  - – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, ~~and~~ $(xE_k, yE_k) = (k, 0)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.
  - – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, ~~and~~ $(xE_k, yE_k) = (k, 1)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.
- – Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 0)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.

– When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified as follows.

- – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 4)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.
- – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 8)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.
- – When transform_size_8x8_flag is equal to 0, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 12)$ with k = 0..15, and S'_L as input and a modified version of S'_L as output.

– For the filtering of both chroma components with iCbCr = 0 for Cb and iCbCr = 1 for Cr, the following applies.

  – Let S'_C be equal to S'_{Cb} when iCbCr = 0 and S'_{Cb} when iCbCr = 1

  – When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical chroma edge is specified as follows.

    – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (0, k)$ with k = 0..MbHeightC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

  – When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified as follows.

    – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (4, k)$ with k = 0..MbHeightC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

    – When chroma_format_idc is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (8, k)$ with k = 0..MbHeightC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

    – When chroma_format_idc is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (12, k)$ with k = 0..MbHeightC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

  – When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows.

    – If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr – 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.

      – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, ~~and~~ $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

      – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, ~~and~~ $(xE_k, yE_k) = (k, 1)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

    – Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

  – When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified as follows.

    – The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 4)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

    – When chroma_format_idc is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 8)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

带格式的: 项目符号和编号

– When chroma_format_idc is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, iCbCr, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, ~~and~~ $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC - 1, and S'_C as input and a modified version of S'_C ~~with C being replaced by Cb for iCbCr = 0 and C being replaced by Cr for iCbCr = 1~~ as output.

NOTE 2 – When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 3 – For example, in 4:2:0 chroma format when transform_size_8x8_flag is equal to 0, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

Finally, the arrays S'_L, S'_{Cb}, S'_{Cr} are assigned to the arrays S_L, S_{Cb}, S_{Cr} (which represent the decoded picture), respectively.

8.7.1 Filtering process for block edges

Inputs to this process are

– ~~-~~a variable chromaEdgeFlag~~, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1),~~

– a variable verticalEdgeFlag~~,~~

– a variable fieldModeFilteringFlag~~, and~~

– a set of nE sample locations (xE_k, yE_k), with k = 0..nE - 1, expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xE_k, yE_k) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0)

– a sample array s'~~.~~

Output of this process is a modified version of the sample array s'.

The variable nE is derived as follows.

– If chromaEdgeFlag is equal to 0, nE is set equal to 16.

– Otherwise (chromaEdgeFlag is equal to 1), nE is set equal to (verticalEdgeFlag = = 1) ? MbHeightC : MbWidthC.

~~Let s' be a variable specifying a luma or chroma sample array, be derived as follows.~~

~~- If chromaEdgeFlag is equal to 0, s' represents the luma sample array S'_L of the current picture.~~

~~- Otherwise, if chromaEdgeFlag is equal to 1 and iCbCr is equal to 0, s' represents the chroma sample array S'_{Cb} of the chroma component Cb of the current picture.~~

~~- Otherwise (chromaEdgeFlag is equal to 1 and iCbCr is equal to 1), s' represents the chroma sample array S'_{Cr} of the chroma component Cr of the current picture.~~

The variable dy is derived as follows.

– If fieldModeFilteringFlag is equal to 1 and MbaffFrameFlag is equal to 1, dy is set equal to 2.

– Otherwise (fieldModeFilteringFlag is equal to 0 or MbaffFrameFlag is equal to 0), dy is set equal to 1.

The position of the upper-left luma sample of the macroblock CurrMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr = CurrMbAddr as input and the output being assigned to (xI, yI).

The variables xP and yP are derived as follows.

– If chromaEdgeFlag is equal to 0, xP is set equal to xI and yP is set equal to yI.

带格式的: 项目符号和编号

带格式的: 项目符号和编号

– Otherwise (chromaEdgeFlag is equal to 1), xP is set equal to xI / SubWidthC and yP is set equal to (yI + SubHeightC – 1) / SubHeightC.

p3	p2	p1	p0	q0	q1	q2	q3

Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

For each sample location (xE_k, yE_k), k = 0 .. nE - 1, the following applies.

– The filtering process is applied to a 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 Figure 8-11 with the edge lying between p_0 and q_0. p_i and q_i with i = 0..3 are specified as follows.

  – If verticalEdgeFlag is equal to 1,

$$q_i = s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] \quad (8\text{-}462)$$

$$p_i = s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] \quad (8\text{-}463)$$

  – Otherwise (verticalEdgeFlag is equal to 0),

$$q_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}464)$$

$$p_i = s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (yE_k \% 2\ )\ ] \quad (8\text{-}465)$$

– The process specified in subclause 8.7.2 is invoked with the sample values p_i and q_i (i = 0..3), chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag as input, and the output is assigned to the filtered result sample values p'_i and q'_i with i = 0..2.

– The input sample values p_i and q_i with i = 0..2 are replaced by the corresponding filtered result sample values p'_i and q'_i with i = 0..2 inside the sample array s' as follows.

  – If verticalEdgeFlag is equal to 1,

$$s'[\ xP + xE_k + i,\ yP + dy * yE_k\ ] = q'_i \quad (8\text{-}466)$$

$$s'[\ xP + xE_k - i - 1,\ yP + dy * yE_k\ ] = p'_i \quad (8\text{-}467)$$

  – Otherwise (verticalEdgeFlag is equal to 0),

$$s'[\ xP + xE_k,\ yP + dy * (\ yE_k + i\ ) - (\ yE_k \% 2\ )\ ] = q'_i \quad (8\text{-}468)$$

$$s'[\ xP + xE_k,\ yP + dy * (\ yE_k - i - 1\ ) - (\ yE_k \% 2\ )\ ] = p'_i \quad (8\text{-}469)$$

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values p_i and q_i with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag.

Outputs of this process are the filtered result sample values p'_i and q'_i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows.

– If chromaEdgeFlag is equal to 0, the derivation process for the content dependent boundary filtering strength specified in subclause 8.7.2.1 is invoked with p_0, q_0, and verticalEdgeFlag as input, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (SubWidthC * x, SubHeightC * y) inside the luma array of the same field, where (x, y) is the location of the chroma sample q_0 inside the chroma array for that field.

The process specified in subclause 8.7.2.2 is invoked with p_0, q_0, p_1, q_1, chromaEdgeFlag, and bS as input, and the output is assigned to filterSamplesFlag, indexA, α, and β.

Depending on the variable filterSamplesFlag, the following applies.

– If filterSamplesFlag is equal to 1, the following applies.

  – If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with p_i and q_i (i = 0..2), chromaEdgeFlag, bS, β, and indexA given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

  – Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with p_i and q_i (i = 0..3), chromaEdgeFlag, α, and β given as input, and the output is assigned to p'_i and q'_i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'_i and q'_i (i = 0..2) are replaced by the corresponding input samples p_i and q_i:

$$\text{for } i = 0..2, \qquad p'_i = p_i \tag{8-470}$$

$$\text{for } i = 0..2, \qquad q'_i = q_i \tag{8-471}$$

8.7.2.1 Derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

Let the variable mixedModeEdgeFlag be derived as follows.

– If MbaffFrameFlag is equal to 1 and the samples p_0 and q_0 are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair, mixedModeEdgeFlag is set equal to 1

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows.

– If the block edge is also a macroblock edge and any of the following conditions are true, a value of bS equal to 4 is the output:

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode

  – the samples p_0 and q_0 are both in frame macroblocks and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode

  – MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– Otherwise, if any of the following conditions are true, a value of bS equal to 3 is the output:

  – mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode

- mixedModeEdgeFlag is equal to 0 and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI
- mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock coded using an Intra macroblock prediction mode
- mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p_0 or q_0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– Otherwise, if the following condition is true, a value of bS equal to 2 is the output:

- the luma block containing sample p_0 or the luma block containing sample q_0 contains non-zero transform coefficient levels

– Otherwise, if any of the following conditions are true, a value of bS equal to 1 is the output:

- mixedModeEdgeFlag is equal to 1
- mixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock/sub-macroblock partition containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the macroblock/sub-macroblock partition containing the sample q_0.

  NOTE 1 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether or not the index position within a reference picture list is different or not.

- mixedModeEdgeFlag is equal to 0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample p_0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples.
- mixedModeEdgeFlag is equal to 0 and two motion vectors and two different reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same two reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two macroblock/sub-macroblock partitions for the same reference picture is greater than or equal to 4 in units of quarter luma frame samples.
- mixedModeEdgeFlag is equal to 0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample p_0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample q_0 and both of the following conditions are true:
  - The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in units of quarter luma frame samples.
  - The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p_0 and list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q_0 is greater than or equal to 4 in units of quarter luma frame samples.

    NOTE 2 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

– Otherwise, a value of bS equal to 0 is the output.

8.7.2.2 Derivation process for the thresholds for each block edge

Inputs to this process are the input sample values p_0, q_0, p_1 and q_1 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, and bS, for the set of input samples, as specified in 8.7.2.

Outputs of this process are the variable filterSamplesFlag, which indicates whether the input samples are filtered, the value of indexA, and the values of the threshold variables α and β.

Let qP_p and qP_q be variables specifying quantisation parameter values for the macroblocks containing the samples p_0 and q_0, respectively. The variables qP_z (with z being replaced by p or q) are derived as follows.

– If chromaEdgeFlag is equal to 0, the following applies.

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set to 0.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set to the value of QP_Y of the macroblock containing the sample z_0.

– Otherwise (chromaEdgeFlag is equal to 1), the following applies.

  – If the macroblock containing the sample z_0 is an I_PCM macroblock, qP_z is set to the value of QP_C that corresponds to a value of 0 for QP_Y as specified in subclause 8.5.7.

  – Otherwise (the macroblock containing the sample z_0 is not an I_PCM macroblock), qP_z is set to the value of QP_C that corresponds to the value QP_Y of the macroblock containing the sample z_0 as specified in subclause 8.5.7.

Let qP_{av} be a variable specifying an average quantisation parameter. It is derived as follows.

$$qP_{av} = (qP_p + qP_q + 1) >> 1 \quad (8\text{-}472)$$

NOTE – In SP and SI slices, qP_{av} is derived in the same way as in other slice types. QS_Y from Equation 7-28 is not used in the deblocking filter.

Let indexA be a variable that is used to access the α table (Table 8-16) as well as the t_{C0} table (Table 8-17), which is used in filtering of edges with bS less than 4 as specified in subclause 8.7.2.3, and let indexB be a variable that is used to access the β table (Table 8-16). The variables indexA and indexB are derived as follows, where the values of FilterOffsetA and FilterOffsetB are the values of those variables specified in subclause 7.4.3 for the slice that contains the macroblock containing sample q_0.

$$indexA = Clip3(0, 51, qP_{av} + FilterOffsetA) \quad (8\text{-}473)$$

$$indexB = Clip3(0, 51, qP_{av} + FilterOffsetB) \quad (8\text{-}474)$$

The variables α' and β' depending on the values of indexA and indexB are specified in Table 8-16. Depending on chromaEdgeFlag, the corresponding threshold variables α and β are derived as follows.

– If chromaEdgeFlag is equal to 0,

$$\alpha = \alpha' * (1 << (BitDepth_Y - 8)) \quad (8\text{-}475)$$

$$\beta = \beta' * (1 << (BitDepth_Y - 8)) \quad (8\text{-}476)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$\alpha = \alpha' * (1 << (BitDepth_C - 8)) \quad (8\text{-}477)$$

$$\beta = \beta' * (1 << (BitDepth_C - 8)) \quad (8\text{-}478)$$

The variable filterSamplesFlag is derived by

$$filterSamplesFlag = (bS != 0 \ \&\& \ Abs(p_0 - q_0) < \alpha \ \&\& \ Abs(p_1 - p_0) < \beta \ \&\& \ Abs(q_1 - q_0) < \beta) \quad (8\text{-}479)$$

Table 8-16 – Derivation of offset dependent threshold variables α' and β' from indexA and indexB

	indexA (for α') or indexB (for β')																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
α'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	5	6	7	8	9	10	12	13
β'	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	3	3	3	3	4	4	4

Table 8-16 (concluded) – Derivation of indexA and indexB from offset dependent threshold variables α' and β'

	indexA (for α') or indexB (for β')																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
α'	15	17	20	22	25	28	32	36	40	45	50	56	63	71	80	90	101	113	127	144	162	182	203	226	255	255
β'	6	6	7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15	16	16	17	17	18	18

8.7.2.3 Filtering process for edges with bS less than 4

Inputs to this process are the input sample values p_i and q_i (i = 0..2) of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, bS, β, and indexA, for the set of input samples, as specified in 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i (i = 0..2) for the set of input sample values.

The filtered result samples p'_0 and q'_0 are derived by

$$\Delta = \text{Clip3}(-t_C, t_C, ((((q_0 - p_0) << 2) + (p_1 - q_1) + 4) >> 3)) \quad (8\text{-}480)$$

$$p'_0 = \text{Clip1}(p_0 + \Delta) \quad (8\text{-}481)$$

$$q'_0 = \text{Clip1}(q_0 - \Delta) \quad (8\text{-}482)$$

where the threshold t_C is determined as follows.

– If chromaEdgeFlag is equal to 0,

$$t_C = t_{C0} + ((a_p < \beta) ? 1 : 0) + ((a_q < \beta) ? 1 : 0) \quad (8\text{-}483)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$t_C = t_{C0} + 1 \quad (8\text{-}484)$$

Depending on the values of indexA and bS the variable t'_{C0} is specified in Table 8-17. Depending on chromaEdgeFlag, the corresponding threshold variable t_{C0} is derived as follows.

– If chromaEdgeFlag is equal to 0,

$$t_{C0} = t'_{C0} * (1 << (\text{BitDepth}_Y - 8)) \quad (8\text{-}485)$$

– Otherwise (chromaEdgeFlag is equal to 1),

$$t_{C0} = t'_{C0} * (1 << (\text{BitDepth}_C - 8)) \quad (8\text{-}486)$$

Let a_p and a_q be two threshold variables specified by

$$a_p = \mathrm{Abs}(p_2 - p_0) \tag{8-487}$$

$$a_q = \mathrm{Abs}(q_2 - q_0) \tag{8-488}$$

The filtered result sample p'_1 is derived as follows

– If chromaEdgeFlag is equal to 0 and a_p is less than β,

$$p'_1 = p_1 + \mathrm{Clip3}(-t_{C0}, t_{C0}, (p_2 + ((p_0 + q_0 + 1) >> 1) - (p_1 << 1)) >> 1) \tag{8-489}$$

– Otherwise (chromaEdgeFlag is equal to 1 or a_p is greater than or equal to β),

$$p'_1 = p_1 \tag{8-490}$$

The filtered result sample q'_1 is derived as follows

– If chromaEdgeFlag is equal to 0 and a_q is less than β,

$$q'_1 = q_1 + \mathrm{Clip3}(-t_{C0}, t_{C0}, (q_2 + ((p_0 + q_0 + 1) >> 1) - (q_1 << 1)) >> 1) \tag{8-491}$$

– Otherwise (chromaEdgeFlag is equal to 1 or a_q is greater than or equal to β),

$$q'_1 = q_1 \tag{8-492}$$

The filtered result samples p'_2 and q'_2 are always set equal to the input samples p_2 and q_2:

$$p'_2 = p_2 \tag{8-493}$$

$$q'_2 = q_2 \tag{8-494}$$

Table 8-17 – Value of variable t'_{C0} as a function of indexA and bS

	indexA																									
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
bS = 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
bS = 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
bS = 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Table 8-17 (concluded) – Value of variable t'_{C0} as a function of indexA and bS

	indexA																									
	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
bS = 1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13
bS = 2	1	1	1	1	1	2	2	2	2	3	3	3	4	4	5	5	6	7	8	8	10	11	12	13	15	17
bS = 3	1	2	2	2	2	3	3	3	4	4	4	5	6	6	7	8	9	10	11	13	14	16	18	20	23	25

8.7.2.4 Filtering process for edges for bS equal to 4

Inputs to this process are the input sample values p_i and q_i (i = 0..3) of a single set of samples across an edge that is to be filtered, the variable chromaEdgeFlag, and the values of the threshold variables α and β for the set of samples, as specified in subclause 8.7.2.

Outputs of this process are the filtered result sample values p'_i and q'_i (i = 0..2) for the set of input sample values.

Let a_p and a_q be two threshold variables as specified in Equations 8-487 and 8-488, respectively, in subclause 8.7.2.3.

The filtered result samples p'_i (i = 0..2) are derived as follows.

– If chromaEdgeFlag is equal to 0 and the following condition holds,

$$a_p < \beta \ \&\& \ \mathrm{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}495)$$

then the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) >> 3 \quad (8\text{-}496)$$

$$p'_1 = (p_2 + p_1 + p_0 + q_0 + 2) >> 2 \quad (8\text{-}497)$$

$$p'_2 = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) >> 3 \quad (8\text{-}498)$$

– Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-495 does not hold), the variables p'_0, p'_1, and p'_2 are derived by

$$p'_0 = (2*p_1 + p_0 + q_1 + 2) >> 2 \quad (8\text{-}499)$$

$$p'_1 = p_1 \quad (8\text{-}500)$$

$$p'_2 = p_2 \quad (8\text{-}501)$$

The filtered result samples q'_i (i = 0..2) are derived as follows.

– If chromaEdgeFlag is equal to 0 and the following condition holds,

$$a_q < \beta \ \&\& \ \mathrm{Abs}(p_0 - q_0) < ((\alpha >> 2) + 2) \quad (8\text{-}502)$$

then the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) >> 3 \quad (8\text{-}503)$$

$$q'_1 = (p_0 + q_0 + q_1 + q_2 + 2) >> 2 \quad (8\text{-}504)$$

$$q'_2 = (2*q_3 + 3*q_2 + q_1 + q_0 + p_0 + 4) >> 3 \quad (8\text{-}505)$$

– Otherwise (chromaEdgeFlag is equal to 1 or the condition in Equation 8-502 does not hold), the variables q'_0, q'_1, and q'_2 are derived by

$$q'_0 = (2*q_1 + q_0 + p_1 + 2) >> 2 \quad (8\text{-}506)$$

$$q'_1 = q_1 \quad (8\text{-}507)$$

$$q'_2 = q_2 \tag{8-508}$$

9 Parsing process

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), te(v) (see subclause 9.1), ce(v) (see subclause 9.2), or ae(v) (see subclause 9.3).

9.1 Parsing process for Exp-Golomb codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ue(v), me(v), se(v), or te(v). For syntax elements in subclauses 7.3.4 and 7.3.5, this process is invoked only when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v), me(v), or se(v) are Exp-Golomb-coded. Syntax elements coded as te(v) are truncated Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = -1;
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read_bits(leadingZeroBits)}$$

where the value returned from read_bits(leadingZeroBits) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9-1 illustrates the structure of the Exp-Golomb code by separating the bit string into “prefix” and “suffix” bits. The “prefix” bits are those bits that are parsed in the above pseudo-code for the computation of leadingZeroBits, and are shown as either 0 or 1 in the bit string column of Table 9-1. The “suffix” bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 9-1, with i being in the range 0 to leadingZeroBits - 1, inclusive. Each x_i can take on values 0 or 1.

Table 9-1 – Bit strings with “prefix” and “suffix” bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
1	0
0 1 x_0	1-2
0 0 1 x_1 x_0	3-6
0 0 0 1 x_2 x_1 x_0	7-14
0 0 0 0 1 x_3 x_2 x_1 x_0	15-30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31-62
…	…

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 0 1 0	9
…	…

Depending on the descriptor, the value of a syntax element is derived as follows.

– If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

– Otherwise, if the syntax element is coded as se(v), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 9.1.1 with codeNum as the input.

– Otherwise, if the syntax element is coded as me(v), the value of the syntax element is derived by invoking the mapping process for coded block pattern as specified in subclause 9.1.2 with codeNum as the input.

– Otherwise (the syntax element is coded as te(v)), the range of possible values for the syntax element is determined first. The range of this syntax element may be between 0 and x, with x being greater than or equal to 1 and the range is used in the derivation of the value of the syntax element value as follows

  – If x is greater than 1, codeNum and the value of the syntax element is derived in the same way as for syntax elements coded as ue(v)

  – Otherwise (x is equal to 1), the parsing process for codeNum which is equal to the value of the syntax element is given by a process equivalent to:

```
b = read_bits( 1 )
codeNum = !b
```

9.1.1 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9-3 provides the assignment rule.

Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

codeNum	syntax element value
0	0
1	1
2	–1
3	2
4	–2
5	3
6	–3
k	$(-1)^{k+1}$ Ceil(k ÷2)

9.1.2 Mapping process for coded block pattern

Input to this process is codeNum as specified in subclause 9.1.

Output of this process is a value of the syntax element coded_block_pattern coded as me(v).

Table 9-4 shows the assignment of coded_block_pattern to codeNum depending on whether the macroblock prediction mode is equal to Intra_4x4, Intra_8x8 or Inter.

Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes

(a) chroma_format_idc is not equal to 0

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	47	0
1	31	16
2	15	1
3	0	2
4	23	4
5	27	8
6	29	32
7	30	3
8	7	5
9	11	10
10	13	12
11	14	15
12	39	47
13	43	7

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
14	45	11
15	46	13
16	16	14
17	3	6
18	5	9
19	10	31
20	12	35
21	19	37
22	21	42
23	26	44
24	28	33
25	35	34
26	37	36
27	42	40
28	44	39
29	1	43
30	2	45
31	4	46
32	8	17
33	17	18
34	18	20
35	20	24
36	24	19
37	6	21
38	9	26
39	22	28
40	25	23
41	32	27
42	33	29
43	34	30
44	36	22
45	40	25

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
46	38	38
47	41	41

(b) chroma_format_idc is equal to 0

codeNum	coded_block_pattern	
	Intra_4x4, Intra_8x8	Inter
0	15	0
1	0	1
2	7	2
3	11	4
4	13	8
5	14	3
6	3	5
7	5	10
8	10	12
9	12	15
10	1	7
11	2	11
12	4	13
13	8	14
14	6	6
15	9	9

9.2 CAVLC parsing process for transform coefficient levels

This process is invoked when parsing syntax elements with descriptor equal to ce(v) in subclause 7.3.5.3.1 and when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx of the current block of transform coefficient levels.

Output of this process is the list coeffLevel containing transform coefficient levels of the luma block with block index luma4x4BlkIdx or the chroma block with block index chroma4x4BlkIdx.

The process is specified in the following ordered steps:

1. All transform coefficient levels, with indices from 0 to maxNumCoeff - 1, in the list coeffLevel are set equal to 0.

2. The total number of non-zero transform coefficient levels TotalCoeff(coeff_token) and the number of trailing one transform coefficient levels TrailingOnes(coeff_token) are derived by parsing coeff_token (see subclause 9.2.1) as follows.
   - If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to 0, the list coeffLevel containing 0 values is returned and no further step is carried out.
   - Otherwise, the following steps are carried out.
     a. The non-zero transform coefficient levels are derived by parsing trailing_ones_sign_flag, level_prefix, and level_suffix (see subclause 9.2.2).
     b. The runs of zero transform coefficient levels before each non-zero transform coefficient level are derived by parsing total_zeros and run_before (see subclause 9.2.3).
     c. The level and run information are combined into the list coeffLevel (see subclause 9.2.4).

9.2.1 Parsing process for total number of transform coefficient levels and trailing ones

Inputs to this process are bits from slice data, a maximum number of non-zero transform coefficient levels maxNumCoeff, the luma block index luma4x4BlkIdx or the chroma block index chroma4x4BlkIdx of the current block of transform.

Outputs of this process are TotalCoeff(coeff_token) and TrailingOnes(coeff_token).

The syntax element coeff_token is decoded using one of the six VLCs specified in the six right-most columns of Table 9-5. Each VLC specifies both TotalCoeff(coeff_token) and TrailingOnes(coeff_token) for a given codeword coeff_token. VLC selection is dependent upon a variable nC that is derived as follows.

- If the CAVLC parsing process is invoked for ChromaDCLevel, nC is derived as follows.
  - If chroma_format_idc is equal to 1, nC is set equal to -1,
  - Otherwise, if chroma_format_idc is equal to 2, nC is set equal to -2,
  - Otherwise (chroma_format_idc is equal to 3), nC is set equal to 0.
- Otherwise, the following applies.
  - When the CAVLC parsing process is invoked for Intra16x16DCLevel, luma4x4BlkIdx is set equal to 0.
  - The variables blkA and blkB are derived as follows.
    - If the CAVLC parsing process is invoked for Intra16x16DCLevel, Intra16x16ACLevel, or LumaLevel, the process specified in subclause 6.4.8.3 is invoked with luma4x4BlkIdx as the input, and the output is assigned to mbAddrA, mbAddrB, luma4x4BlkIdxA, and luma4x4BlkIdxB. The 4x4 luma block specified by mbAddrA\luma4x4BlkIdxA is assigned to blkA, and the 4x4 luma block specified by mbAddrB\luma4x4BlkIdxB is assigned to blkB.
    - Otherwise (the CAVLC parsing process is invoked for ChromaACLevel), the process specified in subclause 6.4.8.4 is invoked with chroma4x4BlkIdx as input, and the output is assigned to mbAddrA, mbAddrB, chroma4x4BlkIdxA, and chroma4x4BlkIdxB. The 4x4 chroma block specified by mbAddrA\iCbCr\chroma4x4BlkIdxA is assigned to blkA, and the 4x4 chroma block specified by mbAddrB\iCbCr\chroma4x4BlkIdxB is assigned to blkB.
  - Let nA and nB be the number of non-zero transform coefficient levels (given by TotalCoeff(coeff_token)) in the block of transform coefficient levels blkA located to the left of the current block and the block of transform coefficient levels blkB located above the current block, respectively.
  - With N replaced by A and B, in mbAddrN, blkN, and nN the following applies.
    - If any of the following conditions is true, nN is set equal to 0.
      - mbAddrN is not available
      - The current macroblock is coded using an Intra prediction mode, constrained_intra_pred_flag is equal to 1 and mbAddrN is coded using Inter prediction and slice data partitioning is in use (nal_unit_type is in the range of 2 to 4, inclusive).

- The macroblock mbAddrN has mb_type equal to P_Skip or B_Skip
- All AC residual transform coefficient levels of the neighbouring block blkN are equal to 0 due to the corresponding bit of CodedBlockPatternLuma or CodedBlockPatternChroma being equal to 0

– Otherwise, if mbAddrN is an I_PCM macroblock, nN is set equal to 16.

– Otherwise, nN is set equal to the value TotalCoeff(coeff_token) of the neighbouring block blkN.

NOTE 1 – The values nA and nB that are derived using TotalCoeff(coeff_token) do not include the DC transform coefficient levels in Intra_16x16 macroblocks or DC transform coefficient levels in chroma blocks, because these transform coefficient levels are decoded separately. When the block above or to the left belongs to an Intra_16x16 macroblock, or is a chroma block, nA and nB is the number of decoded non-zero AC transform coefficient levels.

NOTE 2 – When parsing for Intra16x16DCLevel, the values nA and nB are based on the number of non-zero transform coefficient levels in adjacent 4x4 blocks and not on the number of non-zero DC transform coefficient levels in adjacent 16x16 blocks.

– Given the values of nA and nB, the variable nC is derived as follows.

  – If both mbAddrA and mbAddrB are available, the variable nC is set equal to (nA + nB + 1) >> 1.

  – Otherwise (mbAddrA is not available or mbAddrB is not available), the variable nC is set equal to nA + nB.

The value of TotalCoeff(coeff_token) resulting from decoding coeff_token shall be in the range of 0 to maxNumCoeff, inclusive.

Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token)

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
0	0	1	11	1111	0000 11	01	1
0	1	0001 01	0010 11	0011 11	0000 00	0001 11	0001 111
1	1	01	10	1110	0000 01	1	01
0	2	0000 0111	0001 11	0010 11	0001 00	0001 00	0001 110
1	2	0001 00	0011 1	0111 1	0001 01	0001 10	0001 101
2	2	001	011	1101	0001 10	001	001
0	3	0000 0011 1	0000 111	0010 00	0010 00	0000 11	0000 0011 1
1	3	0000 0110	0010 10	0110 0	0010 01	0000 011	0001 100
2	3	0000 101	0010 01	0111 0	0010 10	0000 010	0001 011
3	3	0001 1	0101	1100	0010 11	0001 01	0000 1
0	4	0000 0001 11	0000 0111	0001 111	0011 00	0000 10	0000 0011 0
1	4	0000 0011 0	0001 10	0101 0	0011 01	0000 0011	0000 0010 1
2	4	0000 0101	0001 01	0101 1	0011 10	0000 0010	0001 010
3	4	0000 11	0100	1011	0011 11	0000 000	0000 01
0	5	0000 0000 111	0000 0100	0001 011	0100 00	-	0000 0001 11
1	5	0000 0001 10	0000 110	0100 0	0100 01	-	0000 0001 10
2	5	0000 0010 1	0000 101	0100 1	0100 10	-	0000 0010 0
3	5	0000 100	0011 0	1010	0100 11	-	0001 001

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
0	6	0000 0000 0111 1	0000 0011 1	0001 001	0101 00	-	0000 0000 111
1	6	0000 0000 110	0000 0110	0011 10	0101 01	-	0000 0000 110
2	6	0000 0001 01	0000 0101	0011 01	0101 10	-	0000 0001 01
3	6	0000 0100	0010 00	1001	0101 11	-	0001 000
0	7	0000 0000 0101 1	0000 0001 111	0001 000	0110 00	-	0000 0000 0111
1	7	0000 0000 0111 0	0000 0011 0	0010 10	0110 01	-	0000 0000 0110
2	7	0000 0000 101	0000 0010 1	0010 01	0110 10	-	0000 0000 101
3	7	0000 0010 0	0001 00	1000	0110 11	-	0000 0001 00
0	8	0000 0000 0100 0	0000 0001 011	0000 1111	0111 00	-	0000 0000 0011 1
1	8	0000 0000 0101 0	0000 0001 110	0001 110	0111 01	-	0000 0000 0101
2	8	0000 0000 0110 1	0000 0001 101	0001 101	0111 10	-	0000 0000 0100
3	8	0000 0001 00	0000 100	0110 1	0111 11	-	0000 0000 100
0	9	0000 0000 0011 11	0000 0000 1111	0000 1011	1000 00	-	
1	9	0000 0000 0011 10	0000 0001 010	0000 1110	1000 01	-	

TrailingOnes (coeff_token)	TotalCoeff (coeff_token)	0 <= nC < 2	2 <= nC < 4	4 <= nC < 8	8 <= nC	nC = = -1	nC = = -2
2	9	0000 0000 0100 1	0000 0001 001	0001 010	1000 10	-	
3	9	0000 0000 100	0000 0010 0	0011 00	1000 11	-	
0	10	0000 0000 0010 11	0000 0000 1011	0000 0111 1	1001 00	-	
1	10	0000 0000 0010 10	0000 0000 1110	0000 1010	1001 01	-	
2	10	0000 0000 0011 01	0000 0000 1101	0000 1101	1001 10	-	
3	10	0000 0000 0110 0	0000 0001 100	0001 100	1001 11	-	
0	11	0000 0000 0001 111	0000 0000 1000	0000 0101 1	1010 00	-	
1	11	0000 0000 0001 110	0000 0000 1010	0000 0111 0	1010 01	-	
2	11	0000 0000 0010 01	0000 0000 1001	0000 1001	1010 10	-	
3	11	0000 0000 0011 00	0000 0001 000	0000 1100	1010 11	-	
0	12	0000 0000 0001 011	0000 0000 0111 1	0000 0100 0	1011 00	-	
1	12	0000 0000 0001 010	0000 0000 0111 0	0000 0101 0	1011 01	-	
2	12	0000 0000 0001 101	0000 0000 0110 1	0000 0110 1	1011 10	-	
3	12	0000 0000 0010 00	0000 0000 1100	0000 1000	1011 11	-	
0	13	0000 0000 0000 1111	0000 0000 0101 1	0000 0011 01	1100 00	-	
1	13	0000 0000 0000 001	0000 0000 0101 0	0000 0011 1	1100 01	-	
2	13	0000 0000 0001 001	0000 0000 0100 1	0000 0100 1	1100 10	-	
3	13	0000 0000 0001 100	0000 0000 0110 0	0000 0110 0	1100 11	-	
0	14	0000 0000 0000 1011	0000 0000 0011 1	0000 0010 01	1101 00	-	
1	14	0000 0000 0000 1110	0000 0000 0010 11	0000 0011 00	1101 01	-	
2	14	0000 0000 0000 1101	0000 0000 0011 0	0000 0010 11	1101 10	-	
3	14	0000 0000 0001 000	0000 0000 0100 0	0000 0010 10	1101 11	-	
0	15	0000 0000 0000 0111	0000 0000 0010 01	0000 0001 01	1110 00	-	
1	15	0000 0000 0000 1010	0000 0000 0010 00	0000 0010 00	1110 01	-	
2	15	0000 0000 0000 1001	0000 0000 0010 10	0000 0001 11	1110 10	-	
3	15	0000 0000 0000 1100	0000 0000 0000 1	0000 0001 10	1110 11	-	
0	16	0000 0000 0000 0100	0000 0000 0001 11	0000 0000 01	1111 00	-	
1	16	0000 0000 0000 0110	0000 0000 0001 10	0000 0001 00	1111 01	-	
2	16	0000 0000 0000 0101	0000 0000 0001 01	0000 0000 11	1111 10	-	
3	16	0000 0000 0000 1000	0000 0000 0001 00	0000 0000 10	1111 11	-	

9.2.2 Parsing process for level information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the number of trailing one transform coefficient levels TrailingOnes(coeff_token).

Output of this process is a list with name level containing transform coefficient levels.

Initially an index i is set equal to 0. Then the following procedure is iteratively applied TrailingOnes(coeff_token) times to decode the trailing one transform coefficient levels (if any):

- A 1-bit syntax element trailing_ones_sign_flag is decoded and evaluated as follows.
  - If trailing_ones_sign_flag is equal to 0, the value +1 is assigned to level[i].
  - Otherwise (trailing_ones_sign_flag is equal to 1), the value -1 is assigned to level[i].
- The index i is incremented by 1.

Following the decoding of the trailing one transform coefficient levels, a variable suffixLength is initialised as follows.

- If TotalCoeff(coeff_token) is greater than 10 and TrailingOnes(coeff_token) is less than 3, suffixLength is set equal to 1.
- Otherwise (TotalCoeff(coeff_token) is less than or equal to 10 or TrailingOnes(coeff_token) is equal to 3), suffixLength is set equal to 0.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – TrailingOnes(coeff_token)) times to decode the remaining levels (if any):

- The syntax element level_prefix is decoded as specified in subclause 9.2.2.1.
- The variable levelSuffixSize is set equal to the variable suffixLength with the exception of the following two cases.
- When level_prefix is equal to 14 and suffixLength is equal to 0, levelSuffixSize is set equal to 4.
- When level_prefix is greater than or equal to 15, levelSuffixSize is set equal to level_prefix - 3.
- The syntax element level_suffix is decoded as follows.
  - If levelSuffixSize is greater than 0, the syntax element level_suffix is decoded as unsigned integer representation u(v) with levelSuffixSize bits.
  - Otherwise (levelSuffixSize is equal to 0), the syntax element level_suffix is inferred to be equal to 0.
- A variable levelCode is set equal to (Min(15, level_prefix) << suffixLength) + level_suffix.
- When level_prefix is greater than or equal to 15 and suffixLength is equal to 0, levelCode is incremented by 15.
- When level_prefix is greater than or equal to 16, levelCode is incremented by (1<<(level_prefix – 3)) – 4096.
- When the index i is equal to TrailingOnes(coeff_token) and TrailingOnes(coeff_token) is less than 3, levelCode is incremented by 2.
- The variable level[i] is derived as follows.
  - If levelCode is an even number, the value (levelCode + 2) >> 1 is assigned to level[i].
  - Otherwise (levelCode is an odd number), the value (-levelCode – 1) >> 1 is assigned to level[i].
- When suffixLength is equal to 0, suffixLength is set equal to 1.
- When the absolute value of level[i] is greater than (3 << (suffixLength – 1)) and suffixLength is less than 6, suffixLength is incremented by 1.
- The index i is incremented by 1.

9.2.2.1 Parsing process for level_prefix

Inputs to this process are bits from slice data.

Output of this process is level_prefix.

The parsing process for this syntax element consists in reading the bits starting at the current location in the bitstream up to and including the first non-zero bit, and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = -1
for( b = 0; !b; leadingZeroBits++ )
    b = read_bits( 1 )
level_prefix = leadingZeroBits
```

Table 9-6 illustrates the codeword table for level_prefix.

Table 9-6 – Codeword table for level_prefix (informative)

level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
7	0000 0001
8	0000 0000 1
9	0000 0000 01
10	0000 0000 001
11	0000 0000 0001
12	0000 0000 0000 1
13	0000 0000 0000 01
14	0000 0000 0000 001
15	0000 0000 0000 0001
…	…

9.2.3 Parsing process for run information

Inputs to this process are bits from slice data, the number of non-zero transform coefficient levels TotalCoeff(coeff_token), and the maximum number of non-zero transform coefficient levels maxNumCoeff.

Output of this process is a list of runs of zero transform coefficient levels preceding non-zero transform coefficient levels called run.

Initially, an index i is set equal to 0.

The variable zerosLeft is derived as follows.

– If the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is equal to the maximum number of non-zero transform coefficient levels maxNumCoeff, a variable zerosLeft is set equal to 0.

– Otherwise (the number of non-zero transform coefficient levels TotalCoeff(coeff_token) is less than the maximum number of non-zero transform coefficient levels maxNumCoeff), total_zeros is decoded and zerosLeft is set equal to its value.

The VLC used to decode total_zeros is derived as follows:

If maxNumCoeff is equal to 4, one of the VLCs specified in Table 9-9 (a) is used.

– Otherwise, if maxNumCoeff is equal to 8, one of the VLCs specified in Table 9-9 (b) is used.

– Otherwise (maxNumCoeff is not equal to 4 and not equal to 8), VLCs from Table 9-7 and 9-8 are used.

The following procedure is then applied iteratively (TotalCoeff(coeff_token) – 1) times:

– The variable run[i] is derived as follows.

  – If zerosLeft is greater than zero, a value run_before is decoded based on Table 9-10 and zerosLeft. run[i] is set equal to run_before.

  – Otherwise (zerosLeft is equal to 0), run[i] is set equal to 0.

– The value of run[i] is subtracted from zerosLeft and the result assigned to zerosLeft. The result of the subtraction shall be greater than or equal to 0.

– The index i is incremented by 1.

Finally the value of zerosLeft is assigned to run[i].

Table 9-7 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 1 to 7

total_zeros	**TotalCoeff(coeff_token)**						
	1	2	3	4	5	6	7
0	1	111	0101	0001 1	0101	0000 01	0000 01
1	011	110	111	111	0100	0000 1	0000 1
2	010	101	110	0101	0011	111	101
3	0011	100	101	0100	111	110	100
4	0010	011	0100	110	110	101	011
5	0001 1	0101	0011	101	101	100	11
6	0001 0	0100	100	100	100	011	010
7	0000 11	0011	011	0011	011	010	0001
8	0000 10	0010	0010	011	0010	0001	001
9	0000 011	0001 1	0001 1	0010	0000 1	001	0000 00
10	0000 010	0001 0	0001 0	0001 0	0001	0000 00	
11	0000 0011	0000 11	0000 01	0000 1	0000 0		
12	0000 0010	0000 10	0000 1	0000 0			
13	0000 0001 1	0000 01	0000 00				
14	0000 0001 0	0000 00					
15	0000 0000 1						

Table 9-8 – total_zeros tables for 4x4 blocks with TotalCoeff(coeff_token) 8 to 15

total_zeros	TotalCoeff(coeff_token)							
	8	9	10	11	12	13	14	15
0	0000 01	0000 01	0000 1	0000	0000	000	00	0
1	0001	0000 00	0000 0	0001	0001	001	01	1
2	0000 1	0001	001	001	01	1	1	
3	011	11	11	010	1	01		
4	11	10	10	1	001			
5	10	001	01	011				
6	010	01	0001					
7	001	0000 1						
8	0000 00							

Table 9-9 – total_zeros tables for chroma DC 2x2 and 2x4 blocks

(a) Chroma DC 2x2 block (4:2:0 chroma sampling)

total_zeros	TotalCoeff(coeff_token)		
	1	2	3
0	1	1	1
1	01	01	0
2	001	00	
3	000		

(b) Chroma DC 2x4 block (4:2:2 chroma sampling)

total_zeros	TotalCoeff(coeff_token)						
	1	2	3	4	5	6	7
0	1	000	000	110	00	00	0
1	010	01	001	00	01	01	1
2	011	001	01	01	10	1	
3	0010	100	10	10	11		
4	0011	101	110	111			
5	0001	110	111				
6	0000 1	111					
7	0000 0						

Table 9-10 – Tables for run_before

run_before	zerosLeft						
	1	2	3	4	5	6	>6
0	1	1	11	11	11	11	111
1	0	01	10	10	10	000	110
2	-	00	01	01	011	001	101
3	-	-	00	001	010	011	100
4	-	-	-	000	001	010	011
5	-	-	-	-	000	101	010
6	-	-	-	-	-	100	001
7	-	-	-	-	-	-	0001
8		-	-	-	-	-	00001
9	-	-	-	-	-	-	000001
10	-	-	-	-	-	-	0000001
11	-	-	-	-	-	-	00000001
12	-	-	-	-	-	-	000000001
13	-	-	-	-	-	-	0000000001
14	-	-	-	-	-	-	00000000001

9.2.4 Combining level and run information

Input to this process are a list of transform coefficient levels called level, a list of runs called run, and the number of non-zero transform coefficient levels TotalCoeff(coeff_token).

Output of this process is an list coeffLevel of transform coefficient levels.

A variable coeffNum is set equal to -1 and an index i is set equal to (TotalCoeff(coeff_token) – 1). The following procedure is iteratively applied TotalCoeff(coeff_token) times:

– coeffNum is incremented by run[i] + 1.

– coeffLevel[coeffNum] is set equal to level[i].

– The index i is decremented by 1.

9.3 CABAC parsing process for slice data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.4 and 7.3.5 when entropy_coding_mode_flag is equal to 1.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause 7.3.4, the initialisation process of the CABAC parsing process is invoked as specified in subclause 9.3.1.

The parsing of syntax elements proceeds as follows:

For each requested value of a syntax element a binarization is derived as described in subclause 9.3.2.

The binarization for the syntax element and the sequence of parsed bins determines the decoding process flow as described in subclause 9.3.3.

For each bin of the binarization of the syntax element, which is indexed by the variable binIdx, a context index ctxIdx is derived as specified in subclause 9.3.3.1.

For each ctxIdx the arithmetic decoding process is invoked as specified in subclause 9.3.3.2.

The resulting sequence (b_0 .. b_{binIdx}) of parsed bins is compared to the set of bin strings given by the binarization process after decoding of each bin. When the sequence matches a bin string in the given set, the corresponding value is assigned to the syntax element.

In case the request for a value of a syntax element is processed for the syntax element mb_type and the decoded value of mb_type is equal to I_PCM, the decoding engine is initialised after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2.

The whole CABAC parsing process is illustrated in the flowchart of Figure 9-1 with the abbreviation SE for syntax element.

Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)

9.3.1 Initialisation process

Outputs of this process are initialised CABAC internal variables.

The processes in subclauses 9.3.1.1 and 9.3.1.2 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

The process in subclause 9.3.1.2 is also invoked after decoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Table 9-12 to Table 9-23 contain the values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 and 7.3.5 except for the end-of-slice flag.

For each context variable, the two variables pStateIdx and valMPS are initialised.

NOTE 1 – The variable pStateIdx corresponds to a probability state index and the variable valMPS corresponds to the value of the most probable symbol as further described in subclause 9.3.3.2.

The two values assigned to pStateIdx and valMPS for the initialisation are derived from $SliceQP_Y$, which is derived in Equation 7-27. Given the two table entries (m, n),

1. preCtxState = Clip3(1, 126, ((m ∗ Clip3(0, 51, $SliceQP_Y$)) >> 4) + n)
2. if(preCtxState <= 63) {

    pStateIdx = 63 - preCtxState

    valMPS = 0

   } else {

    pStateIdx = preCtxState - 64

    valMPS = 1

   }

In Table 9-11, the ctxIdx for which initialisation is needed for each of the slice types are listed. Also listed is the table number that includes the values of m and n needed for the initialisation. For P, SP, and B slice type, the initialisation depends also on the value of the cabac_init_idc syntax element. Note that the syntax element names do not affect the initialisation process.

NOTE – The slice type EI, EP and EB are specified in Annex G

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I, EI	P, SP, EP	B, EB
slice_data( )	mb_skip_flag	Table 9-13 Table 9-14			11-13	24-26
	mb_field_decoding_flag	Table 9-18	70-72	70-72	70-72	70-72
macroblock_layer( )	mb_type	Table 9-12 Table 9-13 Table 9-14	0-10	3-10	14-20	27-35
	transform_size_8x8_flag	Table 9-16	na	399-401	399-401	399-401
	coded_block_pattern (luma)	Table 9-18	73-76	73-76	73-76	73-76
	coded_block_pattern (chroma)	Table 9-18	77-84	77-84	77-84	77-84
	mb_qp_delta	Table 9-17	60-63	60-63	60-63	60-63

Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process

	Syntax element	Table	Slice type			
			SI	I, EI	P, SP, EP	B, EB
mb_pred( )	prev_intra4x4_pred_mode_flag	Table 9-17	68	68	68	68
	rem_intra4x4_pred_mode	Table 9-17	69	69	69	69
	prev_intra8x8_pred_mode_flag	Table 9-17	na	68	68	68
	rem_intra8x8_pred_mode	Table 9-17	na	69	69	69
	intra_chroma_pred_mode	Table 9-17	64-67	64-67	64-67	64-67
mb_pred( ) and sub_mb_pred( )	ref_idx_l0	Table 9-16			54-59	54-59
	ref_idx_l1	Table 9-16				54-59
	mvd_l0[ ][ ][0]	Table 9-15			40-46	40-46
	mvd_l1[ ][ ][0]	Table 9-15				40-46
	mvd_l0[ ][ ][1]	Table 9-15			47-53	47-53
	mvd_l1[ ][ ][1]	Table 9-15				47-53
sub_mb_pred( )	sub_mb_type	Table 9-13 Table 9-14			21-23	36-39
residual_block_cabac( )	coded_block_flag	Table 9-18	85-104	85-104	85-104	85-104
	significant_coeff_flag[ ]	Table 9-19 Table 9-22 Table 9-24 Table 9-24	105-165 277-337	105-165 277-337 402-416 436-450	105-165 277-337 402-416 436-450	105-165 277-337 402-416 436-450
	last_significant_coeff_flag[ ]	Table 9-20 Table 9-23 Table 9-24 Table 9-24	166-226 338-398	166-226 338-398 417-425 451-459	166-226 338-398 417-425 451-459	166-226 338-398 417-425 451-459
	coeff_abs_level_minus1[ ]	Table 9-21 Table 9-24	227-275	227-275 426-435	227-275 426-435	227-275 426-435

NOTE 2 – ctxIdx equal to 276 is associated with the end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdx equal to 276. This decoding process, however, may also be implemented by using the decoding process specified in subclause 9.3.3.2.1. In this case, the initial values associated with ctxIdx equal to 276 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10

Initialisation variables	ctxIdx										
	0	1	2	3	4	5	6	7	8	9	10
m	20	2	3	20	2	3	-28	-23	-6	-1	7
n	-15	54	74	-15	54	74	127	104	53	54	51

Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23

Value of cabac_init_idc	Initialisation variables	ctxIdx												
		11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
0	**m**	23	23	21	1	0	-37	5	-13	-11	1	12	-4	17
	n	33	2	0	9	49	118	57	78	65	62	49	73	50
1	**m**	22	34	16	-2	4	-29	2	-6	-13	5	9	-3	10
	n	25	0	0	9	41	118	65	71	79	52	50	70	54
2	**m**	29	25	14	-10	-3	-27	26	-4	-24	5	6	-17	14
	n	16	0	0	51	62	99	16	85	102	57	57	73	57

Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39

Value of cabac_init_idc	Initialisation variables	ctxIdx															
		24	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
0	**m**	18	9	29	26	16	9	-46	-20	1	-13	-11	1	-6	-17	-6	9
	n	64	43	0	67	90	104	127	104	67	78	65	62	86	95	61	45
1	**m**	26	19	40	57	41	26	-45	-15	-4	-6	-13	5	6	-13	0	8
	n	34	22	0	2	36	69	127	101	76	71	79	52	69	90	52	43
2	**m**	20	20	29	54	37	12	-32	-22	-2	-4	-24	5	-6	-14	-6	4
	n	40	10	0	0	42	97	127	117	74	85	102	57	93	88	44	55

Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53

Value of cabac_init_idc	Initialisation variables	ctxIdx													
		40	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**
0	**m**	-3	-6	-11	6	7	-5	2	0	-3	-10	5	4	-3	0
	n	69	81	96	55	67	86	88	58	76	94	54	69	81	88
1	**m**	-2	-5	-10	2	2	-3	-3	1	-3	-6	0	-3	-7	-5
	n	69	82	96	59	75	87	100	56	74	85	59	81	86	95
2	**m**	-11	-15	-21	19	20	4	6	1	-5	-13	5	6	-3	-1
	n	89	103	116	57	58	84	96	63	85	106	63	75	90	101

Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59, and 399 to 401

Value of cabac_init_idc	Initialisation variables	ctxIdx								
		54	**55**	**56**	**57**	**58**	**59**	**399**	**400**	**401**
I slices	**m**	na	na	na	na	na	na	31	31	25
	n	na	na	na	na	na	na	21	31	50
0	**m**	-7	-5	-4	-5	-7	1	12	11	14
	n	67	74	74	80	72	58	40	51	59
1	**m**	-1	-1	1	-2	-5	0	25	21	21
	n	66	77	70	86	72	61	32	49	54
2	**m**	3	-4	-2	-12	-7	1	21	19	17
	n	55	79	75	97	50	60	33	50	61

Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69

Initialisation variables	ctxIdx									
	60	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**
m	0	0	0	0	-9	4	0	-7	13	3
n	41	63	63	63	83	86	97	72	41	62

Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
70	0	11	0	45	13	15	7	34	**88**	-11	115	-13	108	-4	92	5	78
71	1	55	-4	78	7	51	-9	88	**89**	-12	63	-3	46	0	39	-6	55
72	0	69	-3	96	2	80	-20	127	**90**	-2	68	-1	65	0	65	4	61
73	-17	127	-27	126	-39	127	-36	127	**91**	-15	84	-1	57	-15	84	-14	83
74	-13	102	-28	98	-18	91	-17	91	**92**	-13	104	-9	93	-35	127	-37	127
75	0	82	-25	101	-17	96	-14	95	**93**	-3	70	-3	74	-2	73	-5	79
76	-7	74	-23	67	-26	81	-25	84	**94**	-8	93	-9	92	-12	104	-11	104
77	-21	107	-28	82	-35	98	-25	86	**95**	-10	90	-8	87	-9	91	-11	91
78	-27	127	-20	94	-24	102	-12	89	**96**	-30	127	-23	126	-31	127	-30	127
79	-31	127	-16	83	-23	97	-17	91	**97**	-1	74	5	54	3	55	0	65
80	-24	127	-22	110	-27	119	-31	127	**98**	-6	97	6	60	7	56	-2	79
81	-18	95	-21	91	-24	99	-14	76	**99**	-7	91	6	59	7	55	0	72
82	-27	127	-18	102	-21	110	-18	103	**100**	-20	127	6	69	8	61	-4	92
83	-21	114	-13	93	-18	102	-13	90	**101**	-4	56	-1	48	-3	53	-6	56
84	-30	127	-29	127	-36	127	-37	127	**102**	-5	82	0	68	0	68	3	68
85	-17	123	-7	92	0	80	11	80	**103**	-7	76	-4	69	-7	74	-8	71
86	-12	115	-5	89	-5	89	5	76	**104**	-22	125	-8	88	-9	88	-13	98
87	-16	122	-7	96	-7	94	2	84									

Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
105	-7	93	-2	85	-13	103	-4	86	**136**	-13	101	5	53	0	58	-5	75
106	-11	87	-6	78	-13	91	-12	88	**137**	-13	91	-2	61	-1	60	-8	80
107	-3	77	-1	75	-9	89	-5	82	**138**	-12	94	0	56	-3	61	-21	83
108	-5	71	-7	77	-14	92	-3	72	**139**	-10	88	0	56	-8	67	-21	64
109	-4	63	2	54	-8	76	-4	67	**140**	-16	84	-13	63	-25	84	-13	31
110	-4	68	5	50	-12	87	-8	72	**141**	-10	86	-5	60	-14	74	-25	64
111	-12	84	-3	68	-23	110	-16	89	**142**	-7	83	-1	62	-5	65	-29	94
112	-7	62	1	50	-24	105	-9	69	**143**	-13	87	4	57	5	52	9	75
113	-7	65	6	42	-10	78	-1	59	**144**	-19	94	-6	69	2	57	17	63
114	8	61	-4	81	-20	112	5	66	**145**	1	70	4	57	0	61	-8	74
115	5	56	1	63	-17	99	4	57	**146**	0	72	14	39	-9	69	-5	35
116	-2	66	-4	70	-78	127	-4	71	**147**	-5	74	4	51	-11	70	-2	27
117	1	64	0	67	-70	127	-2	71	**148**	18	59	13	68	18	55	13	91
118	0	61	2	57	-50	127	2	58	**149**	-8	102	3	64	-4	71	3	65
119	-2	78	-2	76	-46	127	-1	74	**150**	-15	100	1	61	0	58	-7	69
120	1	50	11	35	-4	66	-4	44	**151**	0	95	9	63	7	61	8	77
121	7	52	4	64	-5	78	-1	69	**152**	-4	75	7	50	9	41	-10	66
122	10	35	1	61	-4	71	0	62	**153**	2	72	16	39	18	25	3	62
123	0	44	11	35	-8	72	-7	51	**154**	-11	75	5	44	9	32	-3	68
124	11	38	18	25	2	59	-4	47	**155**	-3	71	4	52	5	43	-20	81
125	1	45	12	24	-1	55	-6	42	**156**	15	46	11	48	9	47	0	30
126	0	46	13	29	-7	70	-3	41	**157**	-13	69	-5	60	0	44	1	7
127	5	44	13	36	-6	75	-6	53	**158**	0	62	-1	59	0	51	-3	23
128	31	17	-10	93	-8	89	8	76	**159**	0	65	0	59	2	46	-21	74
129	1	51	-7	73	-34	119	-9	78	**160**	21	37	22	33	19	38	16	66
130	7	50	-2	73	-3	75	-11	83	**161**	-15	72	5	44	-4	66	-23	124
131	28	19	13	46	32	20	9	52	**162**	9	57	14	43	15	38	17	37
132	16	33	9	49	30	22	0	67	**163**	16	54	-1	78	12	42	44	-18
133	14	62	-7	100	-44	127	-5	90	**164**	0	62	0	60	9	34	50	-34
134	-13	108	9	53	0	54	1	67	**165**	12	72	9	69	0	89	-22	127
135	-15	100	2	53	-5	61	-15	72									

Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
166	24	0	11	28	4	45	4	39	**197**	26	-17	28	3	36	-28	28	-3
167	15	9	2	40	10	28	0	42	**198**	30	-25	28	4	38	-28	24	10
168	8	25	3	44	10	31	7	34	**199**	28	-20	32	0	38	-27	27	0
169	13	18	0	49	33	-11	11	29	**200**	33	-23	34	-1	34	-18	34	-14
170	15	9	0	46	52	-43	8	31	**201**	37	-27	30	6	35	-16	52	-44
171	13	19	2	44	18	15	6	37	**202**	33	-23	30	6	34	-14	39	-24
172	10	37	2	51	28	0	7	42	**203**	40	-28	32	9	32	-8	19	17
173	12	18	0	47	35	-22	3	40	**204**	38	-17	31	19	37	-6	31	25
174	6	29	4	39	38	-25	8	33	**205**	33	-11	26	27	35	0	36	29
175	20	33	2	62	34	0	13	43	**206**	40	-15	26	30	30	10	24	33
176	15	30	6	46	39	-18	13	36	**207**	41	-6	37	20	28	18	34	15
177	4	45	0	54	32	-12	4	47	**208**	38	1	28	34	26	25	30	20
178	1	58	3	54	102	-94	3	55	**209**	41	17	17	70	29	41	22	73
179	0	62	2	58	0	0	2	58	**210**	30	-6	1	67	0	75	20	34
180	7	61	4	63	56	-15	6	60	**211**	27	3	5	59	2	72	19	31
181	12	38	6	51	33	-4	8	44	**212**	26	22	9	67	8	77	27	44
182	11	45	6	57	29	10	11	44	**213**	37	-16	16	30	14	35	19	16
183	15	39	7	53	37	-5	14	42	**214**	35	-4	18	32	18	31	15	36
184	11	42	6	52	51	-29	7	48	**215**	38	-8	18	35	17	35	15	36
185	13	44	6	55	39	-9	4	56	**216**	38	-3	22	29	21	30	21	28
186	16	45	11	45	52	-34	4	52	**217**	37	3	24	31	17	45	25	21
187	12	41	14	36	69	-58	13	37	**218**	38	5	23	38	20	42	30	20
188	10	49	8	53	67	-63	9	49	**219**	42	0	18	43	18	45	31	12
189	30	34	-1	82	44	-5	19	58	**220**	35	16	20	41	27	26	27	16
190	18	42	7	55	32	7	10	48	**221**	39	22	11	63	16	54	24	42
191	10	55	-3	78	55	-29	12	45	**222**	14	48	9	59	7	66	0	93
192	17	51	15	46	32	1	0	69	**223**	27	37	9	64	16	56	14	56
193	17	46	22	31	0	0	20	33	**224**	21	60	-1	94	11	73	15	57
194	0	89	-1	84	27	36	8	63	**225**	12	68	-2	89	10	67	26	38
195	26	-19	25	7	33	-25	35	-18	**226**	2	97	-9	108	-10	116	-24	127
196	22	-17	30	-7	34	-30	33	-25									

Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
227	-3	71	-6	76	-23	112	-24	115	**252**	-12	73	-6	55	-16	72	-14	75
228	-6	42	-2	44	-15	71	-22	82	**253**	-8	76	0	58	-7	69	-10	79
229	-5	50	0	45	-7	61	-9	62	**254**	-7	80	0	64	-4	69	-9	83
230	-3	54	0	52	0	53	0	53	**255**	-9	88	-3	74	-5	74	-12	92
231	-2	62	-3	64	-5	66	0	59	**256**	-17	110	-10	90	-9	86	-18	108
232	0	58	-2	59	-11	77	-14	85	**257**	-11	97	0	70	2	66	-4	79
233	1	63	-4	70	-9	80	-13	89	**258**	-20	84	-4	29	-9	34	-22	69
234	-2	72	-4	75	-9	84	-13	94	**259**	-11	79	5	31	1	32	-16	75
235	-1	74	-8	82	-10	87	-11	92	**260**	-6	73	7	42	11	31	-2	58
236	-9	91	-17	102	-34	127	-29	127	**261**	-4	74	1	59	5	52	1	58
237	-5	67	-9	77	-21	101	-21	100	**262**	-13	86	-2	58	-2	55	-13	78
238	-5	27	3	24	-3	39	-14	57	**263**	-13	96	-3	72	-2	67	-9	83
239	-3	39	0	42	-5	53	-12	67	**264**	-11	97	-3	81	0	73	-4	81
240	-2	44	0	48	-7	61	-11	71	**265**	-19	117	-11	97	-8	89	-13	99
241	0	46	0	55	-11	75	-10	77	**266**	-8	78	0	58	3	52	-13	81
242	-16	64	-6	59	-15	77	-21	85	**267**	-5	33	8	5	7	4	-6	38
243	-8	68	-7	71	-17	91	-16	88	**268**	-4	48	10	14	10	8	-13	62
244	-10	78	-12	83	-25	107	-23	104	**269**	-2	53	14	18	17	8	-6	58
245	-6	77	-11	87	-25	111	-15	98	**270**	-3	62	13	27	16	19	-2	59
246	-10	86	-30	119	-28	122	-37	127	**271**	-13	71	2	40	3	37	-16	73
247	-12	92	1	58	-11	76	-10	82	**272**	-10	79	0	58	-1	61	-10	76
248	-15	55	-3	29	-10	44	-8	48	**273**	-12	86	-3	70	-5	73	-13	86
249	-10	60	-1	36	-10	52	-8	61	**274**	-13	90	-6	79	-1	70	-9	83
250	-6	62	1	38	-10	57	-8	66	**275**	-14	97	-8	85	-4	78	-10	87
251	-4	65	2	43	-9	58	-7	70									

Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
277	-6	93	-13	106	-21	126	-22	127	**308**	-16	96	-1	51	-16	77	-10	67
278	-6	84	-16	106	-23	124	-25	127	**309**	-7	88	7	49	-2	64	1	68
279	-8	79	-10	87	-20	110	-25	120	**310**	-8	85	8	52	2	61	0	77
280	0	66	-21	114	-26	126	-27	127	**311**	-7	85	9	41	-6	67	2	64
281	-1	71	-18	110	-25	124	-19	114	**312**	-9	85	6	47	-3	64	0	68
282	0	62	-14	98	-17	105	-23	117	**313**	-13	88	2	55	2	57	-5	78
283	-2	60	-22	110	-27	121	-25	118	**314**	4	66	13	41	-3	65	7	55
284	-2	59	-21	106	-27	117	-26	117	**315**	-3	77	10	44	-3	66	5	59
285	-5	75	-18	103	-17	102	-24	113	**316**	-3	76	6	50	0	62	2	65
286	-3	62	-21	107	-26	117	-28	118	**317**	-6	76	5	53	9	51	14	54
287	-4	58	-23	108	-27	116	-31	120	**318**	10	58	13	49	-1	66	15	44
288	-9	66	-26	112	-33	122	-37	124	**319**	-1	76	4	63	-2	71	5	60
289	-1	79	-10	96	-10	95	-10	94	**320**	-1	83	6	64	-2	75	2	70
290	0	71	-12	95	-14	100	-15	102	**321**	-7	99	-2	69	-1	70	-2	76
291	3	68	-5	91	-8	95	-10	99	**322**	-14	95	-2	59	-9	72	-18	86
292	10	44	-9	93	-17	111	-13	106	**323**	2	95	6	70	14	60	12	70
293	-7	62	-22	94	-28	114	-50	127	**324**	0	76	10	44	16	37	5	64
294	15	36	-5	86	-6	89	-5	92	**325**	-5	74	9	31	0	47	-12	70
295	14	40	9	67	-2	80	17	57	**326**	0	70	12	43	18	35	11	55
296	16	27	-4	80	-4	82	-5	86	**327**	-11	75	3	53	11	37	5	56
297	12	29	-10	85	-9	85	-13	94	**328**	1	68	14	34	12	41	0	69
298	1	44	-1	70	-8	81	-12	91	**329**	0	65	10	38	10	41	2	65
299	20	36	7	60	-1	72	-2	77	**330**	-14	73	-3	52	2	48	-6	74
300	18	32	9	58	5	64	0	71	**331**	3	62	13	40	12	41	5	54
301	5	42	5	61	1	67	-1	73	**332**	4	62	17	32	13	41	7	54
302	1	48	12	50	9	56	4	64	**333**	-1	68	7	44	0	59	-6	76
303	10	62	15	50	0	69	-7	81	**334**	-13	75	7	38	3	50	-11	82
304	17	46	18	49	1	69	5	64	**335**	11	55	13	50	19	40	-2	77
305	9	64	17	54	7	69	15	57	**336**	5	64	10	57	3	66	-2	77
306	-12	104	10	41	-7	69	1	67	**337**	12	70	26	43	18	50	25	42
307	-11	97	7	46	-6	67	0	68									

Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398

ctxIdx	I and SI slices		Value of cabac_init_idc						ctxIdx	I and SI slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
338	15	6	14	11	19	-6	17	-13	**369**	32	-26	31	-4	40	-37	37	-17
339	6	19	11	14	18	-6	16	-9	**370**	37	-30	27	6	38	-30	32	1
340	7	16	9	11	14	0	17	-12	**371**	44	-32	34	8	46	-33	34	15
341	12	14	18	11	26	-12	27	-21	**372**	34	-18	30	10	42	-30	29	15
342	18	13	21	9	31	-16	37	-30	**373**	34	-15	24	22	40	-24	24	25
343	13	11	23	-2	33	-25	41	-40	**374**	40	-15	33	19	49	-29	34	22
344	13	15	32	-15	33	-22	42	-41	**375**	33	-7	22	32	38	-12	31	16
345	15	16	32	-15	37	-28	48	-47	**376**	35	-5	26	31	40	-10	35	18
346	12	23	34	-21	39	-30	39	-32	**377**	33	0	21	41	38	-3	31	28
347	13	23	39	-23	42	-30	46	-40	**378**	38	2	26	44	46	-5	33	41
348	15	20	42	-33	47	-42	52	-51	**379**	33	13	23	47	31	20	36	28
349	14	26	41	-31	45	-36	46	-41	**380**	23	35	16	65	29	30	27	47
350	14	44	46	-28	49	-34	52	-39	**381**	13	58	14	71	25	44	21	62
351	17	40	38	-12	41	-17	43	-19	**382**	29	-3	8	60	12	48	18	31
352	17	47	21	29	32	9	32	11	**383**	26	0	6	63	11	49	19	26
353	24	17	45	-24	69	-71	61	-55	**384**	22	30	17	65	26	45	36	24
354	21	21	53	-45	63	-63	56	-46	**385**	31	-7	21	24	22	22	24	23
355	25	22	48	-26	66	-64	62	-50	**386**	35	-15	23	20	23	22	27	16
356	31	27	65	-43	77	-74	81	-67	**387**	34	-3	26	23	27	21	24	30
357	22	29	43	-19	54	-39	45	-20	**388**	34	3	27	32	33	20	31	29
358	19	35	39	-10	52	-35	35	-2	**389**	36	-1	28	23	26	28	22	41
359	14	50	30	9	41	-10	28	15	**390**	34	5	28	24	30	24	22	42
360	10	57	18	26	36	0	34	1	**391**	32	11	23	40	27	34	16	60
361	7	63	20	27	40	-1	39	1	**392**	35	5	24	32	18	42	15	52
362	-2	77	0	57	30	14	30	17	**393**	34	12	28	29	25	39	14	60
363	-4	82	-14	82	28	26	20	38	**394**	39	11	23	42	18	50	3	78
364	-3	94	-5	75	23	37	18	45	**395**	30	29	19	57	12	70	-16	123
365	9	69	-19	97	12	55	15	54	**396**	34	26	22	53	21	54	21	53
366	-12	109	-35	125	11	65	0	79	**397**	29	39	22	61	14	71	22	56
367	36	-35	27	0	37	-33	36	-16	**398**	19	66	11	86	11	83	25	61
368	36	-34	28	0	39	-36	37	-14									

Table 9-24 – Values of variables m and n for ctxIdx from 402 to 459

ctxIdx	I slices		Value of cabac_init_idc						ctxIdx	I slices		Value of cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
402	-17	120	-4	79	-5	85	-3	78	**431**	-2	55	-12	56	-9	57	-12	59
403	-20	112	-7	71	-6	81	-8	74	**432**	0	61	-6	60	-6	63	-8	63
404	-18	114	-5	69	-10	77	-9	72	**433**	1	64	-5	62	-4	65	-9	67
405	-11	85	-9	70	-7	81	-10	72	**434**	0	68	-8	66	-4	67	-6	68
406	-15	92	-8	66	-17	80	-18	75	**435**	-9	92	-8	76	-7	82	-10	79
407	-14	89	-10	68	-18	73	-12	71	**436**	-14	106	-5	85	-3	81	-3	78
408	-26	71	-19	73	-4	74	-11	63	**437**	-13	97	-6	81	-3	76	-8	74
409	-15	81	-12	69	-10	83	-5	70	**438**	-15	90	-10	77	-7	72	-9	72
410	-14	80	-16	70	-9	71	-17	75	**439**	-12	90	-7	81	-6	78	-10	72
411	0	68	-15	67	-9	67	-14	72	**440**	-18	88	-17	80	-12	72	-18	75
412	-14	70	-20	62	-1	61	-16	67	**441**	-10	73	-18	73	-14	68	-12	71
413	-24	56	-19	70	-8	66	-8	53	**442**	-9	79	-4	74	-3	70	-11	63
414	-23	68	-16	66	-14	66	-14	59	**443**	-14	86	-10	83	-6	76	-5	70
415	-24	50	-22	65	0	59	-9	52	**444**	-10	73	-9	71	-5	66	-17	75
416	-11	74	-20	63	2	59	-11	68	**445**	-10	70	-9	67	-5	62	-14	72
417	23	-13	9	-2	17	-10	9	-2	**446**	-10	69	-1	61	0	57	-16	67
418	26	-13	26	-9	32	-13	30	-10	**447**	-5	66	-8	66	-4	61	-8	53
419	40	-15	33	-9	42	-9	31	-4	**448**	-9	64	-14	66	-9	60	-14	59
420	49	-14	39	-7	49	-5	33	-1	**449**	-5	58	0	59	1	54	-9	52
421	44	3	41	-2	53	0	33	7	**450**	2	59	2	59	2	58	-11	68
422	45	6	45	3	64	3	31	12	**451**	21	-10	21	-13	17	-10	9	-2
423	44	34	49	9	68	10	37	23	**452**	24	-11	33	-14	32	-13	30	-10
424	33	54	45	27	66	27	31	38	**453**	28	-8	39	-7	42	-9	31	-4
425	19	82	36	59	47	57	20	64	**454**	28	-1	46	-2	49	-5	33	-1
426	-3	75	-6	66	-5	71	-9	71	**455**	29	3	51	2	53	0	33	7
427	-1	23	-7	35	0	24	-7	37	**456**	29	9	60	6	64	3	31	12
428	1	34	-7	42	-1	36	-8	44	**457**	35	20	61	17	68	10	37	23
429	1	43	-8	45	-2	42	-11	49	**458**	29	36	55	34	66	27	31	38
430	0	54	-5	48	-2	52	-10	56	**459**	14	67	42	62	47	57	20	64

9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first macroblock of a slice or after the decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

Outputs of this process are the initialised decoding engine registers codIRange and codIOffset both in 16 bit register precision.

The status of the arithmetic decoding engine is represented by the variables codIRange and codIOffset. In the initialisation procedure of the arithmetic decoding process, codIRange is set equal to 0x01FE and codIOffset is set equal to the value returned from read_bits(9) interpreted as a 9 bit binary representation of an unsigned integer with most significant bit written first.

The bitstream shall not contain data that results in a value of codIOffset being equal to 0x01FE or 0x01FF.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, the minimum register precision for the variables codIRange and codIOffset is 9 bits.

9.3.2 Binarization process

Input to this process is a request for a syntax element.

Output of this process is the binarization of the syntax element, maxBinIdxCtx, ctxIdxOffset, and bypassFlag.

Table 9-25 specifies the type of binarization process, maxBinIdxCtx, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 9.3.2.1 to 9.3.2.4, respectively. Other binarizations are specified in subclauses 9.3.2.5 to 9.3.2.7.

Except for I slices, the binarizations for the syntax element mb_type as specified in subclause 9.3.2.5 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in 9.3.2.3, which is used for the binarization of the syntax elements mvd_lX (X = 0, 1) and coeff_abs_level_minus1, and the binarization of the coded_block_pattern also consist of a concatenation of prefix and suffix bit strings. For these binarization processes, the prefix and the suffix bit string are separately indexed using the binIdx variable as specified further in subclause 9.3.3. The two sets of prefix bit strings and suffix bit strings are referred to as the binarization prefix part and the binarization suffix part, respectively.

Associated with each binarization or binarization part of a syntax element is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table 9-25. When two values for each of these variables are specified for one syntax element in Table 9-25, the value in the upper row is related to the prefix part while the value in the lower row is related to the suffix part of the binarization of the corresponding syntax element.

The use of the DecodeBypass process and the variable bypassFlag is derived as follows.

– If no value is assigned to ctxIdxOffset for the corresponding binarization or binarization part in Table 9-25 labelled as “na”, all bins of the bit strings of the corresponding binarization or of the binarization prefix/suffix part are decoded by invoking the DecodeBypass process as specified in subclause 9.3.3.2.3. In such a case, bypassFlag is set equal to 1, where bypassFlag is used to indicate that for parsing the value of the bin from the bitstream the DecodeBypass process is applied.

– Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in Table 9-25, a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

The possible values of the context index ctxIdx are in the range 0 to 459, inclusive. The value assigned to ctxIdxOffset specifies the lower value of the range of ctxIdx assigned to the corresponding binarization or binarization part of a syntax element.

ctxIdx = ctxIdxOffset = 276 is assigned to the syntax element end_of_slice_flag and the bin of mb_type, which specifies the I_PCM macroblock type as further specified in subclause 9.3.3.1. For parsing the value of the corresponding bin from the bitstream, the arithmetic decoding process for decisions before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 is applied.

NOTE – The bins of mb_type in I slices and the bins of the suffix for mb_type in SI slices that correspond to the same value of binIdx share the same ctxIdx. The last bin of the prefix of mb_type and the first bin of the suffix of mb_type in P, SP, and B slices may share the same ctxIdx.

Table 9-25 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
mb_type (SI slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 0 suffix: 6	prefix: 0 suffix: 3
mb_type (I slices only)	as specified in subclause 9.3.2.5	6	3
mb_skip_flag (P, SP slices only)	FL, cMax=1	0	11
mb_type (P, SP slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 2 suffix: 5	prefix: 14 suffix: 17
sub_mb_type (P, SP slices only)	as specified in subclause 9.3.2.5	2	21
mb_skip_flag (B slices only)	FL, cMax=1	0	24
mb_type (B slices only)	prefix and suffix as specified in subclause 9.3.2.5	prefix: 3 suffix: 5	prefix: 27 suffix: 32
sub_mb_type (B slices only)	as specified in subclause 9.3.2.5	3	36
mvd_l0[ ][ ][0], mvd_l1[ ][ ][0]	prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9	prefix: 4 suffix: na	prefix: 40 suffix: na (uses DecodeBypass)
mvd_l0[ ][ ][1], mvd_l1[ ][ ][1]		prefix: 4 suffix: na	prefix: 47 suffix: na (uses DecodeBypass)
ref_idx_l0, ref_idx_l1	U	2	54
mb_qp_delta	as specified in subclause 9.3.2.7	2	60
intra_chroma_pred_mode	TU, cMax=3	1	64
prev_intra4x4_pred_mode_flag, prev_intra8x8_pred_mode_flag	FL, cMax=1	0	68
rem_intra4x4_pred_mode, rem_intra8x8_pred_mode	FL, cMax=7	0	69
mb_field_decoding_flag	FL, cMax=1	0	70
coded_block_pattern	prefix and suffix as specified in subclause 9.3.2.6	prefix: 3 suffix: 1	prefix: 73 suffix: 77
coded_block_flag	FL, cMax=1	0	85
significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	105
last_significant_coeff_flag (frame coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	166
coeff_abs_level_minus1 (blocks with ctxBlockCat < 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 227 suffix: na, (uses DecodeBypass)
coeff_sign_flag	FL, cMax=1	0	na, (uses DecodeBypass)
end_of_slice_flag	FL, cMax=1	0	276

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	277
last_significant_coeff_flag (field coded blocks with ctxBlockCat < 5)	FL, cMax=1	0	338
transform_size_8x8_flag	FL, cMax=1	0	399
significant_coeff_flag (frame coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	402
last_significant_coeff_flag (frame coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	417
coeff_abs_level_minus1 (blocks with ctxBlockCat = = 5)	prefix and suffix as given by UEG0 with signedValFlag=0, uCoff=14	prefix: 1 suffix: na	prefix: 426 suffix: na, (uses DecodeBypass)
significant_coeff_flag (field coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	436
last_significant_coeff_flag (field coded blocks with ctxBlockCat = = 5)	FL, cMax=1	0	451

9.3.2.1 Unary (U) binarization process

Input to this process is a request for a U binarization for a syntax element.

Output of this process is the U binarization of the syntax element.

The bin string of a syntax element having (unsigned integer) value synElVal is a bit string of length synElVal + 1 indexed by BinIdx. The bins for binIdx less than synElVal are equal to 1. The bin with binIdx equal to synElVal is equal to 0.

Table 9-26 illustrates the bin strings of the unary binarization for a syntax element.

Table 9-26 – Bin string of the unary binarization (informative)

Value of syntax element	Bin string					
0 (I_NxN)	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
…						
binIdx	0	1	2	3	4	5

9.3.2.2 Truncated unary (TU) binarization process

Input to this process is a request for a TU binarization for a syntax element and cMax.

Output of this process is the TU binarization of the syntax element.

For syntax element (unsigned integer) values less than cMax, the U binarization process as specified in subclause 9.3.2.1 is invoked. For the syntax element value equal to cMax the bin string is a bit string of length cMax with all bins being equal to 1.

NOTE – TU binarization is always invoked with a cMax value equal to the largest possible value of the syntax element being decoded.

9.3.2.3 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process

Input to this process is a request for a UEGk binarization for a syntax element, signedValFlag and uCoff.

Output of this process is the UEGk binarization of the syntax element.

A UEGk bin string is a concatenation of a prefix bit string and a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part Min(uCoff, Abs(synElVal)) of a syntax element value synElVal as specified in subclause 9.3.2.2 with cMax = uCoff, where uCoff > 0.

The UEGk bin string is derived as follows.

- If one of the following is true, the bin string of a syntax element having value synElVal consists only of a prefix bit string,
  - signedValFlag is equal to 0 and the prefix bit string is not equal to the bit string of length uCoff with all bits equal to 1.
  - signedValFlag is equal to 1 and the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0.
- Otherwise, the bin string of the UEGk suffix part of a syntax element value synElVal is specified by a process equivalent to the following pseudo-code:

```
if( Abs( synElVal )  >=  uCoff ) {
    sufS = Abs( synElVal ) − uCoff
    stopLoop = 0
    do {
        if( sufS  >=  ( 1 << k ) ) {
            put( 1 )
            sufS = sufS − ( 1<<k )
            k++
        } else {
            put( 0 )
            while( k− − )
                put( ( sufS >> k )  &  0x01 )
            stopLoop = 1
        }
    } while( !stopLoop )
}
if( signedValFlag  &&  synElVal  ! =  0)
    if( synElVal  >  0 )
        put( 0 )
    else
        put( 1 )
```

NOTE – The specification for the k-th order Exp-Golomb (EGk) code uses 1's and 0's in reverse meaning for the unary part of the Exp-Golomb code of 0-th order as specified in subclause 9.1.

9.3.2.4 Fixed-length (FL) binarization process

Input to this process is a request for a FL binarization for a syntax element and cMax.

Output of this process is the FL binarization of the syntax element.

FL binarization is constructed by using an fixedLength-bit unsigned integer bin string of the syntax element value, where fixedLength = Ceil(Log2(cMax + 1)). The indexing of bins for the FL binarization is such that the binIdx = 0 relates to the least significant bit with increasing values of binIdx towards the most significant bit.

9.3.2.5 Binarization process for macroblock type and sub-macroblock type

Input to this process is a request for a binarization for syntax elements mb_type or sub_mb_type.

Output of this process is the binarization of the syntax element.

The binarization scheme for decoding of macroblock type in I slices is specified in Table 9-27.

For macroblock types in SI slices, the binarization consists of bin strings specified as a concatenation of a prefix and a suffix bit string as follows.

The prefix bit string consists of a single bit, which is specified by b_0 = ((mb_type = = SI) ? 0 : 1). For the syntax element value for which b_0 is equal to 0, the bin string only consists of the prefix bit string. For the syntax element value for which b_0 is equal to 1, the binarization is given by concatenating the prefix b_0 and the suffix bit string as specified in Table 9-27 for macroblock type in I slices indexed by subtracting 1 from the value of mb_type in SI slices.

Table 9-27 – Binarization for macroblock types in I slices

Value (name) of mb_type	Bin string						
0 (I_4x4)	0						
1 (I_16x16_0_0_0)	1	0	0	0	0	0	
2 (I_16x16_1_0_0)	1	0	0	0	0	1	
3 (I_16x16_2_0_0)	1	0	0	0	1	0	
4 (I_16x16_3_0_0)	1	0	0	0	1	1	
5 (I_16x16_0_1_0)	1	0	0	1	0	0	0
6 (I_16x16_1_1_0)	1	0	0	1	0	0	1
7 (I_16x16_2_1_0)	1	0	0	1	0	1	0
8 (I_16x16_3_1_0)	1	0	0	1	0	1	1
9 (I_16x16_0_2_0)	1	0	0	1	1	0	0
10 (I_16x16_1_2_0)	1	0	0	1	1	0	1
11 (I_16x16_2_2_0)	1	0	0	1	1	1	0
12 (I_16x16_3_2_0)	1	0	0	1	1	1	1
13 (I_16x16_0_0_1)	1	0	1	0	0	0	
14 (I_16x16_1_0_1)	1	0	1	0	0	1	
15 (I_16x16_2_0_1)	1	0	1	0	1	0	
16 (I_16x16_3_0_1)	1	0	1	0	1	1	
17 (I_16x16_0_1_1)	1	0	1	1	0	0	0
18 (I_16x16_1_1_1)	1	0	1	1	0	0	1
19 (I_16x16_2_1_1)	1	0	1	1	0	1	0
20 (I_16x16_3_1_1)	1	0	1	1	0	1	1
21 (I_16x16_0_2_1)	1	0	1	1	1	0	0
22 (I_16x16_1_2_1)	1	0	1	1	1	0	1
23 (I_16x16_2_2_1)	1	0	1	1	1	1	0
24 (I_16x16_3_2_1)	1	0	1	1	1	1	1
25 (I_PCM)	1	1					
binIdx	0	1	2	3	4	5	6

The binarization schemes for P macroblock types in P and SP slices and for B macroblocks in B slices are specified in Table 9-28.

The bin string for I macroblock types in P and SP slices corresponding to mb_type values 5 to 30 consists of a concatenation of a prefix, which consists of a single bit with value equal to 1 as specified in Table 9-28 and a suffix as specified in Table 9-27, indexed by subtracting 5 from the value of mb_type.

mb_type equal to 4 (P_8x8ref0) is not allowed.

For I macroblock types in B slices (mb_type values 23 to 48) the binarization consists of bin strings specified as a concatenation of a prefix bit string as specified in Table 9-28 and suffix bit strings as specified in Table 9-27, indexed by subtracting 23 from the value of mb_type.

Table 9-28 – Binarization for macroblock types in P, SP, and B slices

Slice type	Value (name) of mb_type	Bin string						
P, SP slice	0 (P_L0_16x16)	0	0	0				
	1 (P_L0_L0_16x8)	0	1	1				
	2 (P_L0_L0_8x16)	0	1	0				
	3 (P_8x8)	0	0	1				
	4 (P_8x8ref0)	na						
	5 to 30 (Intra, prefix only)	1						
B slice	0 (B_Direct_16x16)	0						
	1 (B_L0_16x16)	1	0	0				
	2 (B_L1_16x16)	1	0	1				
	3 (B_Bi_16x16)	1	1	0	0	0	0	
	4 (B_L0_L0_16x8)	1	1	0	0	0	1	
	5 (B_L0_L0_8x16)	1	1	0	0	1	0	
	6 (B_L1_L1_16x8)	1	1	0	0	1	1	
	7 (B_L1_L1_8x16)	1	1	0	1	0	0	
	8 (B_L0_L1_16x8)	1	1	0	1	0	1	
	9 (B_L0_L1_8x16)	1	1	0	1	1	0	
	10 (B_L1_L0_16x8)	1	1	0	1	1	1	
	11 (B_L1_L0_8x16)	1	1	1	1	1	0	
	12 (B_L0_Bi_16x8)	1	1	1	0	0	0	0
	13 (B_L0_Bi_8x16)	1	1	1	0	0	0	1
	14 (B_L1_Bi_16x8)	1	1	1	0	0	1	0
	15 (B_L1_Bi_8x16)	1	1	1	0	0	1	1
	16 (B_Bi_L0_16x8)	1	1	1	0	1	0	0
	17 (B_Bi_L0_8x16)	1	1	1	0	1	0	1
	18 (B_Bi_L1_16x8)	1	1	1	0	1	1	0
	19 (B_Bi_L1_8x16)	1	1	1	0	1	1	1
	20 (B_Bi_Bi_16x8)	1	1	1	1	0	0	0
	21 (B_Bi_Bi_8x16)	1	1	1	1	0	0	1
	22 (B_8x8)	1	1	1	1	1	1	
	23 to 48 (Intra, prefix only)	1	1	1	1	0	1	
binIdx		0	1	2	3	4	5	6

For P, SP, and B slices the specification of the binarization for sub_mb_type is given in Table 9-29.

Table 9-29 – Binarization for sub-macroblock types in P, SP, and B slices

Slice type	Value (name) of sub_mb_type	Bin string					
P, SP slice	0 (P_L0_8x8)	1					
	1 (P_L0_8x4)	0	0				
	2 (P_L0_4x8)	0	1	1			
	3 (P_L0_4x4)	0	1	0			
B slice	0 (B_Direct_8x8)	0					
	1 (B_L0_8x8)	1	0	0			
	2 (B_L1_8x8)	1	0	1			
	3 (B_Bi_8x8)	1	1	0	0	0	
	4 (B_L0_8x4)	1	1	0	0	1	
	5 (B_L0_4x8)	1	1	0	1	0	
	6 (B_L1_8x4)	1	1	0	1	1	
	7 (B_L1_4x8)	1	1	1	0	0	0
	8 (B_Bi_8x4)	1	1	1	0	0	1
	9 (B_Bi_4x8)	1	1	1	0	1	0
	10 (B_L0_4x4)	1	1	1	0	1	1
	11 (B_L1_4x4)	1	1	1	1	0	
	12 (B_Bi_4x4)	1	1	1	1	1	
binIdx		0	1	2	3	4	5

9.3.2.6 Binarization process for coded block pattern

Input to this process is a request for a binarization for the syntax element coded_block_pattern.

Output of this process is the binarization of the syntax element.

The binarization of coded_block_pattern consists of a prefix part and (when present) a suffix part. The prefix part of the binarization is given by the FL binarization of CodedBlockPatternLuma with cMax = 15. When chroma_format_idc is not equal to 0 (monochrome), the suffix part is present and consists of the TU binarization of CodedBlockPatternChroma with cMax = 2. The relationship between the value of the syntax element coded_block_pattern and the values of CodedBlockPatternLuma and CodedBlockPatternChroma is given as specified in subclause 7.4.5.

9.3.2.7 Binarization process for mb_qp_delta

Input to this process is a request for a binarization for the syntax element mb_qp_delta.

Output of this process is the binarization of the syntax element.

The bin string of mb_qp_delta is derived by the U binarization of the mapped value of the syntax element mb_qp_delta, where the assignment rule between the signed value of mb_qp_delta and its mapped value is given as specified in Table 9-3.

9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag and ctxIdxOffset as specified in subclause 9.3.2.

Output of this process is the value of the syntax element.

This process specifies how each bit of a bit string is parsed for each syntax element.

After parsing each bit, the resulting bit string is compared to all bin strings of the binarization of the syntax element and the following applies.

– If the bit string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bit string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

When the binarization of the corresponding syntax element consists of a prefix and a suffix binarization part,, the variable binIdx is set equal to 0 for the first bin of each part of the bin string (prefix part or suffix part). In this case, after parsing the prefix bit string, the parsing process of the suffix bit string related to the binarizations specified in subclauses 9.3.2.3 and 9.3.2.5 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.3 and 9.3.2.5. Note that for the binarization of the syntax element coded_block_pattern, the suffix bit string is present regardless of the prefix bit string of length 4 as specified in subclause 9.3.2.6.

Depending on the variable bypassFlag, the following applies.

– If bypassFlag is equal to 1, the bypass decoding process as specified in subclause 9.3.3.2.3 is applied for parsing the value of the bins from the bitstream.

– Otherwise (bypassFlag is equal to 0), the parsing of each bin is specified by the following two ordered steps:

1. Given binIdx, maxBinIdxCtx and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.

2. Given ctxIdx, the value of the bin from the bitstream as specified in subclause 9.3.3.2 is decoded.

9.3.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-30 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxOffset associated with the given bin string or part thereof. The ctxIdx is determined as follows.

– If the ctxIdxOffset is listed in Table 9-30, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-30. When more than one value is listed in Table 9-30 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

– Otherwise (ctxIdxOffset is not listed in Table 9-30), the ctxIdx is specified to be the sum of the following terms: ctxIdxOffset and ctxIdxBlockCatOffset(ctxBlockCat) as specified in Table 9-31 and ctxIdxInc(ctxBlockCat). Subclause 9.3.3.1.3 specifies which ctxBlockCat is used. Subclause 9.3.3.1.1.9 specifies the assignment of ctxIdxInc(ctxBlockCat) for coded_block_flag, and subclause 9.3.3.1.3 specifies the assignment of ctxIdxInc(ctxBlockCat) for significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

All bins with binIdx greater than maxBinIdxCtx are parsed using the value of ctxIdx being assigned to binIdx equal to maxBinIdxCtx.

All entries in Table 9-30 labelled with “na” correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

ctxIdx = 276 is assigned to the binIdx of mb_type indicating the I_PCM mode. For parsing the value of the corresponding bins from the bitstream, the arithmetic decoding process for decisions before termination as specified in subclause 9.3.3.2.4 is applied.

Table 9-30 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

ctxIdxOffset	binIdx						
	0	**1**	**2**	**3**	**4**	**5**	**>= 6**
0	0,1,2 (subclause 9.3.3.1.1.3)	na	na	na	na	na	na
3	0,1,2 (subclause 9.3.3.1.1.3)	ctxIdx=276	3	4	5,6 (subclause 9.3.3.1.2)	6,7 (subclause 9.3.3.1.2)	7
11	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
14	0	1	2,3 (subclause 9.3.3.1.2)	na	na	na	na
17	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
21	0	1	2	na	na	na	na
24	0,1,2 (subclause 9.3.3.1.1.1)	na	na	na	na	na	na
27	0,1,2 (subclause 9.3.3.1.1.3)	3	4,5 (subclause 9.3.3.1.2)	5	5	5	5
32	0	ctxIdx=276	1	2	2,3 (subclause 9.3.3.1.2)	3	3
36	0	1	2,3 (subclause 9.3.3.1.2)	3	3	3	na
40	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
47	0,1,2 (subclause 9.3.3.1.1.7)	3	4	5	6	6	6
54	0,1,2,3 (subclause 9.3.3.1.1.6)	4	5	5	5	5	5
60	0,1 (subclause 9.3.3.1.1.5)	2	3	3	3	3	3
64	0,1,2 (subclause 9.3.3.1.1.8)	3	3	na	na	na	na
68	0	na	na	na	na	na	na
69	0	0	0	na	na	na	na
70	0,1,2 (subclause 9.3.3.1.1.2)	na	na	na	na	na	na
73	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	0,1,2,3 (subclause 9.3.3.1.1.4)	na	na	na
77	0,1,2,3 (subclause 9.3.3.1.1.4)	4,5,6,7 (subclause 9.3.3.1.1.4)	na	na	na	na	na
276	0	na	na	na	na	na	na
399	0,1,2 (subclause 9.3.3.1.1.10)	na	na	na	na	na	na

Table 9-31 shows the values of ctxIdxBlockCatOffset depending on ctxBlockCat for the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1. The specification of ctxBlockCat is given in Table 9-33.

Table 9-31 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Syntax element	ctxBlockCat (as specified in Table 9-33)					
	0	**1**	**2**	**3**	**4**	**5**
coded_block_flag	0	4	8	12	16	na
significant_coeff_flag	0	15	29	44	47	0
last_significant_coeff_flag	0	15	29	44	47	0
coeff_abs_level_minus1	0	10	20	30	39	0

9.3.3.1.1 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause 9.3.3.1.1.1 specifies the derivation process of ctxIdxInc for the syntax element mb_skip_flag.

Subclause 9.3.3.1.1.2 specifies the derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag.

Subclause 9.3.3.1.1.3 specifies the derivation process of ctxIdxInc for the syntax element mb_type.

Subclause 9.3.3.1.1.4 specifies the derivation process of ctxIdxInc for the syntax element coded_block_pattern.

Subclause 9.3.3.1.1.5 specifies the derivation process of ctxIdxInc for the syntax element mb_qp_delta.

Subclause 9.3.3.1.1.6 specifies the derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1.

Subclause 9.3.3.1.1.7 specifies the derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1.

Subclause 9.3.3.1.1.8 specifies the derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode.

Subclause 9.3.3.1.1.9 specifies the derivation process of ctxIdxInc for the syntax element coded_block_flag.

Subclause 9.3.3.1.1.10 specifies the derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag.

9.3.3.1.1.1 Derivation process of ctxIdxInc for the syntax element mb_skip_flag

Output of this process is ctxIdxInc.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied.

The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If mbAddrN is not available or mb_skip_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.

– Otherwise (mbAddrN is available and mb_skip_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \tag{9-1}$$

9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblock addresses and their availability in MBAFF frames as specified in subclause 6.4.7 is invoked and the output is assigned to mbAddrA and mbAddrB.

When both macroblocks mbAddrN and mbAddrN + 1 have mb_type equal to P_Skip or B_Skip, the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 is applied for the macroblock mbAddrN.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0,
  – mbAddrN is not available
  – the macroblock mbAddrN is a frame macroblock.
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \tag{9-2}$$

9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax element mb_type

Input to this process is ctxIdxOffset.

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0
  – mbAddrN is not available
  – ctxIdxOffset is equal to 0 and mb_type for the macroblock mbAddrN is equal to SI
  – ctxIdxOffset is equal to 3 and mb_type for the macroblock mbAddrN is equal to I_NxN
  – ctxIdxOffset is equal to 27 and mb_type for the macroblock mbAddrN is equal to P_Skip, B_Skip, or B_Direct_16x16
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \tag{9-3}$$

9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element coded_block_pattern

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Depending on the value of the variable ctxIdxOffset, the following applies.

– If ctxIdxOffset is equal to 73, the following applies
  – The derivation process for neighbouring 8x8 luma blocks specified in subclause 6.4.8.2 is invoked with luma8x8BlkIdx = binIdx as input and the output is assigned to mbAddrA, mbAddrB, luma8x8BlkIdxA, and luma8x8BlkIdxB.
  – Let the variable condTermFlagN (with N being either A or B) be derived as follows.
    – If any of the following conditions are true, condTermFlagN is set equal to 0
      – mbAddrN is not available
      – mb_type for the macroblock mbAddrN is equal to I_PCM

- the macroblock mbAddrN is not the current macroblock CurrMbAddr and the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, and ((CodedBlockPatternLuma >> luma8x8BlkIdxN) & 1) is not equal to 0 for the value of CodedBlockPatternLuma for the macroblock mbAddrN
- the macroblock mbAddrN is the current macroblock CurrMbAddr and the prior decoded bin value b_k of coded_block_pattern with k = luma8x8BlkIdxN is not equal to 0.

– Otherwise, condTermFlagN is set equal to 1.

– The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB \quad (9\text{-}4)$$

– Otherwise (ctxIdxOffset is equal to 77), the following applies.

– The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

– Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If mbAddrN is available and mb_type for the macroblock mbAddrN is equal to I_PCM, condTermFlagN is set equal to 1

– Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 0

– mbAddrN is not available or the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip

– binIdx is equal to 0 and CodedBlockPatternChroma for the macroblock mbAddrN is equal to 0

– binIdx is equal to 1 and CodedBlockPatternChroma for the macroblock mbAddrN is not equal to 2

– Otherwise, condTermFlagN is set equal to 1.

– The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB + ((binIdx == 1) ? 4 : 0) \quad (9\text{-}5)$$

NOTE – When a macroblock uses an Intra_16x16 prediction mode, the values of CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock are derived from mb_type as specified in Table 7-11.

9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element mb_qp_delta

Output of this process is ctxIdxInc.

Let prevMbAddr be the macroblock address of the macroblock that precedes the current macroblock in decoding order. When the current macroblock is the first macroblock of a slice, prevMbAddr is marked as not available.

Let the variable ctxIdxInc be derived as follows.

– If any of the following conditions is true, ctxIdxInc is set equal to 0

– prevMbAddr is not available or the macroblock prevMbAddr has mb_type equal to P_Skip or B_Skip

– mb_type of the macroblock prevMbAddr is equal to I_PCM

– The macroblock prevMbAddr is not coded in Intra_16x16 prediction mode and both CodedBlockPatternLuma and CodedBlockPatternChroma for the macroblock prevMbAddr are equal to 0

– mb_qp_delta for the macroblock prevMbAddr is equal to 0

– Otherwise, ctxIdxInc is set equal to 1.

9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax elements ref_idx_l0 and ref_idx_l1

Input to this process is mbPartIdx.

Output of this process is ctxIdxInc.

The interpretation of ref_idx_lX and Pred_LX within this subclause is specified as follows.

– If this process is invoked for the derivation of ref_idx_l0, ref_idx_lX is interpreted as ref_idx_l0 and Pred_LX is interpreted as Pred_L0.

– Otherwise (this process is invoked for the derivation of ref_idx_l1), ref_idx_lX is interpreted as ref_idx_l1 and Pred_LX is interpreted as Pred_L1.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for neighbouring partitions specified in subclause 6.4.8.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx = 0 as input and the output is assigned to mbAddrA\mbPartIdxA and mbAddrB\mbPartIdxB.

With ref_idx_lX[mbPartIdxN] (with N being either A or B) specifying the syntax element for the macroblock mbAddrN, let the variable refIdxZeroFlagN be derived as follows.

– If MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

$$refIdxZeroFlagN = ((ref_idx_lX[mbPartIdxN] > 1) ? 0 : 1) \quad (9\text{-}6)$$

– Otherwise,

$$refIdxZeroFlagN = ((ref_idx_lX[mbPartIdxN] > 0) ? 0 : 1) \quad (9\text{-}7)$$

Let the variable predModeEqualFlagN be specified as follows.

– If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.

  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

– Otherwise, the following applies.

  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0

  – mbAddrN is not available

  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip

  – The macroblock mbAddrN is coded in Intra prediction mode

  – predModeEqualFlagN is equal to 0

  – refIdxZeroFlagN is equal to 1

– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived as

$$ctxIdxInc = condTermFlagA + 2 * condTermFlagB \quad (9\text{-}8)$$

9.3.3.1.1.7 Derivation process of ctxIdxInc for the syntax elements mvd_l0 and mvd_l1

Inputs to this process are mbPartIdx, subMbPartIdx, and ctxIdxOffset.

Output of this process is ctxIdxInc.

The interpretation of mvd_lX and Pred_LX within this subclause is specified as follows.

– If this process is invoked for the derivation of mvd_l0, mvd_lX is interpreted as mvd_l0 and Pred_LX is interpreted as Pred_L0.

– Otherwise (this process is invoked for the derivation of mvd_l1), mvd_lX is interpreted as mvd_l1 and Pred_LX is interpreted as Pred_L1.

Let currSubMbType be set equal to sub_mb_type[mbPartIdx].

The derivation process for neighbouring partitions specified in subclause 6.4.8.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrA\mbPartIdxA\subMbPartIdxA and mbAddrB\mbPartIdxB\subMbPartIdxB.

Let the variable compIdx be derived as follows.

– If ctxIdxOffset is equal to 40, compIdx is set equal to 0.

– Otherwise (ctxIdxOffset is equal to 47), compIdx is set equal to 1.

Let the variable predModeEqualFlagN be specified as follows.

– If the macroblock mbAddrN has mb_type equal to P_8x8 or B_8x8, the following applies.

  – If SubMbPredMode(sub_mb_type[mbPartIdxN]) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where sub_mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

– Otherwise, the following applies.

  – If MbPartPredMode(mb_type, mbPartIdxN) is not equal to Pred_LX and not equal to BiPred, predModeEqualFlagN is set equal to 0, where mb_type specifies the syntax element for the macroblock mbAddrN.

  – Otherwise, predModeEqualFlagN is set equal to 1.

Let the variable absMvdCompN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, absMvdCompN is set equal to 0

  – mbAddrN is not available

  – the macroblock mbAddrN has mb_type equal to P_Skip or B_Skip

  – The macroblock mbAddrN is coded in an Intra prediction mode

  – predModeEqualFlagN is equal to 0

– Otherwise, the following applies

  – If compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a frame macroblock, and the macroblock mbAddrN is a field macroblock

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) * 2 (9-9)

  – Otherwise, if compIdx is equal to 1, MbaffFrameFlag is equal to 1, the current macroblock is a field macroblock, and the macroblock mbAddrN is a frame macroblock

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) / 2 (9-10)

  – Otherwise,

    absMvdCompN = Abs(mvd_lX[mbPartIdxN][subMbPartIdxN][compIdx]) (9-11)

The variable ctxIdxInc is derived as follows

– If (absMvdCompA + absMvdCompB) is less than 3, ctxIdxInc is set equal to 0.

– Otherwise, if (absMvdCompA + absMvdCompB) is greater than 32, ctxIdxInc is set equal to 2.

– Otherwise ((absMvdCompA + absMvdCompB) is in the range of 3 to 32, inclusive), ctxIdxInc is set equal to 1.

9.3.3.1.1.8 Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being replaced by either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0

  – mbAddrN is not available

  – The macroblock mbAddrN is coded in Inter prediction mode

  – mb_type for the macroblock mbAddrN is equal to I_PCM

  – intra_chroma_pred_mode for the macroblock mbAddrN is equal to 0

– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \tag{9-12}$$

9.3.3.1.1.9 Derivation process of ctxIdxInc for the syntax element coded_block_flag

Input to this process is ctxBlockCat and additional input is specified as follows.

– If ctxBlockCat is equal to 0, no additional input

– Otherwise, if ctxBlockCat is equal to 1 or 2, luma4x4BlkIdx

– Otherwise, if ctxBlockCat is equal to 3, the chroma component index iCbCr

– Otherwise (ctxBlockCat is equal to 4), chroma4x4BlkIdx and the chroma component index iCbCr

Output of this process is ctxIdxInc(ctxBlockCat).

Let the variable transBlockN (with N being either A or B) be derived as follows.

– If ctxBlockCat is equal to 0, the following applies.

  – The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).

  – The variable transBlockN is derived as follows.

    – If mbAddrN is available and the macroblock mbAddrN is coded in Intra_16x16 prediction mode, the luma DC block of macroblock mbAddrN is assigned to transBlockN

    – Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 1 or 2, the following applies.

  – The derivation process for neighbouring 4x4 luma blocks specified in subclause 6.4.8.3 is invoked with luma4x4BlkIdx as input and the output is assigned to mbAddrN, luma4x4BlkIdxN (with N being either A or B).

  – The variable transBlockN is derived as follows.

    – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the

macroblock mbAddrN, and transform_size_8x8_flag is equal to 0 for the macroblock mbAddrN, the 4x4 luma block with index luma4x4BlkIdxN of macroblock mbAddrN is assigned to transBlockN.

  – Otherwise, if mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip or B_Skip, ((CodedBlockPatternLuma >> (luma4x4BlkIdxN >>2)) & 1) is not equal to 0 for the macroblock mbAddrN, and transform_size_8x8_flag is equal to 1 for the macroblock mbAddrN, the 8x8 luma block with index (luma4x4BlkIdxN >> 2) of macroblock mbAddrN is assigned to transBlockN.

  – Otherwise, transBlockN is marked as not available.

– Otherwise, if ctxBlockCat is equal to 3, the following applies.

  – The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrN (with N being either A or B).

  – The variable transBlockN is derived as follows.

    – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is not equal to 0 for the macroblock mbAddrN, the chroma DC block of chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.

    – Otherwise, transBlockN is marked as not available.

– Otherwise (ctxBlockCat is equal to 4), the following applies.

  – The derivation process for neighbouring 4x4 chroma blocks specified in subclause 6.4.8.4 is invoked with chroma4x4BlkIdx as input and the output is assigned to mbAddrN, chroma4x4BlkIdxN (with N being either A or B).

  – The variable transBlockN is derived as follows.

    – If mbAddrN is available, the macroblock mbAddrN does not have mb_type equal to P_Skip, B_Skip, or I_PCM, and CodedBlockPatternChroma is equal to 2 for the macroblock mbAddrN, the 4x4 chroma block with chroma4x4BlkIdxN of the chroma component iCbCr of macroblock mbAddrN is assigned to transBlockN.

    – Otherwise, transBlockN is marked as not available.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0

  – mbAddrN is not available and the current macroblock is coded in Inter prediction mode

  – mbAddrN is available and transBlockN is not available and mb_type for the macroblock mbAddrN is not equal to I_PCM

  – The current macroblock is coded in Intra prediction mode, constrained_intra_pred_flag is equal to 1, the macroblock mbAddrN is available and coded in Inter prediction mode, and slice data partitioning is in use (nal_unit_type is in the range of 2 through 4, inclusive).

– Otherwise, if any of the following conditions is true, condTermFlagN is set equal to 1

  – mbAddrN is not available and the current macroblock is coded in Intra prediction mode

  – mb_type for the macroblock mbAddrN is equal to I_PCM

– Otherwise, condTermFlagN is set equal to the value of the coded_block_flag of the transform block transBlockN that was decoded for the macroblock mbAddrN.

The variable ctxIdxInc(ctxBlockCat) is derived by

$$\text{ctxIdxInc(ctxBlockCat)} = \text{condTermFlagA} + 2 * \text{condTermFlagB} \qquad (9\text{-}13)$$

9.3.3.1.1.10 Derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag

Output of this process is ctxIdxInc.

The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If any of the following conditions is true, condTermFlagN is set equal to 0.
  – mbAddrN is not available
  – transform_size_8x8_flag for the macroblock mbAddrN is equal to 0
– Otherwise, condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \tag{9-14}$$

9.3.3.1.2 Assignment process of ctxIdxInc using prior decoded bin values

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

Table 9-32 contains the specification of ctxIdxInc for the given values of ctxIdxOffset and binIdx.

For each value of ctxIdxOffset and binIdx, ctxIdxInc is derived by using some of the values of prior decoded bin values (b_0, b_1, b_2,…, b_k), where the value of the index k is less than the value of binIdx.

Table 9-32 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx

Value (name) of ctxIdxOffset	binIdx	ctxIdxInc
3	4	(b_3 != 0) ? 5: 6
	5	(b_3 != 0) ? 6: 7
14	2	(b_1 != 1) ? 2: 3
17	4	(b_3 != 0) ? 2: 3
27	2	(b_1 != 0) ? 4: 5
32	4	(b_3 != 0) ? 2: 3
36	2	(b_1 != 0) ? 2: 3

9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-33.

Table 9-33 – Specification of ctxBlockCat for the different blocks

Block description	maxNumCoeff	ctxBlockCat
block of luma DC transform coefficient levels (i.e., list Intra16x16DCLevel as described in subclause 7.4.5.3)	16	0
block of luma AC transform coefficient levels (i.e., list Intra16x16ACLevel[i] as described in subclause 7.4.5.3)	15	1
block of 16 luma transform coefficient levels (i.e., list LumaLevel[i] as described in subclause 7.4.5.3)	16	2
block of chroma DC transform coefficient levels	4 * NumC8x8	3
block of chroma AC transform coefficient levels	15	4
block of 64 luma transform coefficient levels (i.e., list LumaLevel8x8[i] as described in subclause 7.4.5.3)	64	5

Let the variable levelListIdx be set equal to the index of the list of transform coefficient levels as specified in subclause 7.4.5.3.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat < 5 and ctxBlockCat != 3, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{levelListIdx} \tag{9-15}$$

where levelListIdx ranges from 0 to maxNumCoeff – 2, inclusive.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat = = 3, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{Min}(\text{levelListIdx} / \text{NumC8x8}, 2) \tag{9-16}$$

where levelListIdx ranges from 0 to 4 * NumC8x8 – 2, inclusive.

For the syntax elements significant_coeff_flag and last_significant_coeff_flag in 8x8 luma blocks with ctxBlockCat = = 5, Table 9-34 contains the specification of ctxIdxInc for the given values of levelListIdx, where levelListIdx ranges from 0 to 62, inclusive.

Table 9-34 – Mapping of scanning position to ctxIdxInc for ctxBlockCat = = 5

levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag	levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag
0	0	0	0	**32**	7	9	3
1	1	1	1	**33**	6	9	3
2	2	1	1	**34**	11	10	3
3	3	2	1	**35**	12	10	3
4	4	2	1	**36**	13	8	3
5	5	3	1	**37**	11	11	3
6	5	3	1	**38**	6	12	3

levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag	levelListIdx	ctxIdxInc for significant_coeff_flag (frame coded macroblocks)	ctxIdxInc for significant_coeff_flag (field coded macroblocks)	ctxIdxInc for last_significant_coeff_flag
7	4	4	1	**39**	7	11	3
8	4	5	1	**40**	8	9	4
9	3	6	1	**41**	9	9	4
10	3	7	1	**42**	14	10	4
11	4	7	1	**43**	10	10	4
12	4	7	1	**44**	9	8	4
13	4	8	1	**45**	8	13	4
14	5	4	1	**46**	6	13	4
15	5	5	1	**47**	11	9	4
16	4	6	2	**48**	12	9	5
17	4	9	2	**49**	13	10	5
18	4	10	2	**50**	11	10	5
19	4	10	2	**51**	6	8	5
20	3	8	2	**52**	9	13	6
21	3	11	2	**53**	14	13	6
22	6	12	2	**54**	10	9	6
23	7	11	2	**55**	9	9	6
24	7	9	2	**56**	11	10	7
25	7	9	2	**57**	12	10	7
26	8	10	2	**58**	13	14	7
27	9	10	2	**59**	11	14	7
28	10	8	2	**60**	14	14	8
29	9	11	2	**61**	10	14	8
30	8	12	2	**62**	12	14	8
31	7	11	2				

Let numDecodAbsLevelEq1 denotes the accumulated number of decoded transform coefficient levels with absolute value equal to 1, and let numDecodAbsLevelGt1 denotes the accumulated number of decoded transform coefficient levels with absolute value greater than 1. Both numbers are related to the same transform coefficient block, where the

current decoding process takes place. Then, for decoding of coeff_abs_level_minus1, ctxIdxInc for coeff_abs_level_minus1 is specified depending on binIdx as follows.

– If binIdx is equal to 0, ctxIdxInc is derived by

ctxIdxInc = ((numDecodAbsLevelGt1 != 0) ? 0: Min(4, 1 + numDecodAbsLevelEq1)) (9-17)

– Otherwise (binIdx is greater than 0), ctxIdxInc is derived by

ctxIdxInc = 5 + Min(4 – (ctxBlockCat == 3), numDecodAbsLevelGt1) (9-18)

9.3.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdx as derived in subclause 9.3.3.1, and the state variables codIRange and codIOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9-2 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdx), which is specified as follows.

– If bypassFlag is equal to 1, DecodeBypass( ) as specified in subclause 9.3.3.2.3 is invoked.

– Otherwise, if bypassFlag is equal to 0 and ctxIdx is equal to 276, DecodeTerminate( ) as specified in subclause 9.3.3.2.4 is invoked.

– Otherwise (bypassFlag is equal to 0 and ctxIdx is not equal to 276), DecodeDecision( ) as specified in subclause 9.3.3.2.1 is applied.

Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative)

NOTE – Arithmetic coding is based on the principle of recursive interval subdivision. Given a probability estimation p(0) and p(1) = 1 – p(0) of a binary decision (0, 1), an initially given code sub-interval with the range codIRange will be subdivided into two sub-intervals having range p(0) * codIRange and codIRange – p(0) * codIRange, respectively. Depending on the decision, which has been observed, the corresponding sub-interval will be chosen as the new code interval, and a binary code string pointing into that interval will represent the sequence of observed binary decisions. It is useful to distinguish between the most probable symbol (MPS) and the least probable symbol (LPS), so that binary decisions have to be identified as either MPS or LPS, rather than 0 or 1. Given this terminology, each context is specified by the probability p_{LPS} of the LPS and the value of MPS (valMPS), which is either 0 or 1.

The arithmetic core engine in this Recommendation | International Standard has three distinct properties:

– The probability estimation is performed by means of a finite-state machine with a table-based transition process between 64 different representative probability states { p_{LPS}(pStateIdx) | 0 <= pStateIdx < 64 } for the LPS probability p_{LPS}. The numbering of the states is arranged in such a way that the probability state with index pStateIdx = 0 corresponds to an LPS probability value of 0.5, with decreasing LPS probability towards higher state indices.
– The range codIRange representing the state of the coding engine is quantised to a small set $\{Q_1,\ldots,Q_4\}$ of pre-set quantisation values prior to the calculation of the new interval range. Storing a table containing all 64x4 pre-computed product values of $Q_i * p_{LPS}$(pStateIdx) allows a multiplication-free approximation of the product codIRange * p_{LPS}(pStateIdx).
– For syntax elements or parts thereof for which an approximately uniform probability distribution is assumed to be given a separate simplified encoding and decoding bypass process is used.

9.3.3.2.1 Arithmetic decoding process for a binary decision

Inputs to this process are ctxIdx, codIRange, and codIOffset.

Outputs of this process are the decoded value binVal, and the updated variables codIRange and codIOffset.

Figure 9-3 shows the flowchart for decoding a single decision (DecodeDecision).

1. The value of the variable codIRangeLPS is derived as follows.
   – Given the current value of codIRange, the variable qCodIRangeIdx is derived by

   qCodIRangeIdx =(codIRange >> 6) & 0x03 (9-19)

   – Given qCodIRangeIdx and pStateIdx associated with ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-35 is assigned to codIRangeLPS:

   codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx] (9-20)

2. The variable codIRange is set equal to codIRange – codIRangeLPS and the following applies.
   – If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 - valMPS, codIOffset is decremented by codIRange, and codIRange is set equal to codIRangeLPS.
   – Otherwise, the variable binVal is set equal to valMPS.

Given the value of binVal, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.3.2.2.

9.3.3.2.1.1 State transition process

Inputs to this process are the current pStateIdx, the decoded value binVal and valMPS values of the context variable associated with ctxIdx.

Outputs of this process are the updated pStateIdx and valMPS of the context variable associated with ctxIdx.

Depending on the decoded value binVal, the update of the two variables pStateIdx and valMPS associated with ctxIdx is derived as follows:

```
if( binVal  = =  valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {                                                    (9-21)
    if( pStateIdx  = =  0 )
        valMPS = 1 – valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}
```

Table 9-36 specifies the transition rules transIdxMPS( ) and transIdxLPS( ) after decoding the value of valMPS and 1 – valMPS, respectively.

Figure 9-3 – Flowchart for decoding a decision

Table 9-35 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx

pStateIdx	qCodIRangeIdx				pStateIdx	qCodIRangeIdx			
	0	**1**	**2**	**3**		**0**	**1**	**2**	**3**
0	128	176	208	240	**32**	27	33	39	45
1	128	167	197	227	**33**	26	31	37	43
2	128	158	187	216	**34**	24	30	35	41
3	123	150	178	205	**35**	23	28	33	39
4	116	142	169	195	**36**	22	27	32	37
5	111	135	160	185	**37**	21	26	30	35
6	105	128	152	175	**38**	20	24	29	33
7	100	122	144	166	**39**	19	23	27	31
8	95	116	137	158	**40**	18	22	26	30
9	90	110	130	150	**41**	17	21	25	28
10	85	104	123	142	**42**	16	20	23	27
11	81	99	117	135	**43**	15	19	22	25
12	77	94	111	128	**44**	14	18	21	24
13	73	89	105	122	**45**	14	17	20	23
14	69	85	100	116	**46**	13	16	19	22
15	66	80	95	110	**47**	12	15	18	21
16	62	76	90	104	**48**	12	14	17	20
17	59	72	86	99	**49**	11	14	16	19
18	56	69	81	94	**50**	11	13	15	18
19	53	65	77	89	**51**	10	12	15	17
20	51	62	73	85	**52**	10	12	14	16
21	48	59	69	80	**53**	9	11	13	15
22	46	56	66	76	**54**	9	11	12	14
23	43	53	63	72	**55**	8	10	12	14
24	41	50	59	69	**56**	8	9	11	13
25	39	48	56	65	**57**	7	9	11	12
26	37	45	54	62	**58**	7	9	10	12
27	35	43	51	59	**59**	7	8	10	11
28	33	41	48	56	**60**	6	8	9	11
29	32	39	46	53	**61**	6	7	9	10
30	30	37	43	50	**62**	6	7	8	9
31	29	35	41	48	**63**	2	2	2	2

Table 9-36 – State transition table

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

9.3.3.2.2 Renormalization process in the arithmetic decoding engine

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset.

A flowchart of the renormalization is shown in Figure 9-4. The current value of codIRange is first compared to 0x0100 and further steps are specified as follows.

– If codIRange is greater than or equal to 0x0100, no renormalization is needed and the RenormD process is finished;

– Otherwise (codIRange is less than 0x0100), the renormalization loop is entered. Within this loop, the value of codIRange is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1).

The bitstream shall not contain data that results in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

Figure 9-4 – Flowchart of renormalization

9.3.3.2.3 Bypass decoding process for binary decisions

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variable codIOffset and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9-5 shows a flowchart of the corresponding process.

First, the value of codIOffset is doubled, i.e., left-shifted by 1 and a single bit is shifted into codIOffset by using read_bits(1). Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1 and codIOffset is decremented by codIRange.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0.

The bitstream shall not contain data that results in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

Figure 9-5 – Flowchart of bypass decoding process

9.3.3.2.4 Decoding process for binary decisions before termination

Inputs to this process are bits from slice data and the variables codIRange and codIOffset.

Outputs of this process are the updated variables codIRange and codIOffset, and the decoded value binVal.

This special decoding routine applies to decoding of end_of_slice_flag and of the bin indicating the I_PCM mode corresponding to ctxIdx equal to 276. Figure 9-6 shows the flowchart of the corresponding decoding process, which is specified as follows.

First, the value of codIRange is decremented by 2. Then, the value of codIOffset is compared to the value of codIRange and further steps are specified as follows.

– If codIOffset is greater than or equal to codIRange, the variable binVal is set equal to 1, no renormalization is carried out, and CABAC decoding is terminated. The last bit inserted in register codIOffset is equal to 1. When decoding end_of_slice_flag, this last bit inserted in register codIOffset is interpreted as rbsp_stop_one_bit.

– Otherwise (codIOffset is less than codIRange), the variable binVal is set equal to 0 and renormalization is performed as specified in subclause 9.3.3.2.2.

NOTE – This procedure may also be implemented using DecodeDecision(ctxIdx) with ctxIdx = 276. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision(ctxIdx) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

Figure 9-6 – Flowchart of decoding a decision before termination

9.3.4 Arithmetic encoding process (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are decisions that are to be encoded and written.

Outputs of this process are bits that are written to the RBSP.

This informative subclause describes an arithmetic encoding engine that matches the arithmetic decoding engine described in subclause 9.3.3.2. The encoding engine is essentially symmetric with the decoding engine, i.e., procedures are called in the same order. The following procedures are described in this section: InitEncoder, EncodeDecision, EncodeBypass, EncodeTerminate, which correspond to InitDecoder, DecodeDecision, DecodeBypass, and DecodeTerminate, respectively. The state of the arithmetic encoding engine is represented by a value of the variable codILow pointing to the lower end of a sub-interval and a value of the variable codIRange specifying the corresponding range of that sub-interval.

9.3.4.1 Initialisation process for the arithmetic encoding engine (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

This process is invoked before encoding the first macroblock of a slice, and after encoding any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data for a macroblock of type I_PCM.

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and symCnt of the arithmetic encoding engine.

In the initialisation procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 0x01FE. Furthermore, a firstBitFlag is set equal to 1, and bitsOutstanding and symCnt counters are set equal to 0.

NOTE – The minimum register precision required for codILow is 10 bits and for CodIRange is 9 bits. The precision required for the counters bitsOutstanding and symCnt should be sufficiently large to prevent overflow of the related registers. When MaxBinCountInSlice denotes the maximum total number of binary decisions to encode in one slice, the minimum register precision required for the variables bitsOutstanding and symCnt is given by Ceil(Log2(MaxBinCountInSlice + 1)).

9.3.4.2 Encoding process for a binary decision (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the context index ctxIdx, the value of binVal to be encoded, and the variables codIRange, codILow and symCnt.

Outputs of this process are the variables codIRange, codILow, and symCnt.

Figure 9-7 shows the flowchart for encoding a single decision. In a first step, the variable codIRangeLPS is derived as follows.

Given the current value of codIRange, codIRange is mapped to the index qCodIRangeIdx of a quantised value of codIRange by using Equation 9-19. The value of qCodIRangeIdx and the value of pStateIdx associated with ctxIdx are used to determine the value of the variable rangeTabLPS as specified in Table 9-35, which is assigned to codIRangeLPS. The value of codIRange – codIRangeLPS is assigned to codIRange.

In a second step, the value of binVal is compared to valMPS associated with ctxIdx. When binVal is different from valMPS, codIRange is added to codILow and codIRange is set equal to the value codIRangeLPS. Given the encoded decision, the state transition is performed as specified in subclause 9.3.3.2.1.1. Depending on the current value of codIRange, renormalization is performed as specified in subclause 9.3.4.3. Finally, the variable symCnt is incremented by 1.

Figure 9-7 – Flowchart for encoding a decision

9.3.4.3 Renormalization process in the arithmetic encoding engine (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codIRange, codILow, firstBitFlag, and bitsOutstanding.

Renormalization is illustrated in Figure 9-8.

Figure 9-8 – Flowchart of renormalization in the encoder

The PutBit( ) procedure described in Figure 9-9 provides carry over control. It uses the function WriteBits(B, N) that writes N bits with value B to the bitstream and advances the bitstream pointer by N bit positions. This function assumes the existence of a bitstream pointer with an indication of the position of the next bit to be written to the bitstream by the encoding process.

Figure 9-9 – Flowchart of PutBit(B)

9.3.4.4 Bypass encoding process for binary decisions (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables binVal, codILow, codIRange, bitsOutstanding, and symCnt.

Output of this process is a bit written to the RBSP and the updated variables codILow, bitsOutstanding, and symCnt.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in Figure 9-10.

Figure 9-10 – Flowchart of encoding bypass

9.3.4.5 Encoding process for a binary decision before termination (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Inputs to this process are the variables binVal, codIRange, codILow, bitsOutstanding, and symCnt.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codILow, codIRange, bitsOutstanding, and symCnt.

This encoding routine shown in Figure 9-11 applies to encoding of the end_of_slice_flag and of the bin indicating the I_PCM mb_type both associated with ctxIdx equal to 276.

Figure 9-11 – Flowchart of encoding a decision before termination

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in Figure 9-12 is applied. In this flushing procedure, the last bit written by WriteBits(B, N) is equal to 1. When encoding end_of_slice_flag, this last bit is interpreted as the rbsp_stop_one_bit.

Figure 9-12 – Flowchart of flushing at termination

9.3.4.6 Byte stuffing process (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

This process is invoked after encoding the last macroblock of the last slice of a picture and after encapsulation.

Inputs to this process are the number of bytes NumBytesInVclNALunits of all VCL NAL units of a picture, the number of macroblocks PicSizeInMbs in the picture, and the number of binary symbols BinCountsInNALunits resulting from encoding the contents of all VCL NAL units of the picture.

Outputs of this process are zero or more bytes appended to the NAL unit.

Let the variable k be set equal to Ceil((Ceil(3 * (32 * BinCountsInNALunits – RawMbBits * PicSizeInMbs) ÷ 1024) – NumBytesInVclNALunits) ÷ 3). Depending on the variable k the following applies.

– If k is less than or equal to 0, no cabac_zero_word is appended to the NAL unit.

– Otherwise (k is greater than 0), the 3-byte sequence 0x000003 is appended k times to the NAL unit after encapsulation, where the first two bytes 0x0000 represent a cabac_zero_word and the third byte 0x03 represents an emulation_prevention_three_byte.

AAnnex A
Profiles and levels

(This annex forms an integral part of this Recommendation | International Standard)

[Ed. (JR): there is probably a need to update this annex to handle annex F (SVC)]

Profiles and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles and levels may also be used to indicate interoperability points between individual decoder implementations.

NOTE 1 – This Recommendation | International Standard does not include individually selectable "options" at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

NOTE 2 – Encoders are not required to make use of any particular subset of features supported in a profile.

Each level specifies a set of limits on the values that may be taken by the syntax elements of this Recommendation | International Standard. The same set of level definitions is used with all profiles, but individual implementations may support a different level for each supported profile. For any given profile, levels generally correspond to decoder processing load and memory capability.

A.1 Requirements on video decoder capability

Capabilities of video decoders conforming to this Recommendation | International Standard are specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels specified in this Annex. For each such profile, the level supported for that profile shall also be expressed.

Specific values are specified in this annex for the syntax elements profile_idc and level_idc. All other values of profile_idc and level_idc are reserved for future use by ITU-T | ISO/IEC.

NOTE – Decoders should not infer that when a reserved value of profile_idc or level_idc falls between the values specified in this Recommendation | International Standard that this indicates intermediate capabilities between the specified profiles or levels, as there are no restrictions on the method to be chosen by ITU-T | ISO/IEC for the use of such future reserved values.

A.2 Profiles

A.2.1 Baseline profile

Bitstreams conforming to the Baseline profile shall obey the following constraints:

– Only I and P slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Sequence parameter sets shall have frame_mbs_only_flag equal to 1.

– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.

– Picture parameter sets shall have weighted_pred_flag and weighted_bipred_idc both equal to 0.

– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.

– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.

– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.

– The syntax element level_prefix shall not be greater than 15.

– The level constraints specified for the Baseline profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Baseline profile is specified by profile_idc being equal to 66.

Decoders conforming to the Baseline profile at a specific level shall be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1 and in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.2 Main profile

Bitstreams conforming to the Main profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.

– The syntax element level_prefix shall not be greater than 15 (when present).

– The level constraints specified for the Main profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Main profile is specified by profile_idc being equal to 77.

Decoders conforming to the Main profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 77 or constraint_set1_flag is equal to 1 and in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.3 Extended profile

Bitstreams conforming to the Extended profile shall obey the following constraints:

– Sequence parameter sets shall have direct_8x8_inference_flag equal to 1.

– The syntax elements chroma_format_idc, bit_depth_luma_minus8, bit_depth_chroma_minus8, qpprime_y_zero_transform_bypass_flag, and seq_scaling_matrix_present_flag shall not be present in sequence parameter sets.

– Picture parameter sets shall have entropy_coding_mode_flag equal to 0.

– Picture parameter sets shall have num_slice_groups_minus1 in the range of 0 to 7, inclusive.

– The syntax elements transform_8x8_mode_flag, pic_scaling_matrix_present_flag, and second_chroma_qp_index_offset shall not be present in picture parameter sets.

– The syntax element level_prefix shall not be greater than 15 (when present).

– The level constraints specified for the Extended profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the Extended profile is specified by profile_idc being equal to 88.

Decoders conforming to the Extended profile at a specified level shall be capable of decoding all bitstreams in which profile_idc is equal to 88 or constraint_set2_flag is equal to 1 and in which level_idc represents a level less than or equal to specified level.

Decoders conforming to the Extended profile at a specified level shall also be capable of decoding all bitstreams in which profile_idc is equal to 66 or constraint_set0_flag is equal to 1, in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level.

A.2.4 High profile

Bitstreams conforming to the High profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– Sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.

– Sequence parameter sets shall have bit_depth_luma_minus8 equal to 0 only.

– Sequence parameter sets shall have bit_depth_chroma_minus8 equal to 0 only.

– Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.

– The level constraints specified for the High profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High profile is specified by profile_idc being equal to 100. Decoders conforming to the High profile at a specific level shall be capable of decoding all bitstreams in which level_idc and constraint_set3_flag represents a level less than or equal to the specified level and either or both of the following conditions are true:

– profile_idc is equal to 77 or 100, or

– constraint_set1_flag is equal to 1.

A.2.5 High 10 profile

Bitstreams conforming to the High 10 profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– Sequence parameter sets shall have chroma_format_idc in the range of 0 to 1 inclusive.

– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 2 inclusive.

– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 2 inclusive.

– Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.

– The level constraints specified for the High 10 profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 10 profile is specified by profile_idc being equal to 110. Decoders conforming to the High 10 profile at a specific level shall be capable of decoding all bitstreams in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level and either or both of the following conditions are true:

– profile_idc is equal to 77, 100, or 110, or

– constraint_set1_flag is equal to 1.

A.2.6 High 4:2:2 profile

Bitstreams conforming to the High 4:2:2 profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– Sequence parameter sets shall have chroma_format_idc in the range of 0 to 2 inclusive

– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 2 inclusive.

– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 2 inclusive.

– Sequence parameter sets shall have qpprime_y_zero_transform_bypass_flag equal to 0 only.

– The level constraints specified for the High 4:2:2 profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:2:2 profile is specified by profile_idc being equal to 122. Decoders conforming to the High 4:2:2 profile at a specific level shall be capable of decoding all bitstreams in which level_idc and constraint_set3_flag represents a level less than or equal to the specified level and either or both of the following conditions are true:

– profile_idc is equal to 77, 100, 110, or 122, or

– constraint_set1_flag is equal to 1.

A.2.7 High 4:4:4 profile

Bitstreams conforming to the High 4:4:4 profile shall obey the following constraints:

– Only I, P, and B slice types may be present.

– NAL unit streams shall not contain nal_unit_type values in the range of 2 to 4, inclusive.

– Arbitrary slice order is not allowed.

– Picture parameter sets shall have num_slice_groups_minus1 equal to 0 only.

– Picture parameter sets shall have redundant_pic_cnt_present_flag equal to 0 only.

– Sequence parameter sets shall have bit_depth_luma_minus8 in the range of 0 to 4 inclusive.

– Sequence parameter sets shall have bit_depth_chroma_minus8 in the range of 0 to 4 inclusive.

– The level constraints specified for the High 4:4:4 profile in subclause A.3 shall be fulfilled.

Conformance of a bitstream to the High 4:4:4 profile is specified by profile_idc being equal to 144. Decoders conforming to the High 4:4:4 profile at a specific level shall be capable of decoding all bitstreams in which level_idc and constraint_set3_flag represent a level less than or equal to the specified level and either or both of the following conditions are true:

– profile_idc is equal to 77, 100, 110, 122, or 144, or

– constraint_set1_flag is equal to 1.

A.3 Levels

The following is specified for expressing the constraints in this Annex.

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

– Let picture n be the primary coded picture or the corresponding decoded picture of access unit n.

A.1.1 Level limits common to the Baseline, Main, and Extended profiles

Let the variable fR be derived as follows.

– If picture n is a frame, fR is set equal to 1 ÷ 172.

– Otherwise (picture n is a field), fR is set equal to 1 ÷ (172 * 2).

Bitstreams conforming to the Baseline, Main, or Extended profiles at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n (with n > 0) from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n – 1, and PicSizeInMbs is the number of macroblocks in picture n – 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n) >= Max(PicSizeInMbs \div MaxMBPS, fR)$, where MaxMBPS is the value specified in Table A-1 for picture n, and PicSizeInMbs is the number of macroblocks of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) The sum of the NumBytesInNALunit variables for access unit 0 is less than or equal to $384 * (PicSizeInMbs + MaxMBPS * (t_r(0) - t_{r,n}(0))) \div MinCR$, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture 0 and PicSizeInMbs is the number of macroblocks in picture 0.

d) The sum of the NumBytesInNALunit variables for access unit n (with n > 0) is less than or equal to $384 * MaxMBPS * (t_r(n) - t_r(n - 1)) \div MinCR$, where MaxMBPS and MinCR are the values specified in Table A-1 that apply to picture n.

e) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1

f) PicWidthInMbs <= Sqrt(MaxFS * 8)

g) FrameHeightInMbs <= Sqrt(MaxFS * 8)

h) max_dec_frame_buffering <= MaxDpbSize, where MaxDpbSize is equal to Min(1024 * MaxDPB / (PicWidthInMbs * FrameHeightInMbs * 384), 16) and MaxDPB is given in Table A-1 in units of 1024 bytes.

i) For the VCL HRD parameters, BitRate[SchedSelIdx] <= 1000 * MaxBR and CpbSize[SchedSelIdx] <= 1000 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] is given by Equation E-37 and CpbSize[SchedSelIdx] is given by Equation E-38 when vcl_hrd_parameters_present_flag is equal to 1. MaxBR and MaxCPB are specified in Table A-1 in units of 1000 bits/s and 1000 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

j) For the NAL HRD parameters, BitRate[SchedSelIdx] <= 1200 * MaxBR and CpbSize[SchedSelIdx] <= 1200 * MaxCPB for at least one value of SchedSelIdx, where BitRate[SchedSelIdx] is given by Equation E-37 and CpbSize[SchedSelIdx] is given by Equation E-38 when nal_hrd_parameters_present_flag equal to 1. MaxBR and MaxCPB are specified in Table A-1 in units of 1200 bits/s and 1200 bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1.

k) Vertical motion vector component range luma motion vectors does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1

NOTE 1 – When chroma_format_idc is equal to 1 and the current macroblock is a field macroblock, the motion vector component range for chroma motion vectors may exceed MaxVmvR in units of luma frame samples, due to the method of deriving chroma motion vectors as specified in subclause 8.4.1.4.

l) Horizontal motion vector range does not exceed the range of -2048 to 2047.75, inclusive, in units of luma samples

m) Number of motion vectors per two consecutive macroblocks in decoding order (also applying to the total from the last macroblock of a slice and the first macroblock of the next slice in decoding order, and in particular also applying to the total from the last macroblock of the last slice of a picture and the first macroblock of the first slice of the next picture in decoding order) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is the value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

n) Number of bits of macroblock_layer( ) data for any macroblock is not greater than 3200. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows.

– If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

– Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

Table A-1 specifies the limits for each level. Entries marked "-" in Table A-1 denote the absence of a corresponding limit. For purposes of comparison of level capabilities, a level shall be considered to be a lower (higher) level than some other level if the level appears nearer to the top (bottom) row of Table A-1 than the other level.

A level to which the bitstream conforms shall be indicated by the syntax elements level_idc and constraint_set3_flag as follows.

– If level_idc is equal to 11 and constraint_set3_flag is equal to 1, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 11 or constraint_set3_flag is not equal to 1), level_idc shall be set equal to a value of ten times the level number specified in Table A-1 and constraint_set3_flag shall be set equal to 0.

Table A-1 – Level limits

Level number	Max macroblock processing rate MaxMBPS (MB/s)	Max frame size MaxFS (MBs)	Max decoded picture buffer size MaxDPB (1024 bytes for 4:2:0)	Max video bit rate MaxBR (1000 bits/s, 1200 bits/s, cpbBrVclFactor bits/s, or cpbBrNalFactor bits/s)	Max CPB size MaxCPB (1000 bits, 1200 bits, cpbBrVclFactor bits, or cpbBrNalFactor bits)	Vertical MV component range MaxVmvR (luma frame samples)	Min compression ratio MinCR	Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb
1	1 485	99	148.5	64	175	[-64,+63.75]	2	-
1b	1 485	99	148.5	128	350	[-64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[-128,+127.75]	2	-
1.2	6 000	396	891.0	384	1 000	[-128,+127.75]	2	-
1.3	11 880	396	891.0	768	2 000	[-128,+127.75]	2	-
2	11 880	396	891.0	2 000	2 000	[-128,+127.75]	2	-
2.1	19 800	792	1 782.0	4 000	4 000	[-256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4 000	4 000	[-256,+255.75]	2	-
3	40 500	1 620	3 037.5	10 000	10 000	[-256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[-512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20 000	20 000	[-512,+511.75]	4	16
4	245 760	8 192	12 288.0	20 000	25 000	[-512,+511.75]	4	16
4.1	245 760	8 192	12 288.0	50 000	62 500	[-512,+511.75]	2	16
4.2	522 240	8 704	13 056.0	50 000	62 500	[-512,+511.75]	2	16
5	589 824	22 080	41 400.0	135 000	135 000	[-512,+511.75]	2	16
5.1	983 040	36 864	69 120.0	240 000	240 000	[-512,+511.75]	2	16

Levels with non-integer level numbers in Table A-1 are referred to as “intermediate levels”.

NOTE 2 – All levels have the same status, but some applications may choose to use only the integer-numbered levels.

Informative subclause A.3.2 shows the effect of these limits on frame rates for several example picture formats.

A.3.1 Level limits common to the High, High 10, High 4:2:2, and High 4:4:4 profiles

Let the variable fR be derived as follows.

– If picture n is a frame, fR is set equal to 1 ÷ 172.

– Otherwise (picture n is a field), fR is set equal to 1 ÷ (172 * 2).

Bitstreams conforming to the High, High 10, High 4:2:2, or High 4:4:4 profiles at a specified level shall obey the following constraints:

a) The nominal removal time of access unit n (with n > 0) from the CPB as specified in subclause C.1.2, satisfies the constraint that $t_{r,n}(n) - t_r(n - 1)$ is greater than or equal to Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 that applies to picture n – 1, and PicSizeInMbs is the number of macroblocks in picture n – 1.

b) The difference between consecutive output times of pictures from the DPB as specified in subclause C.2.2, satisfies the constraint that $\Delta t_{o,dpb}(n)$ >= Max(PicSizeInMbs ÷ MaxMBPS, fR), where MaxMBPS is the value specified in Table A-1 for picture n, and PicSizeInMbs is the number of macroblocks of picture n, provided that picture n is a picture that is output and is not the last picture of the bitstream that is output.

c) PicWidthInMbs * FrameHeightInMbs <= MaxFS, where MaxFS is specified in Table A-1

d) PicWidthInMbs <= Sqrt(MaxFS * 8)

e) FrameHeightInMbs <= Sqrt(MaxFS * 8)

f) max_dec_frame_buffering <= MaxDpbSize, where MaxDpbSize is equal to Min(1024 * MaxDPB / (PicWidthInMbs * FrameHeightInMbs * 384), 16) and MaxDPB is specified in Table A-1.

g) Vertical motion vector component range does not exceed MaxVmvR in units of luma frame samples, where MaxVmvR is specified in Table A-1.

h) Horizontal motion vector range does not exceed the range of -2048 to 2047.75, inclusive, in units of luma samples.

i) Number of motion vectors per two consecutive macroblocks in decoding order (also applying to the total from the last macroblock of a slice and the first macroblock of the next slice in decoding order) does not exceed MaxMvsPer2Mb, where MaxMvsPer2Mb is specified in Table A-1. The number of motion vectors for each macroblock is value of the variable MvCnt after the completion of the intra or inter prediction process for the macroblock.

j) Number of bits of macroblock_layer( ) data for any macroblock is not greater than 128 + RawMbBits. Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows.

   – If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

   – Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

Table A-1 specifies the limits for each level. Entries marked "-" in Table A-1 denote the absence of a corresponding limit.

A level to which the bitstream conforms shall be indicated by the syntax element level_idc as follows.

– If level_idc is equal to 9, the indicated level is level 1b.

– Otherwise (level_idc is not equal to 9), level_idc shall be set equal to a value of ten times the level number specified in Table A-1.

A.1.2 Profile-specific level limits

a) In bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 profiles, the removal time of access unit 0 shall satisfy the constraint that the number of slices in picture 0 is less than or equal to $(PicSizeInMbs + MaxMBPS * (t_r(0) - t_{r,n}(0))) \div SliceRate$, where SliceRate is the value specified in Table A-4 that applies to picture 0.

b) In bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 profiles, the difference between consecutive removal time of access units n and n - 1 (with n > 0) shall satisfy the constraint that the number of slices in picture n is less than or equal to $MaxMBPS * (t_r(n) - t_r(n - 1)) \div SliceRate$, where SliceRate is the value specified in Table A-4 that applies to picture n.

c) In bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 profiles, sequence parameter sets shall have direct_8x8_inference_flag equal to 1 for the levels specified in Table A-4.

NOTE 1 – direct_8x8_inference_flag is not relevant to the Baseline profile as it does not allow B slice types (specified in subclause A.2.1), and direct_8x8_inference_flag is equal to 1 for all levels of the Extended profile (specified in subclause A.2.3).

d) In bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 or Extended profiles, sequence parameter sets shall have frame_mbs_only_flag equal to 1 for the levels specified in Table A-4 for the Main, High, High 10, High 4:2:2, and High 4:4:4 profiles and in Table A-5 for the Extended profile.

NOTE 2 – frame_mbs_only_flag is equal to 1 for all levels of the Baseline profile (specified in subclause A.2.1).

e) In bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 or Extended profiles, the value of sub_mb_type in B macroblocks shall not be equal to B_Bi_8x4, B_Bi_4x8, or B_Bi_4x4 for the levels in which MinLumaBiPredSize is shown as 8x8 in Table A-4 for the Main, High, High 10, High 4:2:2, and High 4:4:4 profiles and in Table A-5 for the Extended profile.

f) In bitstreams conforming to the Baseline and Extended profiles, ($xInt_{max} - xInt_{min} + 6$) * ($yInt_{max} - yInt_{min} + 6$) <= MaxSubMbRectSize in macroblocks coded with mb_type equal to P_8x8, P_8x8ref0 or B_8x8 for all invocations of the process specified in subclause 8.4.2.2.1 used to generate the predicted luma sample array for a single reference picture list (reference picture list 0 or reference picture list 1) for each 8x8 sub-macroblock, where NumSubMbPart(sub_mb_type) > 1, where MaxSubMbRectSize is specified in Table A-3 for the Baseline profile and in Table A-5 for the Extended profile and

– $xInt_{min}$ is the minimum value of $xInt_L$ among all luma sample predictions for the sub-macroblock

– $xInt_{max}$ is the maximum value of $xInt_L$ among all luma sample predictions for the sub-macroblock

– $yInt_{min}$ is the minimum value of $yInt_L$ among all luma sample predictions for the sub-macroblock

– $yInt_{max}$ is the maximum value of $yInt_L$ among all luma sample predictions for the sub-macroblock

g) In bitstreams conforming to the High, High 10, High 4:2:2, or High 4:4:4 profile, for the VCL HRD parameters, BitRate[SchedSelIdx] <= cpbBrVclFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrVclFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrVclFactor is specified in Table A-2, BitRate[SchedSelIdx] is specified by Equation E-37 and CpbSize[SchedSelIdx] is specified by Equation E-38 when vcl_hrd_parameters_present_flag is equal to 1. MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrVclFactor bits/s and cpbBrVclFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1, inclusive.

h) In bitstreams conforming to the High, High 10, High 4:2:2, or High 4:4:4 profile, for the NAL HRD parameters, BitRate[SchedSelIdx] <= cpbBrNalFactor * MaxBR and CpbSize[SchedSelIdx] <= cpbBrNalFactor * MaxCPB for at least one value of SchedSelIdx, where cpbBrNalFactor is specified in Table A-2, BitRate[SchedSelIdx] is specified by Equation E-37 and CpbSize[SchedSelIdx] is specified by Equation E-38 when nal_hrd_parameters_present_flag equal to 1. MaxBR and MaxCPB are specified in Table A-1 in units of cpbBrNalFactor bits/s and cpbBrNalFactor bits, respectively. The bitstream shall satisfy these conditions for at least one value of SchedSelIdx in the range 0 to cpb_cnt_minus1.

Table A-2 – Specification of cpbBrVclFactor and cpbBrNalFactor

Profile	cpbBrVclFactor	cpbBrNalFactor
High	1 250	1 500
High 10	3 000	3 600
High 4:2:2	4 000	4 800
High 4:4:4	4 000	4 800

A.3.1.1 Baseline profile limits

Table A-3 specifies limits for each level that are specific to bitstreams conforming to the Baseline profile. Entries marked "-" in Table A-3 denote the absence of a corresponding limit.

Table A-3 – Baseline profile level limits

Level number	MaxSubMbRectSize
1	576
1b	576
1.1	576
1.2	576
1.3	576
2	576
2.1	576
2.2	576
3	576
3.1	-
3.2	-
4	-
4.1	-
4.2	-
5	-
5.1	-

A.3.1.2 Main, High, High 10, High 4:2:2, or High 4:4:4 profile limits

Table A-4 specifies limits for each level that are specific to bitstreams conforming to the Main, High, High 10, High 4:2:2, or High 4:4:4 profiles. Entries marked "-" in Table A-4 denote the absence of a corresponding limit.

Table A-4 – Main, High, High 10, High 4:2:2, or High 4:4:4 profile level limits

Level_number	SliceRate	MinLumaBiPredSize	direct_8x8_inference_flag	frame_mbs_only_flag
1	-	-	-	1
1b	-	-	-	1
1.1	-	-	-	1
1.2	-	-	-	1
1.3	-	-	-	1
2	-	-	-	1
2.1	-	-	-	-
2.2	-	-	-	-
3	22	-	1	-
3.1	60	8x8	1	-
3.2	60	8x8	1	-
4	60	8x8	1	-
4.1	24	8x8	1	-
4.2	24	8x8	1	1
5	24	8x8	1	1
5.1	24	8x8	1	1

A.3.1.3 Extended Profile Limits

Table A-5 specifies limits for each level that are specific to bitstreams conforming to the Extended profile. Entries marked "-" in Table A-5 denote the absence of a corresponding limit.

Table A-5 – Extended profile level limits

Level number	MaxSubMbRectSize	MinLumaBiPredSize	frame_mbs_only_flag
1	576	-	1
1b	576	-	1
1.1	576	-	1
1.2	576	-	1
1.3	576	-	1
2	576	-	1
2.1	576	-	-
2.2	576	-	-
3	576	-	-
3.1	-	8x8	-
3.2	-	8x8	-
4	-	8x8	-
4.1	-	8x8	-
4.2	-	8x8	1
5	-	8x8	1
5.1	-	8x8	1

A.3.2 Effect of level limits on frame rate (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Table A-6 – Maximum frame rates (frames per second) for some example frame sizes

Level:					**1**	**1b**	**1.1**	**1.2**	**1.3**	**2**	**2.1**
Max frame size (macroblocks):					**99**	**99**	**396**	**396**	**396**	**396**	**792**
Max macroblocks/second:					**1 485**	**1 485**	**3 000**	**6 000**	**11 880**	**11 880**	**19 800**
Max frame size (samples):					25 344	25 344	101 376	101 376	101 376	101 376	202 752
Max samples/second:					380 160	380 160	768 000	1 536 000	3 041 280	3 041 280	5 068 800
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	30.9	30.9	62.5	125.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	15.0	15.0	30.3	60.6	120.0	120.0	172.0
QVGA	**320**	**240**	300	76 800	-	-	10.0	20.0	39.6	39.6	66.0
525 SIF	**352**	**240**	330	84 480	-	-	9.1	18.2	36.0	36.0	60.0
CIF	**352**	**288**	396	101 376	-	-	7.6	15.2	30.0	30.0	50.0
525 HHR	**352**	**480**	660	168 960	-	-	-	-	-	-	30.0
625 HHR	**352**	**576**	792	202 752	-	-	-	-	-	-	25.0
VGA	**640**	**480**	1 200	307 200	-	-	-	-	-	-	-
525 4SIF	**704**	**480**	1 320	337 920	-	-	-	-	-	-	-
525 SD	**720**	**480**	1 350	345 600	-	-	-	-	-	-	-
4CIF	**704**	**576**	1 584	405 504	-	-	-	-	-	-	-
625 SD	**720**	**576**	1 620	414 720	-	-	-	-	-	-	-
SVGA	**800**	**600**	1 900	486 400	-	-	-	-	-	-	-
XGA	**1024**	**768**	3 072	786 432	-	-	-	-	-	-	-
720p HD	**1280**	**720**	3 600	921 600	-	-	-	-	-	-	-
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	-	-	-	-
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	-	-	-	-
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	-	-	-
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	-	-	-
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	-	-	-
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	-	-	-
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	-	-	-
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	-
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-6 (continued) – Maximum frame rates (frames per second) for some example frame sizes

Level:					**2.2**	**3**	**3.1**	**3.2**	**4**	**4.1**	**4.2**
Max frame size (macroblocks):					**1 620**	**1 620**	**3 600**	**5 120**	**8 192**	**8 192**	**8 704**
Max macroblocks/second:					**20 250**	**40 500**	**108 000**	**216 000**	**245 760**	**245 760**	**522 240**
Max frame size (samples):					414 720	414 720	921 600	1 310 720	2 097 152	2 097 152	2 228 224
Max samples/second:					5 184 000	10 368 000	27 648 000	55 296 000	62 914 560	62 914 560	133 693 440
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**							
SQCIF	**128**	**96**	48	12 288	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA	**320**	**240**	300	76 800	67.5	135.0	172.0	172.0	172.0	172.0	172.0
525 SIF	**352**	**240**	330	84 480	61.4	122.7	172.0	172.0	172.0	172.0	172.0
CIF	**352**	**288**	396	101 376	51.1	102.3	172.0	172.0	172.0	172.0	172.0
525 HHR	**352**	**480**	660	168 960	30.7	61.4	163.6	172.0	172.0	172.0	172.0
625 HHR	**352**	**576**	792	202 752	25.6	51.1	136.4	172.0	172.0	172.0	172.0
VGA	**640**	**480**	1 200	307 200	16.9	33.8	90.0	172.0	172.0	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	15.3	30.7	81.8	163.6	172.0	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	15.0	30.0	80.0	160.0	172.0	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	12.8	25.6	68.2	136.4	155.2	155.2	172.0
625 SD	**720**	**576**	1 620	414 720	12.5	25.0	66.7	133.3	151.7	151.7	172.0
SVGA	**800**	**600**	1 900	486 400	-	-	56.8	113.7	129.3	129.3	172.0
XGA	**1024**	**768**	3 072	786 432	-	-	35.2	70.3	80.0	80.0	172.0
720p HD	**1280**	**720**	3 600	921 600	-	-	30.0	60.0	68.3	68.3	145.1
4VGA	**1280**	**960**	4 800	1 228 800	-	-	-	45.0	51.2	51.2	108.8
SXGA	**1280**	**1024**	5 120	1 310 720	-	-	-	42.2	48.0	48.0	102.0
525 16SIF	**1408**	**960**	5 280	1 351 680	-	-	-	-	46.5	46.5	98.9
16CIF	**1408**	**1152**	6 336	1 622 016	-	-	-	-	38.8	38.8	82.4
4SVGA	**1600**	**1200**	7 500	1 920 000	-	-	-	-	32.8	32.8	69.6
1080 HD	**1920**	**1088**	8 160	2 088 960	-	-	-	-	30.1	30.1	64.0
2Kx1K	**2048**	**1024**	8 192	2 097 152	-	-	-	-	30.0	30.0	63.8
2Kx1080	**2048**	**1088**	8 704	2 228 224	-	-	-	-	-	-	60.0
4XGA	**2048**	**1536**	12 288	3 145 728	-	-	-	-	-	-	-
16VGA	**2560**	**1920**	19 200	4 915 200	-	-	-	-	-	-	-
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	-	-	-	-	-	-	-
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	-	-	-	-	-	-	-
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	-	-	-	-	-	-
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	-	-	-	-	-	-

Table A-6 (concluded) – Maximum frame rates (frames per second) for some example frame sizes

Level:					**5**	**5.1**
Max frame size (macroblocks):					**22 080**	**36 864**
Max macroblocks/second:					**589 824**	**983 040**
Max frame size (samples):					5 652 480	9 437 184
Max samples/second:					150 994 944	251 658 240
Format	**Luma Width**	**Luma Height**	**MBs Total**	**Luma Samples**		
SQCIF	**128**	**96**	48	12 288	172.0	172.0
QCIF	**176**	**144**	99	25 344	172.0	172.0
QVGA	**320**	**240**	300	76 800	172.0	172.0
525 SIF	**352**	**240**	330	84 480	172.0	172.0
CIF	**352**	**288**	396	101 376	172.0	172.0
525 HHR	**352**	**480**	660	168 960	172.0	172.0
625 HHR	**352**	**576**	792	202 752	172.0	172.0
VGA	**640**	**480**	1 200	307 200	172.0	172.0
525 4SIF	**704**	**480**	1 320	337 920	172.0	172.0
525 SD	**720**	**480**	1 350	345 600	172.0	172.0
4CIF	**704**	**576**	1 584	405 504	172.0	172.0
625 SD	**720**	**576**	1 620	414 720	172.0	172.0
SVGA	**800**	**600**	1 900	486 400	172.0	172.0
XGA	**1024**	**768**	3 072	786 432	172.0	172.0
720p HD	**1280**	**720**	3 600	921 600	163.8	172.0
4VGA	**1280**	**960**	4 800	1 228 800	122.9	172.0
SXGA	**1280**	**1024**	5 120	1 310 720	115.2	172.0
525 16SIF	**1408**	**960**	5 280	1 351 680	111.7	172.0
16CIF	**1408**	**1152**	6 336	1 622 016	93.1	155.2
4SVGA	**1600**	**1200**	7 500	1 920 000	78.6	131.1
1080 HD	**1920**	**1088**	8 160	2 088 960	72.3	120.5
2Kx1K	**2048**	**1024**	8 192	2 097 152	72.0	120.0
2Kx1080	**2048**	**1088**	8 704	2 228 224	67.8	112.9
4XGA	**2048**	**1536**	12 288	3 145 728	48.0	80.0
16VGA	**2560**	**1920**	19 200	4 915 200	30.7	51.2
3616x1536 (2.35:1)	**3616**	**1536**	21 696	5 554 176	27.2	45.3
3672x1536 (2.39:1)	**3680**	**1536**	22 080	5 652 480	26.7	44.5
4Kx2K	**4096**	**2048**	32 768	8 388 608	-	30.0
4096x2304 (16:9)	**4096**	**2304**	36 864	9 437 184	-	26.7

The following should be noted.

– This Recommendation | International Standard is a variable-frame-size specification. The specific frame sizes in Table A-6 are illustrative examples only.

– As used in Table A-6, "525" refers to typical use for environments using 525 analogue scan lines (of which approximately 480 lines contain the visible picture region), and "625" refers to environments using 625 analogue scan lines (of which approximately 576 lines contain the visible picture region).

– XGA is also known as (aka) XVGA, 4SVGA aka UXGA, 16XGA aka 4Kx3K, CIF aka 625 SIF, 625 HHR aka 2CIF aka half 625 D-1, aka half 625 ITU-R BT.601, 525 SD aka 525 D-1 aka 525 ITU-R BT.601, 625 SD aka 625 D-1 aka 625 ITU-R BT.601.

– Frame rates given are correct for progressive scan modes. The frame rates are also correct for interlaced video coding for the cases of frame height divisible by 32.

BAnnex B
Byte stream format

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics of a byte stream format specified for use by applications that deliver some or all of the NAL unit stream as an ordered stream of bytes or bits within which the locations of NAL unit boundaries need to be identifiable from patterns in the data, such as ITU-T Rec. H.222.0 | ISO/IEC 13818-1 systems or ITU-T Rec. H.320 systems. For bit-oriented delivery, the bit order for the byte stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The byte stream format consists of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure contains one start code prefix followed by one nal_unit(NumBytesInNALunit) syntax structure. It may (and under some circumstances, it shall) also contain an additional zero_byte syntax element. It may also contain one or more additional trailing_zero_8bits syntax elements. When it is the first byte stream NAL unit in the bitstream, it may also contain one or more additional leading_zero_8bits syntax elements.

B.1 Byte stream NAL unit syntax and semantics

B.1.1 Byte stream NAL unit syntax

byte_stream_nal_unit(NumBytesInNALunit) {	C	Descriptor
while(next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
leading_zero_8bits /* equal to 0x00 */		f(8)
if(next_bits(24) != 0x000001)		
zero_byte /* equal to 0x00 */		f(8)
start_code_prefix_one_3bytes /* equal to 0x000001 */		f(24)
nal_unit(NumBytesInNALunit)		
while(more_data_in_byte_stream( ) && next_bits(24) != 0x000001 && next_bits(32) != 0x00000001)		
trailing_zero_8bits /* equal to 0x00 */		f(8)
}		

B.1.2 Byte stream NAL unit semantics

The order of byte stream NAL units in the byte stream shall follow the decoding order of the NAL units contained in the byte stream NAL units (see subclause 7.4.1.2). The content of each byte stream NAL unit is associated with the same access unit as the NAL unit contained in the byte stream NAL unit (see subclause 7.4.1.2.3).

leading_zero_8bits is a byte equal to 0x00.

NOTE – The leading_zero_8bits syntax element can only be present in the first byte stream NAL unit of the bitstream, because (as shown in the syntax diagram of subclause B.1.1) any bytes equal to 0x00 that follow a NAL unit syntax structure and precede the four-byte sequence 0x00000001 (which is to be interpreted as a zero_byte followed by a start_code_prefix_one_3bytes) will be considered to be trailing_zero_8bits syntax elements that are part of the preceding byte stream NAL unit.

zero_byte is a single byte equal to 0x00.

When any of the following conditions are fulfilled, the zero_byte syntax element shall be present.

– the nal_unit_type within the nal_unit( ) is equal to 7 (sequence parameter set) or 8 (picture parameter set)

– the byte stream NAL unit syntax structure contains the first NAL unit of an access unit in decoding order, as specified by subclause 7.4.1.2.3.

start_code_prefix_one_3bytes is a fixed-value sequence of 3 bytes equal to 0x000001. This syntax element is called a start code prefix.

trailing_zero_8bits is a byte equal to 0x00.

B.2 Byte stream NAL unit decoding process

Input to this process consists of an ordered stream of bytes consisting of a sequence of byte stream NAL unit syntax structures.

Output of this process consists of a sequence of NAL unit syntax structures.

At the beginning of the decoding process, the decoder initialises its current position in the byte stream to the beginning of the byte stream. It then extracts and discards each leading_zero_8bits syntax element (if present), moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next four bytes in the bitstream form the four-byte sequence 0x00000001.

The decoder then performs the following step-wise process repeatedly to extract and decode each NAL unit syntax structure in the byte stream until the end of the byte stream has been encountered (as determined by unspecified means) and the last NAL unit in the byte stream has been decoded:

1. When the next four bytes in the bitstream form the four-byte sequence 0x00000001, the next byte in the byte stream (which is a zero_byte syntax element) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this discarded byte.
2. The next three-byte sequence in the byte stream (which is a start_code_prefix_one_3bytes) is extracted and discarded and the current position in the byte stream is set equal to the position of the byte following this three-byte sequence.
3. NumBytesInNALunit is set equal to the number of bytes starting with the byte at the current position in the byte stream up to and including the last byte that precedes the location of any of the following conditions:
    a. A subsequent byte-aligned three-byte sequence equal to 0x000000, or
    b. A subsequent byte-aligned three-byte sequence equal to 0x000001, or
    c. The end of the byte stream, as determined by unspecified means.
4. NumBytesInNALunit bytes are removed from the bitstream and the current position in the byte stream is advanced by NumBytesInNALunit bytes. This sequence of bytes is nal_unit(NumBytesInNALunit) and is decoded using the NAL unit decoding process.
5. When the current position in the byte stream is not at the end of the byte stream (as determined by unspecified means) and the next bytes in the byte stream do not start with a three-byte sequence equal to 0x000001 and the next bytes in the byte stream do not start with a four byte sequence equal to 0x00000001, the decoder extracts and discards each trailing_zero_8bits syntax element, moving the current position in the byte stream forward one byte at a time, until the current position in the byte stream is such that the next bytes in the byte stream form the four-byte sequence 0x00000001 or the end of the byte stream has been encountered (as determined by unspecified means).

B.3 Decoder byte-alignment recovery (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

Many applications provide data to a decoder in a manner that is inherently byte aligned, and thus have no need for the bit-oriented byte alignment detection procedure described in this subclause.

A decoder is said to have byte-alignment with a bitstream when the decoder is able to determine whether or not the positions of data in the bitstream are byte-aligned. When a decoder does not have byte alignment with the encoder's byte stream, the decoder may examine the incoming bitstream for the binary pattern '00000000 00000000 00000000 00000001' (31 consecutive bits equal to 0 followed by a bit equal to 1). The bit immediately following this pattern is the first bit of an aligned byte following a start code prefix. Upon detecting this pattern, the decoder will be byte aligned with the encoder and positioned at the start of a NAL unit in the byte stream.

Once byte aligned with the encoder, the decoder can examine the incoming byte stream for subsequent three-byte sequences 0x000001 and 0x000003.

When the three-byte sequence 0x000001 is detected, this is a start code prefix.

When the three-byte sequence 0x000003 is detected, the third byte (0x03) is an emulation_prevention_three_byte to be discarded as specified in subclause 7.4.1.

When an error in the bitstream syntax is detected (e.g., a non-zero value of the forbidden_zero_bit or one of the three-byte or four-byte sequences that are prohibited in subclause 7.4.1), the decoder may consider the detected condition as an indication that byte alignment may have been lost and may discard all bitstream data until the detection of byte alignment at a later position in the bitstream as described in this subclause.

CAnnex C
Hypothetical reference decoder

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies the hypothetical reference decoder (HRD) and its use to check bitstream and decoder conformance.

Two types of bitstreams are subject to HRD conformance checking for this Recommendation | International Standard. The first such type of bitstream, called Type I bitstream, is a NAL unit stream containing only the VCL NAL units and filler data NAL units for all access units in the bitstream. The second type of bitstream, called a Type II bitstream, contains, in addition to the VCL NAL units and filler data NAL units for all access units in the bitstream, at least one of the following.

– additional non-VCL NAL units other than filler data NAL units

– all leading_zero_8bits, zero_byte, start_code_prefix_one_3bytes, and trailing_zero_8bits syntax elements that form a byte stream from the NAL unit stream (as specified in Annex B)

Figure C-1 shows the types of bitstream conformance points checked by the HRD.

Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks

The syntax elements of non-VCL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic subclauses of clause 7 and Annexes D and E.

Two types of HRD parameter sets are used. The HRD parameter sets are signalled through video usability information as specified in subclauses E.1 and E.2, which is part of the sequence parameters set syntax structure.

In order to check conformance of a bitstream using the HRD, all sequence parameter sets and picture parameters sets referred to in the VCL NAL units, and corresponding buffering period and picture timing SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-VCL NAL units), or by other means not specified in this Recommendation | International Standard.

In Annexes C, D and E, the specification for "presence" of non-VCL NAL units is also satisfied when those NAL units (or just some of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 1 – As an example, synchronization of a non-VCL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non-VCL NAL unit would have been present in the bitstream, had the encoder decided to convey it in the bitstream.

When the content of a non-VCL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-VCL NAL unit is not required to use the same syntax specified in this annex.

NOTE 2 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data is supplied by some other means not specified in this Recommendation | International Standard.

The HRD contains a coded picture buffer (CPB), an instantaneous decoding process, a decoded picture buffer (DPB), and output cropping as shown in Figure C-2.

Figure C-2 – HRD buffer model

The CPB size (number of bits) is CpbSize[SchedSelIdx]. The DPB size (number of frame buffers) is Max(1, max_dec_frame_buffering).

The HRD operates as follows. Data associated with access units that flow into the CPB according to a specified arrival schedule are delivered by the HSS. The data associated with each access unit are removed and decoded instantaneously by the instantaneous decoding process at CPB removal times. Each decoded picture is placed in the DPB at its CPB removal time unless it is output at its CPB removal time and is a non-reference picture. When a picture is placed in the DPB it is removed from the DPB at the later of the DPB output time or the time that it is marked as "unused for reference".

The operation of the CPB is specified in subclause C.1. The instantaneous decoder operation is specified in clauses 8 and 9. The operation of the DPB is specified in subclause C.2. The output cropping is specified in subclause C.2.2.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in subclauses E.1.1, E.1.2, E.2.1 and E.2.2. The HRD is initialised as specified by the buffering period SEI message as specified in subclauses D.1.1 and D.2.1. The removal timing of access units from the CPB and output timing from the DPB are specified in the picture timing SEI message as specified in subclauses D.1.2 and D.2.2. All timing information relating to a specific access unit shall arrive prior to the CPB removal time of the access unit.

The HRD is used to check conformance of bitstreams and decoders as specified in subclauses C.3 and C.4, respectively.

NOTE 3 – While conformance is guaranteed under the assumption that all frame-rates and clocks used to generate the bitstream match exactly the values signalled in the bitstream, in a real system each of these may vary from the signalled or specified value.

All the arithmetic in this annex is done with real values, so that no rounding errors can propagate. For example, the number of bits in a CPB just prior to or after removal of an access unit is not necessarily an integer.

The variable t_c is derived as follows and is called a clock tick.

$$t_c = \text{num_units_in_tick} \div \text{time_scale} \qquad \text{(C-1)}$$

The following is specified for expressing the constraints in this Annex.

– Let access unit n be the n-th access unit in decoding order with the first access unit being access unit 0.

– Let picture n be the primary coded picture or the decoded primary picture of access unit n.

C.1 Operation of coded picture buffer (CPB)

The specifications in this subclause apply independently to each set of CPB parameters that is present and to both the Type I and Type II conformance points shown in Figure C-1.

C.1.1 Timing of bitstream arrival

The HRD may be initialised at any one of the buffering period SEI messages. Prior to initialisation, the CPB is empty.

NOTE – After initialisation, the HRD is not initialised again by subsequent buffering period SEI messages.

Each access unit is referred to as access unit n, where the number n identifies the particular access unit. The access unit that is associated with the buffering period SEI message that initializes the CPB is referred to as access unit 0. The value of n is incremented by 1 for each subsequent access unit in decoding order.

The time at which the first bit of access unit n begins to enter the CPB is referred to as the initial arrival time $t_{ai}(n)$.

The initial arrival time of access units is derived as follows.

– If the access unit is access unit 0, $t_{ai}(0) = 0$,

– Otherwise (the access unit is access unit n with n > 0), the following applies.

  – If cbr_flag[SchedSelIdx] is equal to 1, the initial arrival time for access unit n, is equal to the final arrival time (which is derived below) of access unit n - 1, i.e.

$$t_{ai}(n) = t_{af}(n - 1) \quad \text{(C-2)}$$

  – Otherwise (cbr_flag[SchedSelIdx] is equal to 0), the initial arrival time for access unit n is derived by

$$t_{ai}(n) = \mathrm{Max}(t_{af}(n - 1), t_{ai,earliest}(n)) \quad \text{(C-3)}$$

where $t_{ai,earliest}(n)$ is derived as follows

  – If access unit n is not the first access unit of a subsequent buffering period, $t_{ai,earliest}(n)$ is derived as

$$t_{ai,earliest}(n) = t_{r,n}(n) - (\text{initial_cpb_removal_delay[SchedSelIdx]} + \text{initial_cpb_removal_delay_offset[SchedSelIdx]}) \div 90000 \quad \text{(C-4)}$$

with $t_{r,n}(n)$ being the nominal removal time of access unit n from the CPB as specified in subclause C.1.2 and initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] being specified in the previous buffering period SEI message.

  – Otherwise (access unit n is the first access unit of a subsequent buffering period), $t_{ai,earliest}(n)$ is derived as

$$t_{ai,earliest}(n) = t_{r,n}(n) - (\text{initial_cpb_removal_delay[SchedSelIdx]} \div 90000) \quad \text{(C-5)}$$

with initial_cpb_removal_delay[SchedSelIdx] being specified in the buffering period SEI message associated with access unit n.

The final arrival time for access unit n is derived by

$$t_{af}(n) = t_{ai}(n) + b(n) \div \text{BitRate[SchedSelIdx]} \quad \text{(C-6)}$$

where b(n) is the size in bits of access unit n, counting the bits of the VCL NAL units and the filler data NAL units for the Type I conformance point or all bits of the Type II bitstream for the Type II conformance point, where the Type I and Type II conformance points are as shown in Figure C-1.

The values of SchedSelIdx, BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] are constrained as follows.

– If access unit n and access unit n - 1 are part of different coded video sequences and the content of the active sequence parameter sets of the two coded video sequences differ, the HSS selects a value SchedSelIdx1 of SchedSelIdx from among the values of SchedSelIdx provided for the coded video sequence containing access unit n that results in a BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] for the second of the two coded video sequences (which contains access unit n). The value of BitRate[SchedSelIdx1] or CpbSize[SchedSelIdx1] may differ from the value of BitRate[SchedSelIdx0] or CpbSize[SchedSelIdx0] for the value SchedSelIdx0 of SchedSelIdx that was in use for the coded video sequence containing access unit n - 1.

– Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[SchedSelIdx] and CpbSize[SchedSelIdx].

When the HSS selects values of BitRate[SchedSelIdx] or CpbSize[SchedSelIdx] that differ from those of the previous access unit, the following applies.

– the variable BitRate[SchedSelIdx] comes into effect at time $t_{ai}(n)$

– the variable CpbSize[SchedSelIdx] comes into effect as follows.

  – If the new value of CpbSize[SchedSelIdx] exceeds the old CPB size, it comes into effect at time $t_{ai}(n)$,

  – Otherwise, the new value of CpbSize[SchedSelIdx] comes into effect at the time $t_r(n)$.

C.1.2 Timing of coded picture removal

For access unit 0, the nominal removal time of the access unit from the CPB is specified by

$$t_{r,n}(0) = \text{initial_cpb_removal_delay[SchedSelIdx]} \div 90000 \qquad \text{(C-7)}$$

For the first access unit of a buffering period that does not initialise the HRD, the nominal removal time of the access unit from the CPB is specified by

$$t_{r,n}(n) = t_{r,n}(n_b) + t_c * \text{cpb_removal_delay(n)} \qquad \text{(C-8)}$$

where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the previous buffering period and cpb_removal_delay(n) is the value of cpb_removal_delay specified in the picture timing SEI message associated with access unit n.

When an access unit n is the first access unit of a buffering period, n_b is set equal to n at the removal time of access unit n.

The nominal removal time $t_{r,n}(n)$ of an access unit n that is not the first access unit of a buffering period is given by

$$t_{r,n}(n) = t_{r,n}(n_b) + t_c * \text{cpb_removal_delay(n)} \qquad \text{(C-9)}$$

where $t_{r,n}(n_b)$ is the nominal removal time of the first access unit of the current buffering period and cpb_removal_delay(n) is the value of cpb_removal_delay specified in the picture timing SEI message associated with access unit n.

The removal time of access unit n is specified as follows.

– If low_delay_hrd_flag is equal to 0 or $t_{r,n}(n) >= t_{af}(n)$, the removal time of access unit n is specified by

$$t_r(n) = t_{r,n}(n) \qquad \text{(C-10)}$$

– Otherwise (low_delay_hrd_flag is equal to 1 and $t_{r,n}(n) < t_{af}(n)$), the removal time of access unit n is specified by

$$t_r(n) = t_{r,n}(n) + t_c * \text{Ceil}((t_{af}(n) - t_{r,n}(n)) \div t_c) \qquad \text{(C-11)}$$

NOTE – The latter case indicates that the size of access unit n, b(n), is so large that it prevents removal at the nominal removal time.

C.2 Operation of the decoded picture buffer (DPB)

The decoded picture buffer contains frame buffers. Each of the frame buffers may contain a decoded frame, a decoded complementary field pair or a single (non-paired) decoded field that are marked as "used for reference" (reference pictures) or are held for future output (reordered or delayed pictures). Prior to initialisation, the DPB is empty (the DPB fullness is set to zero). The following steps of the subclauses of this subclause all happen instantaneously at $t_r(n)$ and in the sequence listed.

C.2.1 Decoding of gaps in frame_num and storage of "non-existing" frames

If applicable, gaps in frame_num are detected by the decoding process and the generated frames are marked and inserted into the DPB as specified below.

Gaps in frame_num are detected by the decoding process and the generated frames are marked as specified in subclause 8.2.5.2.

After the marking of each generated frame, each picture m marked by the “sliding window” process as “unused for reference” is removed from the DPB when it is also marked as "non-existing" or its DPB output time is less than or equal to the CPB removal time of the current picture n; i.e., $t_{o,dpb}(m) <= t_r(n)$. When a frame or the last field in a frame buffer is removed from the DPB, the DPB fullness is decremented by one. The “non-existing” generated frame is inserted into the DPB and the DPB fullness is incremented by one.

C.2.2 Picture decoding and output

Picture n is decoded and its DPB output time $t_{o,dpb}(n)$ is derived by

$$t_{o,dpb}(n) = t_r(n) + t_c * \text{dpb_output_delay}(n) \qquad \text{(C-12)}$$

The output of the current picture is specified as follows.

– If $t_{o,dpb}(n) = t_r(n)$, the current picture is output.

NOTE – When the current picture is a reference picture it will be stored in the DPB.

– Otherwise ($t_{o,dpb}(n) > t_r(n)$), the current picture is output later and will be stored in the DPB (as specified in subclause C.2.4) and is output at time $t_{o,dpb}(n)$ unless indicated not to be output by the decoding or inference of no_output_of_prior_pics_flag equal to 1 at a time that precedes $t_{o,dpb}(n)$.

The output picture shall be cropped, using the cropping rectangle specified in the sequence parameter set for the sequence.

When picture n is a picture that is output and is not the last picture of the bitstream that is output, the value of $\Delta t_{o,dpb}(n)$ is defined as:

$$\Delta t_{o,dpb}(n) = t_{o,dpb}(n_n) - t_{o,dpb}(n) \qquad \text{(C-13)}$$

where n_n indicates the picture that follows after picture n in output order.

The decoded picture is temporarily stored (not in the DPB).

C.2.3 Removal of pictures from the DPB before possible insertion of the current picture

The removal of pictures from the DPB before possible insertion of the current picture proceeds as follows.

– If the decoded picture is an IDR picture the following applies.

  – All reference pictures in the DPB are marked as "unused for reference" as specified in subclause 8.2.5.1.

  – When the IDR picture is not the first IDR picture decoded and the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the sequence parameter set that was active for the preceding sequence, respectively,

no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

NOTE – Decoder implementations should try to handle frame or DPB size changes more gracefully than the HRD in regard to changes in PicWidthInMbs or FrameHeightInMbs.

- When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all frame buffers in the DPB are emptied without output of the pictures they contain, and DPB fullness is set to 0.

- Otherwise (the decoded picture is not an IDR picture), the following applies.
  - If the slice header of the current picture includes memory_management_control_operation equal to 5, all reference pictures in the DPB are marked as "unused for reference".
  - Otherwise (the slice header of the current picture does not include memory_management_control_operation equal to 5), the decoded reference picture marking process specified in subclause 8.2.5 is invoked.

All pictures m in the DPB, for which all of the following conditions are true, are removed from the DPB.

- picture m is marked as “unused for reference” or picture m is a non-reference picture. When a picture is a reference frame, it is considered to be marked as "unused for reference" only when both of its fields have been marked as "unused for reference".
- picture m is marked as "non-existing" or its DPB output time is less than or equal to the CPB removal time of the current picture n; i.e., $t_{o,dpb}(m) <= t_r(n)$

When a frame or the last field in a frame buffer is removed from the DPB, the DPB fullness is decremented by one.

C.2.4 Current decoded picture marking and storage

C.2.4.1 Marking and storage of a reference decoded picture into the DPB

When the current picture is a reference picture it is stored in the DPB as follows.

- If the current decoded picture is a second field (in decoding order) of a complementary reference field pair, and the first field of the pair is still in the DPB, the current decoded picture is stored in the same frame buffer as the first field of the pair.
- Otherwise, the current decoded picture is stored in an empty frame buffer, and the DPB fullness is incremented by one.

C.2.4.2 Storage of a non-reference picture into the DPB

When the current picture is a non-reference picture and current picture n has $t_{o,dpb}(n) > t_r(n)$, it is stored in the DPB as follows.

- If the current decoded picture is a second field (in decoding order) of a complementary non-reference field pair, and the first field of the pair is still in the DPB, the current decoded picture is stored in the same frame buffer as the first field of the pair.
- Otherwise, the current decoded picture is stored in an empty frame buffer, and the DPB fullness is incremented by one.

C.3 Bitstream conformance

A bitstream of coded data conforming to this Recommendation | International Standard fulfils the following requirements.

The bitstream is constructed according to the syntax, semantics, and constraints specified in this Recommendation | International Standard outside of this Annex.

The bitstream is tested by the HRD as specified below:

For Type I bitstreams, the number of tests carried out is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the vcl_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination specified by hrd_parameters( ) following the vcl_hrd_parameters_present_flag. Each of these tests is conducted at the Type I conformance point shown in Figure C-1.

For Type II bitstreams there are two sets of tests. The number of tests of the first set is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the vcl_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination. Each of these tests is conducted at the Type I conformance point shown in Figure C-1. For these tests, only VCL and filler data NAL units are counted for the input bit rate and CPB storage.

The number of tests of the second set, for Type II bitstreams, is equal to cpb_cnt_minus1 + 1 where cpb_cnt_minus1 is either the syntax element of hrd_parameters( ) following the nal_hrd_parameters_present_flag or is determined by the application by other means not specified in this Recommendation | International Standard. One test is carried out for each bit rate and CPB size combination specified by hrd_parameters( ) following the nal_hrd_parameters_present_flag. Each of these tests is conducted at the Type II conformance point shown in Figure C-1. For these tests, all NAL units (of a Type II NAL unit stream) or all bytes (of a byte stream) are counted for the input bit rate and CPB storage.

NOTE 1 – NAL HRD parameters established by a value of SchedSelIdx for the Type II conformance point shown in Figure C-1 are sufficient to also establish VCL HRD conformance for the Type I conformance point shown in Figure C-1 for the same values of initial_cpb_removal_delay[SchedSelIdx], BitRate[SchedSelIdx], and CpbSize[SchedSelIdx] for the VBR case (cbr_flag[SchedSelIdx] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CPB is allowed to become empty and stay empty until the time a next picture is scheduled to begin to arrive. For example, when NAL HRD parameters are provided for the Type II conformance point that not only fall within the bounds set for NAL HRD parameters for profile conformance in item j of subclause A.1.1 or item i of subclause A.1.2 (depending on the profile in use) but also fall within the bounds set for VCL HRD parameters for profile conformance in item i of subclause A.1.1 or item h of subclause A.1.2 (depending on the profile in use), conformance of the VCL HRD for the Type I conformance point is also assured to fall within the bounds of item i of subclause A.1.1.

For conforming bitstreams, all of the following conditions shall be fulfilled for each of the tests.

– For each access unit n, with n>0, associated with a buffering period SEI message, with $\Delta t_{g,90}(n)$ specified by

$$\Delta t_{g,90}(n) = 90000 * (t_{r,n}(n) - t_{af}(n - 1)) \qquad \text{(C-14)}$$

The value of initial_cpb_removal_delay[SchedSelIdx] shall be constrained as follows.

– If cbr_flag[SchedSelIdx] is equal to 0,

$$\text{initial_cpb_removal_delay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \qquad \text{(C-15)}$$

– Otherwise (cbr_flag[SchedSelIdx] is equal to 1),

$$\text{Floor}(\Delta t_{g,90}(n)) <= \text{initial_cpb_removal_delay[SchedSelIdx]} <= \text{Ceil}(\Delta t_{g,90}(n)) \qquad \text{(C-16)}$$

NOTE 2 – The exact number of bits in the CPB at the removal time of each picture may depend on which buffering period SEI message is selected to initialize the HRD. Encoders must take this into account to ensure that all specified constraints must be obeyed regardless of which buffering period SEI message is selected to initialize the HRD, as the HRD may be initialised at any one of the buffering period SEI messages.

– A CPB overflow is specified as the condition in which the total number of bits in the CPB is larger than the CPB size. The CPB shall never overflow.

– A CPB underflow is specified as the condition in which tr,n(n) is less than taf(n). When low_delay_hrd_flag is equal to 0, the CPB shall never underflow.

– The nominal removal times of pictures from the CPB (starting from the second picture in decoding order), shall satisfy the constraints on tr,n(n) and tr(n) expressed in subclauses A.1.1 through A.1.2 for the profile and level specified in the bitstream.

– Immediately after any decoded picture is added to the DPB, the fullness of the DPB shall be less than or equal to the DPB size as constrained by Annexes A, D, and E for the profile and level specified in the bitstream.

– All reference pictures shall be present in the DPB when needed for prediction. Each picture shall be present in the DPB at its DPB output time unless it is not stored in the DPB at all, or is removed from the DPB before its output time by one of the processes specified in subclause C.2.

– The value of $\Delta_{to,dpb}(n)$ as given by Equation C-13, which is the difference between the output time of a picture and that of the picture immediately following it in output order, shall satisfy the constraint expressed in subclause A.1.1 for the profile and level specified in the bitstream.

C.4 Decoder conformance

A decoder conforming to this Recommendation | International Standard fulfils the following requirements.

A decoder claiming conformance to a specific profile and level shall be able to decode successfully all conforming bitstreams specified for decoder conformance in subclause C.3, provided that all sequence parameter sets and picture parameters sets referred to in the VCL NAL units, and appropriate buffering period and picture timing SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-VCL NAL units), or by external means not specified by this Recommendation | International Standard.

There are two types of conformance that can be claimed by a decoder: output timing conformance and output order conformance.

To check conformance of a decoder, test bitstreams conforming to the claimed profile and level, as specified by subclause C.3 are delivered by a hypothetical stream scheduler (HSS) both to the HRD and to the decoder under test (DUT). All pictures output by the HRD shall also be output by the DUT and, for each picture output by the HRD, the values of all samples that are output by the DUT for the corresponding picture shall be equal to the values of the samples output by the HRD.

For output timing decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of SchedSelIdx for which the bit rate and CPB size are restricted as specified in Annex A for the specified profile and level, or with "interpolated" delivery schedules as specified below for which the bit rate and CPB size are restricted as specified in Annex A. The same delivery schedule is used for both the HRD and DUT.

When the HRD parameters and the buffering period SEI messages are present with cpb_cnt_minus1 greater than 0, the decoder shall be capable of decoding the bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CPB size c(r), and initial CPB removal delay (f(r) ÷ r) as follows

$$\alpha = (r - BitRate[SchedSelIdx - 1]) \div (BitRate[SchedSelIdx] - BitRate[SchedSelIdx - 1]), \quad \text{(C-17)}$$

$$c(r) = \alpha * CpbSize[SchedSelIdx] + (1 - \alpha) * CpbSize[SchedSelIdx-1], \quad \text{(C-18)}$$

$$f(r) = \alpha * initial_cpb_removal_delay[SchedSelIdx] * BitRate[SchedSelIdx] + (1 - \alpha) * initial_cpb_removal_delay[SchedSelIdx - 1] * BitRate[SchedSelIdx - 1] \quad \text{(C-19)}$$

for any SchedSelIdx > 0 and r such that BitRate[SchedSelIdx - 1] <= r <= BitRate[SchedSelIdx] such that r and c(r) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile and level.

NOTE 1 – initial_cpb_removal_delay[SchedSelIdx] can be different from one buffering period to another and have to be re-calculated.

For output timing decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of picture output is the same for both HRD and the DUT up to a fixed delay.

For output order decoder conformance, the HSS delivers the bitstream to the DUT "by demand" from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

NOTE 2 – This means that for this test, the coded picture buffer of the DUT could be as small as the size of the largest access unit.

A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the bitstream such that the bit rate and CPB size are restricted as specified in Annex A. The order of pictures output shall be the same for both HRD and the DUT.

For output order decoder conformance, the HRD CPB size is equal to CpbSize[SchedSelIdx] for the selected schedule and the DPB size is equal to MaxDpbSize. Removal time from the CPB for the HRD is equal to final bit arrival time and decoding is immediate. The operation of the DPB of this HRD is described below.

C.4.1 Operation of the output order DPB

The decoded picture buffer contains frame buffers. Each of the frame buffers may contain a decoded frame, a decoded complementary field pair or a single (non-paired) decoded field that is marked as "used for reference" or is held for future output (reordered pictures). At HRD initialization, the DPB fullness, measured in frames, is set to 0. The following steps all happen instantaneously when an access unit is removed from the CPB, and in the order listed.

C.4.2 Decoding of gaps in frame_num and storage of "non-existing" pictures

When applicable, gaps in frame_num are detected by the decoding process and the necessary number of "non-existing" frames are inferred in the order specified by the generation of values of UnusedShortTermFrameNum in Equation 7-21 and are marked as specified in subclause 8.2.5.2. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied. Each "non-existing" frame is stored in the DPB as follows.

– When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the "non-existing" frame.

– The "non-existing" frame is stored in an empty frame buffer and is marked as "not needed for output", and the DPB fullness is incremented by one.

C.4.3 Picture decoding

Primary coded picture n is decoded and is temporarily stored (not in the DPB).

C.4.4 Removal of pictures from the DPB before possible insertion of the current picture

The removal of pictures from the DPB before possible insertion of the current picture proceeds as follows.

– If the decoded picture is an IDR picture the following applies.

  – All reference pictures in the DPB are marked as "unused for reference" as specified in subclause 8.2.5.

  – When the IDR picture is not the first IDR picture decoded and the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the active sequence parameter set is different from the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering derived from the sequence parameter set that was active for the preceding sequence, respectively, no_output_of_prior_pics_flag is inferred to be equal to 1 by the HRD, regardless of the actual value of no_output_of_prior_pics_flag.

    NOTE – Decoder implementations should try to handle changes in the value of PicWidthInMbs or FrameHeightInMbs or max_dec_frame_buffering more gracefully than the HRD.

  – When no_output_of_prior_pics_flag is equal to 1 or is inferred to be equal to 1, all frame buffers in the DPB are emptied without output of the pictures they contain, and DPB fullness is set to 0.

– Otherwise (the decoded picture is not an IDR picture), the decoded reference picture marking process is invoked as specified in subclause 8.2.5. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied.

When the current picture has a memory_management_control_operation equal to 5 or is an IDR picture for which no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, the following two steps are performed.

1. Frame buffers containing a frame or a complementary field pair or a non-paired field which are marked as "not needed for output" and "unused for reference" are emptied (without output), and the DPB fullness is decremented by the number of frame buffers emptied.
2. All non-empty frame buffers in the DPB are emptied by repeatedly invoking the “bumping” process specified in subclause C.4.5.3, and the DPB fullness is set to 0.

C.4.5 Current decoded picture marking and storage

C.4.5.1 Storage and marking of a reference decoded picture into the DPB

When the current picture is a reference picture, it is stored in the DPB as follows.

- If the current decoded picture is the second field (in decoding order) of a complementary reference field pair, and the first field of the pair is still in the DPB, the current picture is stored in the same frame buffer as the first field of the pair and is marked as "needed for output".
- Otherwise, the following operations are performed:
  - When there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the "bumping" process specified in subclause C.4.5.3 is invoked repeatedly until there is an empty frame buffer in which to store the current decoded picture.
  - The current decoded picture is stored in an empty frame buffer and is marked as "needed for output", and the DPB fullness is incremented by one.

C.4.5.2 Storage and marking of a non-reference decoded picture into the DPB

When the current picture is a non-reference picture, the following operations are performed.

- If the current decoded picture is the second field (in decoding order) of a complementary non-reference field pair and the first field of the pair is still in the DPB, the current picture is stored in the same frame buffer as the first field of the pair and is marked as "needed for output".
- Otherwise, the following operations are performed repeatedly until the current decoded picture has been cropped and output or has been stored in the DPB:
  - If there is no empty frame buffer (i.e., DPB fullness is equal to DPB size), the following applies.
    - If the current picture does not have a lower value of PicOrderCnt( ) than all pictures in the DPB that are marked as "needed for output", the "bumping" process described in subclause C.4.5.3 is performed.
    - Otherwise (the current picture has a lower value of PicOrderCnt( ) than all pictures in the DPB that are marked as "needed for output"), the current picture is cropped, using the cropping rectangle specified in the sequence parameter set for the sequence and the cropped picture is output.
  - Otherwise (there is an empty frame buffer, i.e., DPB fullness is less than DPB size) the current decoded picture is stored in an empty frame buffer and is marked as "needed for output", and the DPB fullness is incremented by one.

C.4.5.3 "Bumping" process

The "bumping" process is invoked in the following cases.

- There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and a empty frame buffer is needed for storage of an inferred "non-existing" frame, as specified in subclause C.4.2.
- The current picture is an IDR picture and no_output_of_prior_pics_flag is not equal to 1 and is not inferred to be equal to 1, as specified in subclause C.4.4.
- The current picture has memory_management_control_operation equal to 5, as specified in subclause C.4.4.
- There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and an empty frame buffer is needed for storage of a decoded (non-IDR) reference picture, as specified in subclause C.4.5.1.
- There is no empty frame buffer (i.e., DPB fullness is equal to DPB size) and the current picture is a non-reference picture that is not the second field of a complementary non-reference field pair and there are pictures in the DPB that are marked as "needed for output" that precede the current non-reference picture in output order, as specified in subclause C.4.5.2, so an empty buffer is needed for storage of the current picture.

The "bumping" process consists of the following:

- The picture or complementary reference field pair that is first for output is selected as follows.
  - The frame buffer is selected that contains the picture having the smallest value of PicOrderCnt( ) of all pictures in the DPB marked as "needed for output".
  - If this frame buffer contains a complementary non-reference field pair with both fields marked as "needed for output" and both fields have the same PicOrderCnt( ), the first of these two fields in decoding order is considered first for output.

  - Otherwise, if this frame buffer contains a complementary reference field pair with both fields marked as "needed for output" and both fields have the same PicOrderCnt( ), the entire complementary reference field pair is considered first for output.
  - Otherwise, the picture in this frame buffer that has the smallest value of PicOrderCnt( ) is considered first for output.
- If a single picture is considered first for output, this picture is cropped, using the cropping rectangle specified in the sequence parameter set for the sequence, the cropped picture is output, and the picture is marked as "not needed for output".
- Otherwise (a complementary reference field pair is considered first for output), the two fields of the complementary reference field pair are both cropped, using the cropping rectangle specified in the sequence parameter set for the sequence, the two fields of the complementary reference field pair are output together, and both fields of the complementary reference field pair are marked as "not needed for output".
- The frame buffer that included the picture or complementary reference field pair that was cropped and output is checked, and when any of the following conditions is satisfied, the frame buffer is emptied and the DPB fullness is decremented by 1.
  - The frame buffer contains a non-reference non-paired field.
  - The frame buffer contains a non-reference frame.
  - The frame buffer contains a complementary non-reference field pair with both fields marked as "not needed for output".
  - The frame buffer contains a non-paired reference field marked as "unused for reference".
  - The frame buffer contains a reference frame with both fields marked as "unused for reference".
  - The frame buffer contains a complementary reference field pair with both fields marked as "unused for reference" and "not needed for output".

DAnnex D
Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard)

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, display or other purposes. However, SEI messages are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex C for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In Annex D, specification for presence of SEI messages are also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in subclauses 7.3.2.3 and 7.4.2.3 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

D.1 SEI payload syntax

sei_payload(payloadType, payloadSize) {	C	Descriptor
if(payloadType = = 0)		
buffering_period(payloadSize)	5	
else if(payloadType = = 1)		
pic_timing(payloadSize)	5	
else if(payloadType = = 2)		
pan_scan_rect(payloadSize)	5	
else if(payloadType = = 3)		
filler_payload(payloadSize)	5	
else if(payloadType = = 4)		
user_data_registered_itu_t_t35(payloadSize)	5	
else if(payloadType = = 5)		
user_data_unregistered(payloadSize)	5	
else if(payloadType = = 6)		
recovery_point(payloadSize)	5	
else if(payloadType = = 7)		
dec_ref_pic_marking_repetition(payloadSize)	5	
else if(payloadType = = 8)		
spare_pic(payloadSize)	5	
else if(payloadType = = 9)		
scene_info(payloadSize)	5	
else if(payloadType = = 10)		
sub_seq_info(payloadSize)	5	
else if(payloadType = = 11)		
sub_seq_layer_characteristics(payloadSize)	5	
else if(payloadType = = 12)		
sub_seq_characteristics(payloadSize)	5	
else if(payloadType = = 13)		
full_frame_freeze(payloadSize)	5	
else if(payloadType = = 14)		
full_frame_freeze_release(payloadSize)	5	
else if(payloadType = = 15)		
full_frame_snapshot(payloadSize)	5	
else if(payloadType = = 16)		
progressive_refinement_segment_start(payloadSize)	5	
else if(payloadType = = 17)		
progressive_refinement_segment_end(payloadSize)	5	
else if(payloadType = = 18)		
motion_constrained_slice_group_set(payloadSize)	5	
else if(payloadType = = 19)		
film_grain_characteristics(payloadSize)	5	
else if(payloadType = = 20)		
deblocking_filter_display_preference(payloadSize)	5	
else if(payloadType = = 21)		
stereo_video_info(payloadSize)	5	

else if(payloadType = = 22)		
scalabilty_info(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 23)		
sub_pic_scalable_layer(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 24)		
non_required_pic(payloadSize) /* specified in Annex G */	5	
else if(payloadType = = 25)		
quality_layers_info(payloadSize) /* specified in Annex G */	5	
else if (payloadType == 26)		
scalability_info_layers_notpresent(payloadSize~~)~~) /* specified in Annex G */	5	
else if (payloadType == 27)		
scalability_info_dependency_change(payloadSize~~)~~) /* specified in Annex G */	5	
else if (payloadType == 28)		
scalable_nesting(payloadSize) /* specified in Annex G */	5	
else		
reserved_sei_message(payloadSize)	5	
if(!byte_aligned( )) {		
bit_equal_to_one /* equal to 1 */	5	f(1)
while(!byte_aligned( ))		
bit_equal_to_zero /* equal to 0 */	5	f(1)
}		
}		

D.1.1 Buffering period SEI message syntax

buffering_period(payloadSize) {	C	Descriptor
seq_parameter_set_id	5	ue(v)
if(NalHrdBpPresentFlag) {		
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
initial_cpb_removal_delay[SchedSelIdx **]**	5	u(v)
initial_cpb_removal_delay_offset[SchedSelIdx **]**	5	u(v)
}		
}		
if(VclHrdBpPresentFlag) {		
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
initial_cpb_removal_delay[SchedSelIdx **]**	5	u(v)
initial_cpb_removal_delay_offset[SchedSelIdx **]**	5	u(v)
}		
}		
}		

D.1.2 Picture timing SEI message syntax

pic_timing(payloadSize) {	C	Descriptor
if(CpbDpbDelaysPresentFlag) {		
cpb_removal_delay	5	u(v)
dpb_output_delay	5	u(v)
}		
if(pic_struct_present_flag) {		
pic_struct	5	u(4)
for(i = 0; i < NumClockTS ; i++) {		
clock_timestamp_flag[i]	5	u(1)
if(clock_timestamp_flag[i]) {		
ct_type	5	u(2)
nuit_field_based_flag	5	u(1)
counting_type	5	u(5)
full_timestamp_flag	5	u(1)
discontinuity_flag	5	u(1)
cnt_dropped_flag	5	u(1)
n_frames	5	u(8)
if(full_timestamp_flag) {		
seconds_value /* 0..59 */	5	u(6)
minutes_value /* 0..59 */	5	u(6)
hours_value /* 0..23 */	5	u(5)
} else {		
seconds_flag	5	u(1)
if(seconds_flag) {		
seconds_value /* range 0..59 */	5	u(6)
minutes_flag	5	u(1)
if(minutes_flag) {		
minutes_value /* 0..59 */	5	u(6)
hours_flag	5	u(1)
if(hours_flag)		
hours_value /* 0..23 */	5	u(5)
}		
}		
}		
if(time_offset_length > 0)		
time_offset	5	i(v)
}		
}		
}		
}		

D.1.3 Pan-scan rectangle SEI message syntax

pan_scan_rect(payloadSize) {	C	Descriptor
pan_scan_rect_id	5	ue(v)
pan_scan_rect_cancel_flag	5	u(1)
if(!pan_scan_rect_cancel_flag) {		
pan_scan_cnt_minus1	5	ue(v)
for(i = 0; i <= pan_scan_cnt_minus1; i++) {		
pan_scan_rect_left_offset[i **]**	5	se(v)
pan_scan_rect_right_offset[i **]**	5	se(v)
pan_scan_rect_top_offset[i **]**	5	se(v)
pan_scan_rect_bottom_offset[i **]**	5	se(v)
}		
pan_scan_rect_repetition_period	5	ue(v)
}		
}		

D.1.4 Filler payload SEI message syntax

filler_payload(payloadSize) {	C	Descriptor
for(k = 0; k < payloadSize; k++)		
ff_byte /* equal to 0xFF */	5	f(8)
}		

D.1.5 User data registered by ITU-T Rec. T.35 SEI message syntax

user_data_registered_itu_t_t35(payloadSize) {	C	Descriptor
itu_t_t35_country_code	5	b(8)
if(itu_t_t35_country_code != 0xFF)		
i = 1		
else {		
itu_t_t35_country_code_extension_byte	5	b(8)
i = 2		
}		
do {		
itu_t_t35_payload_byte	5	b(8)
i++		
} while(i < payloadSize)		
}		

D.1.6 User data unregistered SEI message syntax

user_data_unregistered(payloadSize) {	C	Descriptor
uuid_iso_iec_11578	**5**	u(128)
for(i = 16; i < payloadSize; i++)		
user_data_payload_byte	5	b(8)
}		

D.1.7 Recovery point SEI message syntax

recovery_point(payloadSize) {	C	Descriptor
recovery_frame_cnt	5	ue(v)
exact_match_flag	5	u(1)
broken_link_flag	5	u(1)
changing_slice_group_idc	5	u(2)
}		

D.1.8 Decoded reference picture marking repetition SEI message syntax

dec_ref_pic_marking_repetition(payloadSize) {	C	Descriptor
original_idr_flag	5	u(1)
original_frame_num	5	ue(v)
if(!frame_mbs_only_flag) {		
original_field_pic_flag	5	u(1)
if(original_field_pic_flag)		
original_bottom_field_flag	5	u(1)
}		
dec_ref_pic_marking( )	5	
}		

D.1.9 Spare picture SEI message syntax

spare_pic(payloadSize) {	C	Descriptor
target_frame_num	5	ue(v)
spare_field_flag	5	u(1)
if(spare_field_flag)		
target_bottom_field_flag	5	u(1)
num_spare_pics_minus1	5	ue(v)
for(i = 0; i < num_spare_pics_minus1 + 1; i++) {		
delta_spare_frame_num[i **]**	5	ue(v)
if(spare_field_flag)		
spare_bottom_field_flag[i **]**	5	u(1)
spare_area_idc[i **]**	5	ue(v)
if(spare_area_idc[i] = = 1)		
for(j = 0; j < PicSizeInMapUnits; j++)		
spare_unit_flag[i **][** j **]**	5	u(1)
else if(spare_area_idc[i] = = 2) {		
mapUnitCnt = 0		
for(j=0; mapUnitCnt < PicSizeInMapUnits; j++) {		
zero_run_length[i **][** j **]**	5	ue(v)
mapUnitCnt += zero_run_length[i][j] + 1		
}		
}		
}		
}		

D.1.10 Scene information SEI message syntax

scene_info(payloadSize) {	C	Descriptor
scene_info_present_flag	5	u(1)
if(scene_info_present_flag) {		
scene_id	5	ue(v)
scene_transition_type	5	ue(v)
if(scene_transition_type > 3)		
second_scene_id	5	ue(v)
}		
}		

D.1.11 Sub-sequence information SEI message syntax

sub_seq_info(payloadSize) {	C	Descriptor
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
first_ref_pic_flag	5	u(1)
leading_non_ref_pic_flag	5	u(1)
last_pic_flag	5	u(1)
sub_seq_frame_num_flag	5	u(1)
if(sub_seq_frame_num_flag)		
sub_seq_frame_num	5	ue(v)
}		

D.1.12 Sub-sequence layer characteristics SEI message syntax

sub_seq_layer_characteristics(payloadSize) {	C	Descriptor
num_sub_seq_layers_minus1	5	ue(v)
for(layer = 0; layer <= num_sub_seq_layers_minus1; layer++) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
}		

D.1.13 Sub-sequence characteristics SEI message syntax

sub_seq_characteristics(payloadSize) {	C	Descriptor
sub_seq_layer_num	5	ue(v)
sub_seq_id	5	ue(v)
duration_flag	5	u(1)
if(duration_flag)		
sub_seq_duration	5	u(32)
average_rate_flag	5	u(1)
if(average_rate_flag) {		
accurate_statistics_flag	5	u(1)
average_bit_rate	5	u(16)
average_frame_rate	5	u(16)
}		
num_referenced_subseqs	5	ue(v)
for(n = 0; n < num_referenced_subseqs; n++) {		
ref_sub_seq_layer_num	5	ue(v)
ref_sub_seq_id	5	ue(v)
ref_sub_seq_direction	5	u(1)
}		
}		

D.1.14 Full-frame freeze SEI message syntax

full_frame_freeze(payloadSize) {	C	Descriptor
full_frame_freeze_repetition_period	5	ue(v)
}		

D.1.15 Full-frame freeze release SEI message syntax

full_frame_freeze_release(payloadSize) {	C	Descriptor
}		

D.1.16 Full-frame snapshot SEI message syntax

full_frame_snapshot(payloadSize) {	C	Descriptor
snapshot_id	5	ue(v)
}		

D.1.17 Progressive refinement segment start SEI message syntax

progressive_refinement_segment_start(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
num_refinement_steps_minus1	5	ue(v)
}		

D.1.18 Progressive refinement segment end SEI message syntax

progressive_refinement_segment_end(payloadSize) {	C	Descriptor
progressive_refinement_id	5	ue(v)
}		

D.1.19 Motion-constrained slice group set SEI message syntax

motion_constrained_slice_group_set(payloadSize) {	C	Descriptor
num_slice_groups_in_set_minus1	5	ue(v)
for(i = 0; i <= num_slice_groups_in_set_minus1; i++)		
slice_group_id[i **]**	5	u(v)
exact_sample_value_match_flag	5	u(1)
pan_scan_rect_flag	5	u(1)
if(pan_scan_rect_flag)		
pan_scan_rect_id	5	ue(v)
}		

D.1.20 Film grain characteristics SEI message syntax

film_grain_characteristics(payloadSize) {	C	Descriptor
film_grain_characteristics_cancel_flag	5	u(1)
if(!film_grain_**characteristics**_cancel_flag) {		
model_id	5	u(2)
separate_colour_description_present_flag	5	u(1)
if(separate_colour_description_present_flag) {		
film_grain_bit_depth_luma_minus8	5	u(3)
film_grain_bit_depth_chroma_minus8	5	u(3)
film_grain_full_range_flag	5	u(1)
film_grain_colour_primaries	5	u(8)
film_grain_transfer_characteristics	5	u(8)
film_grain_matrix_coefficients	5	u(8)
}		
blending_mode_id	5	u(2)
log2_scale_factor	5	u(4)
for(c = 0; c < 3; c++)		
comp_model_present_flag[c]	5	u(1)
for(c = 0; c < 3; c++)		
if(comp_model_present_flag[c]) {		
num_intensity_intervals_minus1[c **]**	5	u(8)
num_model_values_minus1[c **]**	5	u(3)
for(i = 0; i <= num_intensity_intervals_minus1[c]; i++) {		
intensity_interval_lower_bound[c **][** i **]**	5	u(8)
intensity_interval_upper_bound[c **][** i **]**	5	u(8)
for(j = 0; j <= num_model_values_minus1[c]; j++)		
comp_model_value[c **][** i **][** j **]**	5	se(v)
}		
}		
film_grain_characteristics_repetition_period	5	ue(v)
}		
}		

D.1.21 Deblocking filter display preference SEI message syntax

deblocking_filter_display_preference(payloadSize) {	C	Descriptor
deblocking_display_preference_cancel_flag	5	u(1)
if(!deblocking_display_preference_cancel_flag) {		
display_prior_to_deblocking_preferred_flag	5	u(1)
dec_frame_buffering_constraint_flag	5	u(1)
deblocking_display_preference_repetition_period	5	ue(v)
}		
}		

D.1.22 Stereo video information SEI message syntax

stereo_video_info(payloadSize) {	C	Descriptor
field_views_flag	5	u(1)
if(field_views_flag)		
top_field_is_left_view_flag	5	u(1)
else {		
current_frame_is_left_view_flag	5	u(1)
next_frame_is_second_view_flag	5	u(1)
}		
left_view_self_contained_flag	5	u(1)
right_view_self_contained_flag	5	u(1)
}		

D.1.23 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	C	Descriptor
for(i = 0; i < payloadSize; i++)		
reserved_sei_message_payload_byte	5	b(8)
}		

D.2 SEI payload semantics

D.2.1 Buffering period SEI message semantics

When NalHrdBpPresentFlag or VclHrdBpPresentFlag are equal to 1, a buffering period SEI message can be associated with any access unit in the bitstream, and a buffering period SEI message shall be associated with each IDR access unit and with each access unit associated with a recovery point SEI message.

NOTE – For some applications, the frequent presence of a buffering period SEI message may be desirable.

A buffering period is specified as the set of access units between two instances of the buffering period SEI message in decoding order.

seq_parameter_set_id specifies the sequence parameter set that contains the sequence HRD attributes. The value of seq_parameter_set_id shall be equal to the value of seq_parameter_set_id in the picture parameter set referenced by the primary coded picture associated with the buffering period SEI message. The value of seq_parameter_set_id shall be in the range of 0 to 31, inclusive.

initial_cpb_removal_delay[SchedSelIdx] specifies the delay for the SchedSelIdx-th CPB between the time of arrival in the CPB of the first bit of the coded data associated with the access unit associated with the buffering period SEI message and the time of removal from the CPB of the coded data associated with the same access unit, for the first buffering period after HRD initialisation. The syntax element has a length in bits given by initial_cpb_removal_delay_length_minus1 + 1. It is in units of a 90 kHz clock. initial_cpb_removal_delay[SchedSelIdx] shall not be equal to 0 and shall not exceed 90000 * (CpbSize[SchedSelIdx] ÷ BitRate[SchedSelIdx]), the time-equivalent of the CPB size in 90 kHz clock units.

initial_cpb_removal_delay_offset[SchedSelIdx] is used for the SchedSelIdx-th CPB in combination with the cpb_removal_delay to specify the initial delivery time of coded access units to the CPB. initial_cpb_removal_delay_offset[SchedSelIdx] is in units of a 90 kHz clock. The initial_cpb_removal_delay_offset[SchedSelIdx] syntax element is a fixed length code whose length in bits is given by initial_cpb_removal_delay_length_minus1 + 1. This syntax element is not used by decoders and is needed only for the delivery scheduler (HSS) specified in Annex C.

Over the entire coded video sequence, the sum of initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] shall be constant for each value of SchedSelIdx.

D.2.2 Picture timing SEI message semantics

The presence of picture timing SEI message in the bitstream is specified as follows.

– If CpbDpbDelaysPresentFlag is equal to 1 or pic_struct_present_flag is equal to 1, one picture timing SEI message shall be present in every access unit of the coded video sequence.

– Otherwise (CpbDpbDelaysPresentFlag is equal to 0 and pic_struct_present_flag is equal to 0), no picture timing SEI messages shall be present in any access unit of the coded video sequence.

cpb_removal_delay specifies how many clock ticks (see subclause E.2.1) to wait after removal from the CPB of the access unit associated with the most recent buffering period SEI message before removing from the buffer the access unit data associated with the picture timing SEI message. This value is also used to calculate an earliest possible time of arrival of access unit data into the CPB for the HSS, as specified in Annex C. The syntax element is a fixed length code whose length in bits is given by cpb_removal_delay_length_minus1 + 1. The cpb_removal_delay is the remainder of a $2^{(cpb_removal_delay_length_minus1 + 1)}$ counter.

The value of cpb_removal_delay for the first picture in the bitstream shall be equal to 0.

dpb_output_delay is used to compute the DPB output time of the picture. It specifies how many clock ticks to wait after removal of an access unit from the CPB before the decoded picture can be output from the DPB (see subclause C.2).

NOTE 1 – A picture is not removed from the DPB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

NOTE 2 – Only one dpb_output_delay is specified for a decoded picture.

The size of the syntax element dpb_output_delay is given in bits by dpb_output_delay_length_minus1 + 1. When max_dec_frame_buffering is equal to 0, dpb_output_delay shall be equal to 0.

The output time derived from the dpb_output_delay of any picture that is output from an output timing conforming decoder as specified in subclause C.2 shall precede the output time derived from the dpb_output_delay of all pictures in any subsequent coded video sequence in decoding order.

The output time derived from the dpb_output_delay of the second field, in decoding order, of a complementary non-reference field pair shall exceed the output time derived from the dpb_output_delay of the first field of the same complementary non-reference field pair.

The picture output order established by the values of this syntax element shall be the same order as established by the values of PicOrderCnt( ) as specified by subclauses C.4.1 to C.4.5, except that when the two fields of a complementary reference field pair have the same value of PicOrderCnt( ), the two fields have different output times.

For pictures that are not output by the "bumping" process of subclause C.4.5 because they precede, in decoding order, an IDR picture with no_output_of_prior_pics_flag equal to 1 or inferred to be equal to 1, the output times derived from dpb_output_delay shall be increasing with increasing value of PicOrderCnt( ) relative to all pictures within the same coded video sequence subsequent to any picture having a memory_management_control_operation equal to 5.

pic_struct indicates whether a picture should be displayed as a frame or one or more fields, according to Table D-1. Frame doubling (pic_struct equal to 7) indicates that the frame should be displayed two times consecutively, and frame tripling (pic_struct equal to 8) indicates that the frame should be displayed three times consecutively.

NOTE 3 – Frame doubling can facilitate the display, for example, of 25p video on a 50p display and 29.97p video on a 59.94p display. Using frame doubling and frame tripling in combination on every other frame can facilitate the display of 23.98p video on a 59.94p display.

Table D-1 – Interpretation of pic_struct

Value	Indicated display of picture	Restrictions	NumClockTS
0	frame	field_pic_flag shall be 0	1
1	top field	field_pic_flag shall be 1, bottom_field_flag shall be 0	1
2	bottom field	field_pic_flag shall be 1, bottom_field_flag shall be 1	1
3	top field, bottom field, in that order	field_pic_flag shall be 0	2
4	bottom field, top field, in that order	field_pic_flag shall be 0	2
5	top field, bottom field, top field repeated, in that order	field_pic_flag shall be 0	3
6	bottom field, top field, bottom field repeated, in that order	field_pic_flag shall be 0	3
7	frame doubling	field_pic_flag shall be 0 fixed_frame_rate_flag shall be 1	2
8	frame tripling	field_pic_flag shall be 0 fixed_frame_rate_flag shall be 1	3
9..15	reserved		

NumClockTS is determined by pic_struct as specified in Table D-1. There are up to NumClockTS sets of clock timestamp information for a picture, as specified by clock_timestamp_flag[i] for each set. The sets of clock timestamp information apply to the field(s) or the frame(s) associated with the picture by pic_struct.

The contents of the clock timestamp syntax elements indicate a time of origin, capture, or alternative ideal display. This indicated time is computed as

$$\text{clockTimestamp} = ((hH * 60 + mM) * 60 + sS) * \text{time_scale} + \text{nFrames} * (\text{num_units_in_tick} * (1 + \text{nuit_field_based_flag})) + \text{tOffset}, \qquad \text{(D-1)}$$

in units of clock ticks of a clock with clock frequency equal to time_scale Hz, relative to some unspecified point in time for which clockTimestamp is equal to 0. Output order and DPB output timing are not affected by the value of clockTimestamp. When two or more frames with pic_struct equal to 0 are consecutive in output order and have equal values of clockTimestamp, the indication is that the frames represent the same content and that the last such frame in output order is the preferred representation.

NOTE 4 – clockTimestamp time indications may aid display on devices with refresh rates other than those well-matched to DPB output times.

clock_timestamp_flag[i **]** equal to 1 indicates that a number of clock timestamp syntax elements are present and follow immediately. clock_timestamp_flag[i] equal to 0 indicates that the associated clock timestamp syntax elements are not present. When NumClockTS is greater than 1 and clock_timestamp_flag[i] is equal to 1 for more than one value of i, the value of clockTimestamp shall be non-decreasing with increasing value of i.

ct_type indicates the scan type (interlaced or progressive) of the source material as specified in Table D-2.

Two fields of a coded frame may have different values of ct_type.

When clockTimestamp is equal for two fields of opposite parity that are consecutive in output order, both with ct_type equal to 0 (progressive) or ct_type equal to 2 (unknown), the two fields are indicated to have come from the same original progressive frame. Two consecutive fields in output order shall have different values of clockTimestamp when the value of ct_type for either field is 1 (interlaced).

Table D-2 – Mapping of ct_type to source picture scan

Value	Original picture scan
0	progressive
1	interlaced
2	unknown
3	reserved

nuit_field_based_flag is used in calculating clockTimestamp, as specified in Equation D-1.

counting_type specifies the method of dropping values of the n_frames as specified in Table D-3.

Table D-3 – Definition of counting_type values

Value	Interpretation
0	no dropping of n_frames count values and no use of time_offset
1	no dropping of n_frames count values
2	dropping of individual zero values of n_frames count
3	dropping of individual MaxFPS-1 values of n_frames count
4	dropping of the two lowest (value 0 and 1) n_frames counts when seconds_value is equal to 0 and minutes_value is not an integer multiple of 10
5	dropping of unspecified individual n_frames count values
6	dropping of unspecified numbers of unspecified n_frames count values
7..31	reserved

full_timestamp_flag equal to 1 specifies that the n_frames syntax element is followed by seconds_value, minutes_value, and hours_value. full_timestamp_flag equal to 0 specifies that the n_frames syntax element is followed by seconds_flag.

discontinuity_flag equal to 0 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous clock timestamp in output order can be interpreted as the time difference between the times of origin or capture of the associated frames or fields. discontinuity_flag equal to 1 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous clock timestamp in output order should not be interpreted as the time difference between the times of origin or capture of the associated frames or fields. When discontinuity_flag is equal to 0, the value of clockTimestamp shall be greater than or equal to all values of clockTimestamp present for the preceding picture in DPB output order.

cnt_dropped_flag specifies the skipping of one or more values of n_frames using the counting method specified by counting_type.

n_frames specifies the value of nFrames used to compute clockTimestamp. n_frames shall be less than

$$\text{MaxFPS} = \text{Ceil}(\ \text{time_scale} \div \text{num_units_in_tick}\ ) \qquad \text{(D-2)}$$

NOTE 5 – n_frames is a frame-based counter. For field-specific timing indications, time_offset should be used to indicate a distinct clockTimestamp for each field.

When counting_type is equal to 2 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 1 and the value of n_frames for the previous picture in output order shall not be equal to 0 unless discontinuity_flag is equal to 1.

NOTE 6 – When counting_type is equal to 2, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 25 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value n_frames equal to 0 when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 1 to 12, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 3 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 0 and the value of n_frames for the previous picture in output order shall not be equal to MaxFPS – 1 unless discontinuity_flag is equal to 1.

NOTE 7 – When counting_type is equal to 3, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 25 and num_units_in_tick equal to 2 and nuit_field_based_flag equal to 0) can be avoided by occasionally skipping over the value n_frames equal to MaxFPS when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 0 to 11, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 4 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 2 and the specified value of sS shall be zero and the specified value of mM shall not be an integer multiple of ten and n_frames for the previous picture in output order shall not be equal to 0 or 1 unless discontinuity_flag is equal to 1.

NOTE 8 – When counting_type is equal to 4, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates (e.g., 30000÷1001 frames per second with time_scale equal to 60000 and num_units_in_tick equal to 1 001 and nuit_field_based_flag equal to 1) can be reduced by occasionally skipping over the value n_frames equal to MaxFPS when counting (e.g., counting n_frames from 0 to 29, then incrementing seconds_value and counting n_frames from 0 to 29, etc., until the seconds_value is zero and minutes_value is not an integer multiple of ten, then counting n_frames from 2 to 29, then incrementing seconds_value and counting n_frames from 0 to 29, etc.). This counting method is well known in industry and is often referred to as "NTSC drop-frame" counting.

When counting_type is equal to 5 or 6 and cnt_dropped_flag is equal to 1, n_frames shall not be equal to 1 plus the value of n_frames for the previous picture in output order modulo MaxFPS unless discontinuity_flag is equal to 1.

NOTE 9 – When counting_type is equal to 5 or 6, the need for increasingly large magnitudes of tOffset in Equation D-1 when using fixed non-integer frame rates can be avoided by occasionally skipping over some values of n_frames when counting. The specific values of n_frames that are skipped are not specified when counting_type is equal to 5 or 6.

seconds_flag equal to 1 specifies that seconds_value and minutes_flag are present when full_timestamp_flag is equal to 0. seconds_flag equal to 0 specifies that seconds_value and minutes_flag are not present.

seconds_value specifies the value of sS used to compute clockTimestamp. The value of seconds_value shall be in the range of 0 to 59, inclusive. When seconds_value is not present, the previous seconds_value in decoding order shall be used as sS to compute clockTimestamp.

minutes_flag equal to 1 specifies that minutes_value and hours_flag are present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1. minutes_flag equal to 0 specifies that minutes_value and hours_flag are not present.

minutes_value specifies the value of mM used to compute clockTimestamp. The value of minutes_value shall be in the range of 0 to 59, inclusive. When minutes_value is not present, the previous minutes_value in decoding order shall be used as mM to compute clockTimestamp.

hours_flag equal to 1 specifies that hours_value is present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1 and minutes_flag is equal to 1.

hours_value specifies the value of hH used to compute clockTimestamp. The value of hours_value shall be in the range of 0 to 23, inclusive. When hours_value is not present, the previous hours_value in decoding order shall be used as hH to compute clockTimestamp.

time_offset specifies the value of tOffset used to compute clockTimestamp. The number of bits used to represent time_offset shall be equal to time_offset_length. When time_offset is not present, the value 0 shall be used as tOffset to compute clockTimestamp.

D.2.3 Pan-scan rectangle SEI message semantics

The pan-scan rectangle SEI message syntax elements specify the coordinates of a rectangle relative to the cropping rectangle of the sequence parameter set. Each coordinate of this rectangle is specified in units of one-sixteenth sample spacing relative to the luma sampling grid.

pan_scan_rect_id contains an identifying number that may be used to identify the purpose of the pan-scan rectangle (for example, to identify the rectangle as the area to be shown on a particular display device or as the area that contains a particular actor in the scene). The value of pan_scan_rect_id shall be in the range of 0 to $2^{32} – 1$, inclusive.

Values of pan_scan_rect_id from 0 to 255 and from 512 to 2^{31}-1 may be used as determined by the application. Values of pan_scan_rect_id from 256 to 511 and from 2^{31} to 2^{32}-1 are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of pan_scan_rect_id in the range of 256 to 511 or in the range of 2^{31} to 2^{32} - 1 shall ignore (remove from the bitstream and discard) it.

pan_scan_rect_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous pan-scan rectangle SEI message in output order. pan_scan_rect_cancel_flag equal to 0 indicates that pan-scan rectangle information follows.

pan_scan_cnt_minus1 specifies the number of pan-scan rectangles that are present in the SEI message. pan_scan_cnt_minus1 shall be in the range of 0 to 2, inclusive. pan_scan_cnt_minus1 equal to 0 indicates that a single pan-scan rectangle is present that applies to all fields of the decoded picture. pan_scan_cnt_minus1 shall be equal to 0 when the current picture is a field. pan_scan_cnt_minus1 equal to 1 indicates that two pan-scan rectangles are present, the first of which applies to the first field of the picture in output order and the second of which applies to the second field of the picture in output order. pan_scan_cnt_minus1 equal to 2 indicates that three pan-scan rectangles are present, the first of which applies to the first field of the picture in output order, the second of which applies to the second field of the picture in output order, and the third of which applies to a repetition of the first field as a third field in output order.

pan_scan_rect_left_offset[i **]**, **pan_scan_rect_right_offset[** i **]**, **pan_scan_rect_top_offset[** i **]**, and **pan_scan_rect_bottom_offset[** i **]**, specify, as signed integer quantities in units of one-sixteenth sample spacing relative to the luma sampling grid, the location of the pan-scan rectangle. The values of each of these four syntax elements shall be in the range of -2^{31} to 2^{31} - 1, inclusive.

The pan-scan rectangle is specified, in units of one-sixteenth sample spacing relative to a luma frame sampling grid, as the region with frame horizontal coordinates from 16*CropUnitX * frame_crop_left_offset + pan_scan_rect_left_offset[i] to 16 * (16 * PicWidthInMbs – CropUnitX * frame_crop_right_offset) + pan_scan_rect_right_offset[i] – 1 and with vertical coordinates from 16 *CropUnitY * frame_crop_top_offset + pan_scan_rect_top_offset[i] to 16 * (16 * PicHeightInMbs – CropUnitY * frame_crop_bottom_offset) + pan_scan_rect_bottom_offset[i] – 1, inclusive. The value of 16 * CropUnitX * frame_crop_left_offset + pan_scan_rect_left_offset[i] shall be less than or equal to 16 * (16 * PicWidthInMbs – CropUnitX * frame_crop_right_offset) + pan_scan_rect_right_offset[i] – 1; and the value of 16 CropUnitY * frame_crop_top_offset + pan_scan_rect_top_offset[i] shall be less than or equal to 16 * (16 * PicHeightInMbs – CropUnitY * frame_crop_bottom_offset) + pan_scan_rect_bottom_offset[i] – 1.

When the pan-scan rectangular area includes samples outside of the cropping rectangle, the region outside of the cropping rectangle may be filled with synthesized content (such as black video content or neutral grey video content) for display.

pan_scan_rect_repetition_period specifies the persistence of the pan-scan rectangle SEI message and may specify a picture order count interval within which another pan-scan rectangle SEI message with the same value of pan_scan_rect_id or the end of the coded video sequence shall be present in the bitstream. The value of pan_scan_rect_repetition_period shall be in the range of 0 to 16 384, inclusive. When pan_scan_cnt_minus1 is greater than 0, pan_scan_rect_repetition_period shall not be greater than 1.

pan_scan_rect_repetition_period equal to 0 specifies that the pan-scan rectangle information applies to the current decoded picture only.

pan_scan_rect_repetition_period equal to 1 specifies that the pan-scan rectangle information persists in output order until any of the following conditions are true.

– A new coded video sequence begins

– A picture in an access unit containing a pan-scan rectangle SEI message with the same value of pan_scan_rect_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

pan_scan_rect_repetition_period equal to 0 or equal to 1 indicates that another pan-scan rectangle SEI message with the same value of pan_scan_rect_id may or may not be present.

pan_scan_rect_repetition_period greater than 1 specifies that the pan-scan rectangle information persists until any of the following conditions are true.

– A new coded video sequence begins

– A picture in an access unit containing a pan-scan rectangle SEI message with the same value of pan_scan_rect_id is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + pan_scan_rect_repetition_period.

pan_scan_rect_repetition_period greater than 1 indicates that another pan-scan rectangle SEI message with the same value of pan_scan_rect_id shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + pan_scan_rect_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.4 Filler payload SEI message semantics

This message contains a series of payloadSize bytes of value 0xFF, which can be discarded.

ff_byte shall be a byte having the value 0xFF.

D.2.5 User data registered by ITU-T Rec. T.35 SEI message semantics

This message contains user data registered as specified by ITU-T Rec. T.35, the contents of which are not specified by this Recommendation | International Standard.

itu_t_t35_country_code shall be a byte having a value specified as a country code by ITU-T Rec. T.35 Annex A.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as a country code by ITU-T Rec. T.35 Annex B.

itu_t_t35_payload_byte shall be a byte containing data registered as specified by ITU-T Rec. T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the itu_t_t35_payload_byte, in the format specified by the Administration that issued the terminal provider code. Any remaining itu_t_t35_payload_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

D.2.6 User data unregistered SEI message semantics

This message contains unregistered user data identified by a UUID, the contents of which are not specified by this Recommendation | International Standard.

uuid_iso_iec_11578 shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996 Annex A.

user_data_payload_byte shall be a byte containing data having syntax and semantics as specified by the UUID generator.

D.2.7 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable pictures for display after the decoder initiates random access or after the encoder indicates a broken link in the sequence. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded pictures at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content. Decoded pictures produced by random access at or before the picture associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the picture associated with the recovery point SEI message may contain references to pictures not available in the decoded picture buffer.

In addition, by use of the broken_link_flag, the recovery point SEI message can indicate to the decoder the location of some pictures in the bitstream that can result in serious visual artefacts when displayed, even when the decoding process was begun at the location of a previous IDR access unit in decoding order.

NOTE 1 – The broken_link_flag can be used by encoders to indicate the location of a point after which the decoding process for the decoding of some pictures may cause references to pictures that, though available for use in the decoding process, are not the pictures that were used for reference when the bitstream was originally encoded (e.g., due to a splicing operation performed during the generation of the bitstream).

The recovery point is specified as a count in units of frame_num increments subsequent to the frame_num of the current access unit at the position of the SEI message.

NOTE 2 – When HRD information is present in the bitstream, a buffering period SEI message should be associated with the access unit associated with the recovery point SEI message in order to establish initialisation of the HRD buffer model after a random access.

recovery_frame_cnt specifies the recovery point of output pictures in output order. All decoded pictures in output order are indicated to be correct or approximately correct in content starting at the output order position of the reference picture having the frame_num equal to the frame_num of the VCL NAL units for the current access unit incremented by recovery_frame_cnt in modulo MaxFrameNum arithmetic. recovery_frame_cnt shall be in the range of 0 to MaxFrameNum – 1, inclusive.

exact_match_flag indicates whether decoded pictures at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message shall be an exact match to the pictures that would be produced by starting the decoding process at the location of a previous IDR access unit in the NAL unit stream. The value 0 indicates that the match need not be exact and the value 1 indicates that the match shall be exact.

When decoding starts from the location of the recovery point SEI message, all references to not available reference pictures shall be inferred as references to pictures containing only macroblocks coded using Intra macroblock prediction modes and having sample values given by Y samples equal to 128, Cb samples equal to 128, and Cr samples equal to 128 (mid-level grey) for purposes of determining the conformance of the value of exact_match_flag.

NOTE 3 – When performing random access, decoders should infer all references to not available reference pictures as references to pictures containing only intra macroblocks and having sample values given by Y equal to 128, Cb equal to 128, and Cr equal to 128 (mid-level grey), regardless of the value of exact_match_flag.

When exact_match_flag is equal to 0, the quality of the approximation at the recovery point is chosen by the encoding process and is not specified by this Recommendation | International Standard.

broken_link_flag indicates the presence or absence of a broken link in the NAL unit stream at the location of the recovery point SEI message and is assigned further semantics as follows.

– If broken_link_flag is equal to 1, pictures produced by starting the decoding process at the location of a previous IDR access unit may contain undesirable visual artefacts to the extent that decoded pictures at and subsequent to the access unit associated with the recovery point SEI message in decoding order should not be displayed until the specified recovery point in output order.

– Otherwise (broken_link_flag is equal to 0), no indication is given regarding any potential presence of visual artefacts.

Regardless of the value of the broken_link_flag, pictures subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

NOTE 4 – When a sub-sequence information SEI message is present in conjunction with a recovery point SEI message in which broken_link_flag is equal to 1 and when sub_seq_layer_num is equal to 0, sub_seq_id should be different from the latest sub_seq_id for sub_seq_layer_num equal to 0 that was decoded prior to the location of the recovery point SEI message. When broken_link_flag is equal to 0, the sub_seq_id in sub-sequence layer 0 should remain unchanged.

changing_slice_group_idc equal to 0 indicates that decoded pictures are correct or approximately correct in content at and subsequent to the recovery point in output order when all macroblocks of the primary coded pictures are decoded within the changing slice group period, i.e., the period between the access unit associated with the recovery point SEI message (inclusive) and the specified recovery point (inclusive) in decoding order. changing_slice_group_idc shall be equal to 0 when num_slice_groups_minus1 is equal to 0 in any primary coded picture within the changing slice group period.

When changing_slice_group_idc is equal to 1 or 2, num_slice_groups_minus1 shall be equal to 1 and the macroblock-to-slice-group map type 3, 4, or 5 shall be applied in each primary coded picture in the changing slice group period.

changing_slice_group_idc equal to 1 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 0 are used for inter prediction of any macroblock within slice group 0. In addition, changing_slice_group_idc equal to 1 indicates that when all macroblocks in slice group 0 within the changing slice group period are decoded, decoded pictures will be correct or approximately correct in content at and subsequent to the specified recovery point in output order regardless of whether any macroblock in slice group 1 within the changing slice group period is decoded.

changing_slice_group_idc equal to 2 indicates that within the changing slice group period no sample values outside the decoded macroblocks covered by slice group 1 are used for inter prediction of any macroblock within slice group 1. In addition, changing_slice_group_idc equal to 2 indicates that when all macroblocks in slice group 1 within the changing slice group period are decoded, decoded pictures will be correct or approximately correct in content at and subsequent

to the specified recovery point in output order regardless of whether any macroblock in slice group 0 within the changing slice group period is decoded.

changing_slice_group_idc shall be in the range of 0 to 2, inclusive.

D.2.8 Decoded reference picture marking repetition SEI message semantics

The decoded reference picture marking repetition SEI message is used to repeat the decoded reference picture marking syntax structure that was located in the slice header of an earlier picture in the sequence in decoding order.

original_idr_flag shall be equal to 1 when the decoded reference picture marking syntax structure occurred originally in an IDR picture. original_idr_flag shall be equal to 0 when the repeated decoded reference picture marking syntax structure did not occur in an IDR picture originally.

original_frame_num shall be equal to the frame_num of the picture where the repeated decoded reference picture marking syntax structure originally occurred. The picture indicated by original_frame_num is the previous coded picture having the specified value of frame_num. The value of original_frame_num used to refer to a picture having a memory_management_control_operation equal to 5 shall be 0.

original_field_pic_flag shall be equal to the field_pic_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

original_bottom_field_flag shall be equal to the bottom_field_flag of the picture where the repeated decoded reference picture marking syntax structure originally occurred.

dec_ref_pic_marking( ) shall contain a copy of the decoded reference picture marking syntax structure of the picture whose frame_num was original_frame_num. The nal_unit_type used for specification of the repeated dec_ref_pic_marking( ) syntax structure shall be the nal_unit_type of the slice header(s) of the picture whose frame_num was original_frame_num (i.e., nal_unit_type as used in subclause 7.3.3.3 shall be considered equal to 5 when original_idr_flag is equal to 1 and shall not be considered equal to 5 when original_idr_flag is equal to 0).

D.2.9 Spare picture SEI message semantics

This SEI message indicates that certain slice group map units, called spare slice group map units, in one or more decoded reference pictures resemble the co-located slice group map units in a specified decoded picture called the target picture. A spare slice group map unit may be used to replace a co-located, incorrectly decoded slice group map unit, in the target picture. A decoded picture containing spare slice group map units is called a spare picture.

For all spare pictures identified in a spare picture SEI message, the value of frame_mbs_only_flag shall be equal to the value of frame_mbs_only_flag of the target picture in the same SEI message. The spare pictures in the SEI message are constrained as follows.

– If the target picture is a decoded field, all spare pictures identified in the same SEI message shall be decoded fields.

– Otherwise (the target picture is a decoded frame), all spare pictures identified in the same SEI message shall be decoded frames.

For all spare pictures identified in a spare picture SEI message, the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1 shall be equal to the values of pic_width_in_mbs_minus1 and pic_height_in_map_units_minus1, respectively, of the target picture in the same SEI message. The picture associated (as specified in subclause 7.4.1.2.3) with this message shall appear after the target picture, in decoding order.

target_frame_num indicates the frame_num of the target picture.

spare_field_flag equal to 0 indicates that the target picture and the spare pictures are decoded frames. spare_field_flag equal to 1 indicates that the target picture and the spare pictures are decoded fields.

target_bottom_field_flag equal to 0 indicates that the target picture is a top field. target_bottom_field_flag equal to 1 indicates that the target picture is a bottom field.

A target picture is a decoded reference picture whose corresponding primary coded picture precedes the current picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to target_frame_num, spare_field_flag and target_bottom_field_flag, respectively.

num_spare_pics_minus1 indicates the number of spare pictures for the specified target picture. The number of spare pictures is equal to num_spare_pics_minus1 + 1. The value of num_spare_pics_minus1 shall be in the range of 0 to 15, inclusive.

delta_spare_frame_num[i **]** is used to identify the spare picture that contains the i-th set of spare slice group map units, hereafter called the i-th spare picture, as specified below. The value of delta_spare_frame_num[i] shall be in the range of 0 to MaxFrameNum - 1 - !spare_field_flag, inclusive.

The frame_num of the i-th spare picture, spareFrameNum[i], is derived as follows for all values of i from 0 to num_spare_pics_minus1, inclusive:

```
candidateSpareFrameNum = target_frame_num - !spare_field_flag
for ( i = 0; i <= num_spare_pics_minus1; i++ ) {
    if( candidateSpareFrameNum < 0 )
        candidateSpareFrameNum = MaxFrameNum – 1
    spareFrameNum[ i ] = candidateSpareFrameNum – delta_spare_frame_num[ i ]          (D-3)
    if( spareFrameNum[ i ] < 0 )
        spareFrameNum[ i ] = MaxFrameNum + spareFrameNum[ i ]
    candidateSpareFrameNum = spareFrameNum[ i ] - !spare_field_flag
}
```

spare_bottom_field_flag[i **]** equal to 0 indicates that the i-th spare picture is a top field. spare_bottom_field_flag[i] equal to 1 indicates that the i-th spare picture is a bottom field.

The 0-th spare picture is a decoded reference picture whose corresponding primary coded picture precedes the target picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to spareFrameNum[0], spare_field_flag and spare_bottom_field_flag[0], respectively. The i-th spare picture is a decoded reference picture whose corresponding primary coded picture precedes the (i - 1)-th spare picture, in decoding order, and in which the values of frame_num, field_pic_flag (when present) and bottom_field_flag (when present) are equal to spareFrameNum[i], spare_field_flag and spare_bottom_field_flag[i], respectively.

spare_area_idc[i **]** indicates the method used to identify the spare slice group map units in the i-th spare picture. spare_area_idc[i] shall be in the range of 0 to 2, inclusive. spare_area_idc[i] equal to 0 indicates that all slice group map units in the i-th spare picture are spare units. spare_area_idc[i] equal to 1 indicates that the value of the syntax element spare_unit_flag[i][j] is used to identify the spare slice group map units. spare_area_idc[i] equal to 2 indicates that the zero_run_length[i][j] syntax element is used to derive the values of spareUnitFlagInBoxOutOrder[i][j], as described below.

spare_unit_flag[i **][** j **]** equal to 0 indicates that the j-th slice group map unit in raster scan order in the i-th spare picture is a spare unit. spare_unit_flag[i][j] equal to 1 indicates that the j-th slice group map unit in raster scan order in the i-th spare picture is not a spare unit.

zero_run_length[i **][** j **]** is used to derive the values of spareUnitFlagInBoxOutOrder[i][j] when spare_area_idc[i] is equal to 2. In this case, the spare slice group map units identified in spareUnitFlagInBoxOutOrder[i][j] appear in counter-clockwise box-out order, as specified in subclause 8.2.2.4, for each spare picture. spareUnitFlagInBoxOutOrder[i][j] equal to 0 indicates that the j-th slice group map unit in counter-clockwise box-out order in the i-th spare picture is a spare unit. spareUnitFlagInBoxOutOrder[i][j] equal to 1 indicates that the j-th slice group map unit in counter-clockwise box-out order in the i-th spare picture is not a spare unit.

When spare_area_idc[0] is equal to 2, spareUnitFlagInBoxOutOrder[0][j] is derived as follows:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
    for( k = 0; k < zero_run_length[ 0 ][ loop ]; k++ )
        spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 0                                     (D-4)
    spareUnitFlagInBoxOutOrder[ 0 ][ j++ ] = 1
}
```

When spare_area_idc[i] is equal to 2 and the value of i is greater than 0, spareUnitFlagInBoxOutOrder[i][j] is derived as follows:

```
for( j = 0, loop = 0; j < PicSizeInMapUnits; loop++ ) {
    for( k = 0; k < zero_run_length[ i ][ loop ]; k++ )
        spareUnitFlagInBoxOutOrder[ i ][ j ] = spareUnitFlagInBoxOutOrder[ i - 1 ][ j++ ]   (D-5)
    spareUnitFlagInBoxOutOrder[ i ][ j ] = !spareUnitFlagInBoxOutOrder[ i - 1 ][ j++ ]
}
```

D.2.10 Scene information SEI message semantics

A scene and a scene transition are herein defined as a set of consecutive pictures in output order.

NOTE 1 – Decoded pictures within one scene generally have similar content. The scene information SEI message is used to label pictures with scene identifiers and to indicate scene changes. The message specifies how the source pictures for the labelled pictures were created. The decoder may use the information to select an appropriate algorithm to conceal transmission errors. For example, a specific algorithm may be used to conceal transmission errors that occurred in pictures belonging to a gradual scene transition. Furthermore, the scene information SEI message may be used in a manner determined by the application, such as for indexing the scenes of a coded sequence.

A scene information SEI message labels all pictures, in decoding order, from the primary coded picture to which the SEI message is associated (inclusive), as specified in subclause 7.4.1.2.3, to the primary coded picture to which the next scene information SEI message (if present) in decoding order is associated (exclusive) or (otherwise) to the last access unit in the bitstream (inclusive). These pictures are herein referred to as the target pictures.

scene_info_present_flag equal to 0 indicates that the scene or scene transition to which the target pictures belong is unspecified. scene_info_present_flag equal to 1 indicates that the target pictures belong to the same scene or scene transition.

scene_id identifies the scene to which the target pictures belong. When the value of scene_transition_type of the target pictures is less than 4, and the previous picture in output order is marked with a value of scene_transition_type less than 4, and the value of scene_id is the same as the value of scene_id of the previous picture in output order, this indicates that the source scene for the target pictures and the source scene for the previous picture (in output order) are considered by the encoder to have been the same scene. When the value of scene_transition_type of the target pictures is greater than 3, and the previous picture in output order is marked with a value of scene_transition_type less than 4, and the value of scene_id is the same as the value of scene_id of the previous picture in output order, this indicates that one of the source scenes for the target pictures and the source scene for the previous picture (in output order) are considered by the encoder to have been the same scene. When the value of scene_id is not equal to the value of scene_id of the previous picture in output order, this indicates that the target pictures and the previous picture (in output order) are considered by the encoder to have been from different source scenes.

The value of scene_id shall be in the range of 0 to 2^{32}-1, inclusive. Values of scene_id in the range of 0 to 255, inclusive, and in the range of 512 to $2^{31} - 1$, inclusive, may be used as determined by the application. Values of scene_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to $2^{32} - 1$, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of scene_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to 2^{32} - 1, inclusive, shall ignore (remove from the bitstream and discard) it.

scene_transition_type specifies in which type of a scene transition (if any) the target pictures are involved. The valid values of scene_transition_type are specified in Table D-4.

Table D-4 – scene_transition_type values

Value	Description
0	No transition
1	Fade to black
2	Fade from black
3	Unspecified transition from or to constant colour
4	Dissolve
5	Wipe
6	Unspecified mixture of two scenes

When scene_transition_type is greater than 3, the target pictures include contents both from the scene labelled by its scene_id and the next scene, in output order, which is labelled by second_scene_id (see below). The term “the current scene” is used to indicate the scene labelled by scene_id. The term “the next scene” is used to indicate the scene labelled by second_scene_id. It is not required for any following picture, in output order, to be labelled with scene_id equal to second_scene_id of the current SEI message.

Scene transition types are specified as follows.

“No transition” specifies that the target pictures are not involved in a gradual scene transition.

NOTE 2 – When two consecutive pictures in output order have scene_transition_type equal to 0 and different values of scene_id, a scene cut occurred between the two pictures.

"Fade to black" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade to black scene transition, i.e., the luma samples of the scene gradually approach zero and the chroma samples of the scene gradually approach 128.

NOTE 3 – When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Fade to black", the later one, in output order, is darker than the previous one.

"Fade from black" indicates that the target pictures are part of a sequence of pictures, in output order, involved in a fade from black scene transition, i.e., the luma samples of the scene gradually diverge from zero and the chroma samples of the scene may gradually diverge from 128.

NOTE 4 – When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Fade from black", the later one in output order is lighter than the previous one.

"Dissolve" indicates that the sample values of each target picture (before encoding) were generated by calculating a sum of co-located weighted sample values of a picture from the current scene and a picture from the next scene. The weight of the current scene gradually decreases from full level to zero level, whereas the weight of the next scene gradually increases from zero level to full level. When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Dissolve", the weight of the current scene for the later one, in output order, is less than the weight of the current scene for the previous one, and the weight of the next scene for the later one, in output order, is greater than the weight of the next scene for the previous one.

"Wipe" indicates that some of the sample values of each target picture (before encoding) were generated by copying co-located sample values of a picture in the current scene and the remaining sample values of each target picture (before encoding) were generated by copying co-located sample values of a picture in the next scene. When two pictures are labelled to belong to the same scene transition and their scene_transition_type is "Wipe", the number of samples copied from the next scene to the later picture in output order is greater than the number of samples copied from the next scene to the previous picture.

second_scene_id identifies the next scene in the gradual scene transition in which the target pictures are involved. The value of second_scene_id shall not be equal to the value of scene_id. The value of second_scene_id shall not be equal to the value of scene_id in the previous picture in output order. When the next picture in output order is marked with a value of scene_transition_type less than 4, and the value of second_scene_id is the same as the value of scene_id of the next picture in output order, this indicates that the encoder considers one of the source scenes for the target pictures and the source scene for the next picture (in output order) to have been the same scene. When the value of second_scene_id is not equal to the value of scene_id or second_scene_id (if present) of the next picture in output order, this indicates that the encoder considers the target pictures and the next picture (in output order) to have been from different source scenes.

When the value of scene_id of a picture is equal to the value of scene_id of the following picture in output order and the value of scene_transition_type in both of these pictures is less than 4, this indicates that the encoder considers the two pictures to have been from the same source scene. When the values of scene_id, scene_transition_type and second_scene_id (if present) of a picture are equal to the values of scene_id, scene_transition_type and second_scene_id (respectively) of the following picture in output order and the value of scene_transition_type is greater than 0, this indicates that the encoder considers the two pictures to have been from the same source gradual scene transition.

The value of second_scene_id shall be in the range of 0 to 2^{32}-1, inclusive. Values of second_scene_id in the range of 0 to 255, inclusive, and in the range of 512 to 2^{31}-1, inclusive, may be used as determined by the application. Values of second_scene_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to 2^{32}-1, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of second_scene_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to 2^{32}-1, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.11 Sub-sequence information SEI message semantics

The sub-sequence information SEI message is used to indicate the position of a picture in data dependency hierarchy that consists of sub-sequence layers and sub-sequences.

A sub-sequence layer contains a subset of the coded pictures in a sequence. Sub-sequence layers are numbered with non-negative integers. A layer having a larger layer number is a higher layer than a layer having a smaller layer number. The layers are ordered hierarchically based on their dependency on each other so that any picture in a layer shall not be predicted from any picture on any higher layer.

NOTE 1 – In other words, any picture in layer 0 must not be predicted from any picture in layer 1 or above, pictures in layer 1 may be predicted from layer 0, pictures in layer 2 may be predicted from layers 0 and 1, etc.

NOTE 2 – The subjective quality is expected to increase along with the number of decoded layers.

A sub-sequence is a set of coded pictures within a sub-sequence layer. A picture shall reside in one sub-sequence layer and in one sub-sequence only. Any picture in a sub-sequence shall not be predicted from any picture in another sub-sequence in the same or in a higher sub-sequence layer. A sub-sequence in layer 0 can be decoded independently of any picture that does not belong to the sub-sequence.

The sub-sequence information SEI message concerns the current access unit. The primary coded picture in the access unit is herein referred to as the current picture.

The sub-sequence information SEI message shall not be present unless gaps_in_frame_num_value_allowed_flag in the sequence parameter set referenced by the picture associated with the sub-sequence SEI message is equal to 1.

sub_seq_layer_num specifies the sub-sequence layer number of the current picture. When sub_seq_layer_num is greater than 0, memory management control operations shall not be used in any slice header of the current picture. When the current picture resides in a sub-sequence whose first picture in decoding order is an IDR picture, the value of sub_seq_layer_num shall be equal to 0. For a non-paired reference field, the value of sub_seq_layer_num shall be equal to 0. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

sub_seq_id identifies the sub-sequence within a layer. When the current picture resides in a sub-sequence whose first picture in decoding order is an IDR picture, the value of sub_seq_id shall be the same as the value of idr_pic_id of the IDR picture. sub_seq_id shall be in the range of 0 to 65535, inclusive.

first_ref_pic_flag equal to 1 specifies that the current picture is the first reference picture of the sub-sequence in decoding order. When the current picture is not the first picture of the sub-sequence in decoding order, the first_ref_pic_flag shall be equal to 0.

leading_non_ref_pic_flag equal to 1 specifies that the current picture is a non-reference picture preceding any reference picture in decoding order within the sub-sequence or that the sub-sequence contains no reference pictures. When the current picture is a reference picture or the current picture is a non-reference picture succeeding at least one reference picture in decoding order within the sub-sequence, the leading_non_ref_pic_flag shall be equal to 0.

last_pic_flag equal to 1 indicates that the current picture is the last picture of the sub-sequence (in decoding order), including all reference and non-reference pictures of the sub-sequence. When the current picture is not the last picture of the sub-sequence (in decoding order), last_pic_flag shall be equal to 0.

The current picture is assigned to a sub-sequence as follows.

- If one or more of the following conditions is true, the current picture is the first picture of a sub-sequence in decoding order.
  - no earlier picture in decoding order is labelled with the same values of sub_seq_id and sub_seq_layer_num as the current picture
  - the value of leading_non_ref_pic_flag is equal to 1 and the value of leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture
  - the value of first_ref_pic_flag is equal to 1 and the value of leading_non_ref_pic_flag is equal to 0 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture
  - the value of last_pic_flag is equal to 1 in the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture
- Otherwise, the current picture belongs to the same sub-sequence as the previous picture in decoding order having the same values of sub_seq_id and sub_seq_layer_num as the current picture.

sub_seq_frame_num_flag equal to 0 specifies that sub_seq_frame_num is not present. sub_seq_frame_num_flag equal to 1 specifies that sub_seq_frame_num is present.

sub_seq_frame_num shall be equal to 0 for the first reference picture of the sub-sequence and for any non-reference picture preceding the first reference picture of the sub-sequence in decoding order. sub_seq_frame_num is further constrained as follows.

- If the current picture is not the second field of a complementary field pair, sub_seq_frame_num shall be incremented by 1, in modulo MaxFrameNum operation, relative to the previous reference picture, in decoding order, that belongs to the sub-sequence.

– Otherwise (the current picture is the second field of a complementary field pair), the value of sub_seq_frame_num shall be the same as the value of sub_seq_frame_num for the first field of the complementary field pair.

sub_seq_frame_num shall be in the range of 0 to MaxFrameNum – 1, inclusive.

When the current picture is an IDR picture, it shall start a new sub-sequence in sub-sequence layer 0. Thus, the sub_seq_layer_num shall be 0, the sub_seq_id shall be different from the previous sub-sequence in sub-sequence layer 0, first_ref_pic_flag shall be 1, and leading_non_ref_pic_flag shall be equal to 0.

When the sub-sequence information SEI message is present for both coded fields of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, when present, shall be identical for both of these pictures.

When the sub-sequence information SEI message is present only for one coded field of a complementary field pair, the values of sub_seq_layer_num, sub_seq_id, leading_non_ref_pic_flag and sub_seq_frame_num, when present, are also applicable to the other coded field of the complementary field pair.

D.2.12 Sub-sequence layer characteristics SEI message semantics

The sub-sequence layer characteristics SEI message specifies the characteristics of sub-sequence layers.

num_sub_seq_layers_minus1 plus 1 specifies the number of sub-sequence layers in the sequence. num_sub_seq_layers_minus1 shall be in the range of 0 to 255, inclusive.

A pair of average_bit_rate and average_frame_rate characterizes each sub-sequence layer. The first pair of average_bit_rate and average_frame_rate specifies the characteristics of sub-sequence layer 0. When present, the second pair specifies the characteristics of sub-sequence layers 0 and 1 jointly. Each pair in decoding order specifies the characteristics for a range of sub-sequence layers from layer number 0 to the layer number specified by the layer loop counter. The values are in effect from the point they are decoded until an update of the values is decoded.

accurate_statistics_flag equal to 1 indicates that the values of average_bit_rate and average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate somewhat from the correct values.

When accurate_statistics_flag is equal to 0, the quality of the approximation used in the computation of the values of average_bit_rate and the average_frame_rate is chosen by the encoding process and is not specified by this Recommendation | International Standard.

average_bit_rate indicates the average bit rate in units of 1000 bits per second. All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. The average bit rate is derived according to the access unit removal time specified in Annex C of the Recommendation | International Standard. In the following, bTotal is the number of bits in all NAL units succeeding a sub-sequence layer characteristics SEI message (including the bits of the NAL units of the current access unit) and preceding the next access unit (in decoding order) including a sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise). t_1 is the removal time (in seconds) of the current access unit, and t_2 is the removal time (in seconds) of the latest access unit (in decoding order) before the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows.

– If t_1 is not equal to t_2, the following condition shall be true

$$\text{average_bit_rate} == \text{Round}(\text{bTotal} \div ((t_2 - t_1) * 1000))) \quad \text{(D-6)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true

$$\text{average_bit_rate} == 0 \quad \text{(D-7)}$$

average_frame_rate indicates the average frame rate in units of frames/(256 seconds). All NAL units in the range of sub-sequence layers specified above are taken into account in the calculation. In the following, fTotal is the number of frames, complementary field pairs and non-paired fields between the current picture (inclusive) and the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise). t_1 is the removal time (in seconds) of the current access unit, and t_2 is the removal time (in seconds) of the latest access unit (in decoding order) before the next sub-sequence layer characteristics SEI message (if present) or the end of the stream (otherwise).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows.

– If t_1 is not equal to t_2, the following condition shall be true

$$\text{average_frame_rate} == \text{Round}(\text{fTotal} * 256 \div (t_2 - t_1)) \quad \text{(D-8)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true

$$\text{average_frame_rate} == 0 \quad \text{(D-9)}$$

D.2.13 Sub-sequence characteristics SEI message semantics

The sub-sequence characteristics SEI message indicates the characteristics of a sub-sequence. It also indicates inter prediction dependencies between sub-sequences. This message shall be contained in the first access unit in decoding order of the sub-sequence to which the sub-sequence characteristics SEI message applies. This sub-sequence is herein called the target sub-sequence.

sub_seq_layer_num identifies the sub-sequence layer number of the target sub-sequence. sub_seq_layer_num shall be in the range of 0 to 255, inclusive.

sub_seq_id identifies the target sub-sequence. sub_seq_id shall be in the range of 0 to 65535, inclusive.

duration_flag equal to 0 indicates that the duration of the target sub-sequence is not specified.

sub_seq_duration specifies the duration of the target sub-sequence in clock ticks of a 90-kHz clock.

average_rate_flag equal to 0 indicates that the average bit rate and the average frame rate of the target sub-sequence are unspecified.

accurate_statistics_flag indicates how reliable the values of average_bit_rate and average_frame_rate are. accurate_statistics_flag equal to 1, indicates that the average_bit_rate and the average_frame_rate are rounded from statistically correct values. accurate_statistics_flag equal to 0 indicates that the average_bit_rate and the average_frame_rate are estimates and may deviate from the statistically correct values.

average_bit_rate indicates the average bit rate in (1000 bits)/second of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average bit rate is derived according to the access unit removal time specified in subclause C.1.2. In the following, nB is the number of bits in all NAL units in the sub-sequence. t_1 is the removal time (in seconds) of the first access unit of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last access unit of the sub-sequence (in decoding order).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows.

– If t_1 is not equal to t_2, the following condition shall be true

$$\text{average_bit_rate} == \text{Round}(\text{nB} \div ((t_2 - t_1) * 1000)) \quad \text{(D-10)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true

$$\text{average_bit_rate} == 0 \quad \text{(D-11)}$$

average_frame_rate indicates the average frame rate in units of frames/(256 seconds) of the target sub-sequence. All NAL units of the target sub-sequence are taken into account in the calculation. The average frame rate is derived according to the access unit removal time specified in subclause C.1.2. In the following, fC is the number of frames, complementary field pairs and non-paired fields in the sub-sequence. t_1 is the removal time (in seconds) of the first access unit of the sub-sequence (in decoding order), and t_2 is the removal time (in seconds) of the last access unit of the sub-sequence (in decoding order).

When accurate_statistics_flag is equal to 1, the following conditions shall be fulfilled as follows.

– If t_1 is not equal to t_2, the following condition shall be true

$$\text{average_frame_rate} == \text{Round}(\text{fC} * 256 \div (t_2 - t_1)) \quad \text{(D-12)}$$

– Otherwise (t_1 is equal to t_2), the following condition shall be true

average_frame_rate = = 0 (D-13)

num_referenced_subseqs specifies the number of sub-sequences that contain pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. num_referenced_subseqs shall be in the range of 0 to 255, inclusive.

ref_sub_seq_layer_num, ref_sub_seq_id, and **ref_sub_seq_direction** identify the sub-sequence that contains pictures that are used as reference pictures for inter prediction in the pictures of the target sub-sequence. Depending on ref_sub_seq_direction, the following applies.

– If ref_sub_seq_direction is equal to 0, a set of candidate sub-sequences consists of the sub-sequences whose sub_seq_id is equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and whose first picture in decoding order precedes the first picture of the target sub-sequence in decoding order.

– Otherwise (ref_sub_seq_direction is equal to 1), a set of candidate sub-sequences consists of the sub-sequences whose sub_seq_id is equal to ref_sub_seq_id, which reside in the sub-sequence layer having sub_seq_layer_num equal to ref_sub_seq_layer_num, and whose first picture in decoding order succeeds the first picture of the target sub-sequence in decoding order.

The sub-sequence used as a reference for the target sub-sequence is the sub-sequence among the set of candidate sub-sequences whose first picture is the closest to the first picture of the target sub-sequence in decoding order.

D.2.14 Full-frame freeze SEI message semantics

The full-frame freeze SEI message indicates that the current picture and any subsequent pictures in output order that meet specified conditions should not affect the content of the display. No more than one full-frame freeze SEI message shall be present in any access unit.

full_frame_freeze_repetition_period specifies the persistence of the full-frame freeze SEI message and may specify a picture order count interval within which another full-frame freeze SEI message or a full-frame freeze release SEI or the end of the coded video sequence shall be present in the bitstream. The value of full_frame_freeze_repetition_period shall be in the range of 0 to 16 384, inclusive.

full_frame_freeze_repetition_period equal to 0 specifies that the full-frame freeze SEI message applies to the current decoded picture only.

full_frame_freeze_repetition_period equal to 1 specifies that the full-frame freeze SEI message persists in output order until any of the following conditions are true.

– A new coded video sequence begins

– A picture in an access unit containing a full-frame freeze SEI message or a full-frame freeze release SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

full_frame_freeze_repetition_period greater than 1 specifies that the full-frame freeze SEI message persists until any of the following conditions are true.

– A new coded video sequence begins

– A picture in an access unit containing a full-frame freeze SEI message or a full-frame freeze release SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + full_frame_freeze_repetition_period.

full_frame_freeze_repetition_period greater than 1 indicates that another full-frame freeze SEI message or a full-frame freeze release SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + full_frame_freeze_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.15 Full-frame freeze release SEI message semantics

The full-frame freeze release SEI message cancels the effect of any full-frame freeze SEI message sent with pictures that precede the current picture in output order. The full-frame freeze release SEI message indicates that the current picture and subsequent pictures in output order should affect the contents of the display.

No more than one full-frame freeze release SEI message shall be present in any access unit. A full-frame freeze release SEI message shall not be present in an access unit containing a full-frame freeze SEI message. When a full-frame freeze SEI message is present in an access unit containing a field of a complementary field pair in which the values of PicOrderCnt(CurrPic) for the two fields of the complementary field pair are equal to each other, a full-frame freeze release SEI message shall not be present in either of the two access units.

D.2.16 Full-frame snapshot SEI message semantics

The full-frame snapshot SEI message indicates that the current frame is labelled for use as determined by the application as a still-image snapshot of the video content.

snapshot_id specifies a snapshot identification number. snapshot_id shall be in the range of 0 to 2^{32} - 1, inclusive.

Values of snapshot_id in the range of 0 to 255, inclusive, and in the range of 512 to 2^{31}-1, inclusive, may be used as determined by the application. Values of snapshot_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to 2^{32}-1, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of snapshot_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to 2^{32}-1, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.17 Progressive refinement segment start SEI message semantics

The progressive refinement segment start SEI message specifies the beginning of a set of consecutive coded pictures that is labelled as the current picture followed by a sequence of one or more pictures of refinement of the quality of the current picture, rather than as a representation of a continually moving scene.

The tagged set of consecutive coded pictures shall continue until one of the following conditions is true. When a condition below becomes true, the next slice to be decoded does not belong to the tagged set of consecutive coded pictures.

1. The next slice to be decoded belongs to an IDR picture.
2. num_refinement_steps_minus1 is greater than 0 and the frame_num of the next slice to be decoded is (currFrameNum + num_refinement_steps_minus1 + 1) % MaxFrameNum, where currFrameNum is the value of frame_num of the picture in the access unit containing the SEI message.
3. num_refinement_steps_minus1 is 0 and a progressive refinement segment end SEI message with the same progressive_refinement_id as the one in this SEI message is decoded.

The decoding order of picture within the tagged set of consecutive pictures should be the same as their output order. **progressive_refinement_id** specifies an identification number for the progressive refinement operation. progressive_refinement_id shall be in the range of 0 to 2^{32} - 1, inclusive.

Values of progressive_refinement_id in the range of 0 to 255, inclusive, and in the range of 512 to 2^{31} - 1, inclusive, may be used as determined by the application. Values of progressive_refinement_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to 2^{32}-1, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to 2^{32} - 1, inclusive, shall ignore (remove from the bitstream and discard) it.

num_refinement_steps_minus1 specifies the number of reference frames in the tagged set of consecutive coded pictures as follows.

– If num_refinement_steps_minus1 is equal to 0, the number of reference frames in the tagged set of consecutive coded pictures is unknown.

– Otherwise, the number of reference frames in the tagged set of consecutive coded pictures is equal to num_refinement_steps_minus1 + 1.

num_refinement_steps_minus1 shall be in the range of 0 to MaxFrameNum - 1, inclusive.

D.2.18 Progressive refinement segment end SEI message semantics

The progressive refinement segment end SEI message specifies the end of a set of consecutive coded pictures that has been labelled by use of a progressive refinement segment start SEI message as an initial picture followed by a sequence of one or more pictures of the refinement of the quality of the initial picture, and ending with the current picture.

progressive_refinement_id specifies an identification number for the progressive refinement operation. progressive_refinement_id shall be in the range of 0 to 2^{32} - 1, inclusive.

The progressive refinement segment end SEI message specifies the end of any progressive refinement segment previously started using a progressive refinement segment start SEI message with the same value of progressive_refinement_id.

Values of progressive_refinement_id in the range of 0 to 255, inclusive, and in the range of 512 to 2^{31} - 1, inclusive, may be used as determined by the application. Values of progressive_refinement_id in the range of 256 to 511, inclusive, and in the range of 2^{31} to 2^{32} - 1, inclusive, are reserved for future use by ITU-T | ISO/IEC. Decoders encountering a value of progressive_refinement_id in the range of 256 to 511, inclusive, or in the range of 2^{31} to 2^{32} - 1, inclusive, shall ignore (remove from the bitstream and discard) it.

D.2.19 Motion-constrained slice group set SEI message semantics

This SEI message indicates that inter prediction over slice group boundaries is constrained as specified below. When present, the message shall only appear where it is associated, as specified in subclause 7.4.1.2.3, with an IDR access unit.

The target picture set for this SEI message contains all consecutive primary coded pictures in decoding order starting with the associated primary coded IDR picture (inclusive) and ending with the following primary coded IDR picture (exclusive) or with the very last primary coded picture in the bitstream (inclusive) in decoding order when there is no following primary coded IDR picture. The slice group set is a collection of one or more slice groups, identified by the slice_group_id[i] syntax element.

This SEI message indicates that, for each picture in the target picture set, the inter prediction process is constrained as follows: No sample value outside the slice group set, and no sample value at a fractional sample position that is derived using one or more sample values outside the slice group set is used to inter predict any sample within the slice group set.

num_slice_groups_in_set_minus1 + 1 specifies the number of slice groups in the slice group set. The allowed range of num_slice_groups_in_set_minus1 is 0 to num_slice_groups_minus1, inclusive. The allowed range of num_slice_groups_minus1 is specified in Annex A.

slice_group_id[i **]** identifies the slice group(s) contained within the slice group set. The allowed range is from 0 to num_slice_groups_in_set_minus1, inclusive. The size of the slice_group_id[i] syntax element is Ceil(Log2(num_slice_groups_minus1 + 1)) bits.

exact_sample_value_match_flag equal to 0 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set need not be exactly the same as the value of the same sample when all the macroblocks are decoded. exact_sample_value_match_flag equal to 1 indicates that, within the target picture set, when the macroblocks that do not belong to the slice group set are not decoded, the value of each sample in the slice group set shall be exactly the same as the value of the same sample when all the macroblocks in the target picture set are decoded.

NOTE 1 – When disable_deblocking_filter_idc is equal to 2 in all slices in the target picture set, exact_sample_value_match_flag should be 1.

pan_scan_rect_flag equal to 0 specifies that pan_scan_rect_id is not present. pan_scan_rect_flag equal to 1 specifies that pan_scan_rect_id is present.

pan_scan_rect_id indicates that the specified slice group set covers at least the pan-scan rectangle identified by pan_scan_rect_id within the target picture set.

NOTE 2 – Multiple motion_constrained_slice_group_set SEI messages may be associated with the same IDR picture. Consequently, more than one slice group set may be active within a target picture set.

NOTE 3 – The size, shape, and location of the slice groups in the slice group set may change within the target picture set.

D.2.20 Film grain characteristics SEI message semantics

This SEI message provides the decoder with a parameterised model for film grain synthesis. For example, an encoder may use the film grain characteristics SEI message to characterise film grain that was present in the original source video material and was removed by pre-processing filtering techniques. Synthesis of simulated film grain on the decoded images for the display process is optional and does not affect the decoding process specified in this Recommendation | International Standard. If synthesis of simulated film grain on the decoded images for the display process is performed, there is no requirement that the method by which the synthesis is performed be the same as the parameterised model for the film grain as provided in the film grain characteristics SEI message.

NOTE 1 – The display process is not specified in this Recommendation | International Standard.

film_grain_characteristics_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous film grain characteristics SEI message in output order. film_grain_characteristics_cancel_flag equal to 0 indicates that film grain modelling information follows.

model_id identifies the film grain simulation model as specified in Table D-5. The value of model_id shall be in the range of 0 to 1, inclusive.

Table D-5 – model_id values

Value	Description
0	frequency filtering
1	auto-regression
2	reserved
3	reserved

separate_colour_description_present_flag equal to 1 indicates that a distinct colour space description for the film grain characteristics specified in the SEI message is present in the film grain characteristics SEI message syntax. separate_colour_description_present_flag equal to 0 indicates that the colour description for the film grain characteristics specified in the SEI message is the same as for the coded video sequence as specified in subclause E.2.1.

NOTE 2 – When separate_colour_description_present_flag is equal to 1, the colour space specified for the film grain characteristics specified in the SEI message may differ from the colour space specified for the coded video as specified in subclause E.2.1.

film_grain_bit_depth_luma_minus8 plus 8 specifies the bit depth used for the luma component of the film grain characteristics specified in the SEI message. When film_grain_bit_depth_luma_minus8 is not present in the film grain characteristics SEI message, the value of film_grain_bit_depth_luma_minus8 shall be inferred to be equal to bit_depth_luma_minus8.

The value of filmGrainBitDepth[0] is derived as

$$filmGrainBitDepth[\ 0\ ] = film_grain_bit_depth_luma_minus8 + 8 \qquad \text{(D-14)}$$

film_grain_bit_depth_chroma_minus8 plus 8 specifies the bit depth used for the Cb and Cr components of the film grain characteristics specified in the SEI message. When film_grain_bit_depth_chroma_minus8 is not present in the film grain characteristics SEI message, the value of film_grain_bit_depth_chroma_minus8 shall be inferred to be equal to bit_depth_chroma_minus8.

The value of filmGrainBitDepth[c] for c = 1 and 2 is derived as

$$filmGrainBitDepth[\ c\ ] = film_grain_bit_depth_chroma_minus8 + 8 \quad \text{with } c = 1, 2 \qquad \text{(D-15)}$$

film_grain_full_range_flag has the same semantics as specified in subclause E.2.1 for the video_full_range_flag syntax element, except as follows.

– film_grain_full_range_flag specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.

– When film_grain_full_range_flag is not present in the film grain characteristics SEI message, the value of film_grain_full_range_flag shall be inferred to be equal to video_full_range_flag.

film_grain_colour_primaries has the same semantics as specified in subclause E.2.1 for the colour_primaries syntax element, except as follows.

– film_grain_colour_primaries specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.

– When film_grain_colour_primaries is not present in the film grain characteristics SEI message, the value of film_grain_colour_primaries shall be inferred to be equal to colour_primaries.

film_grain_transfer_characteristics has the same semantics as specified in subclause E.2.1 for the transfer_characteristics syntax element, except as follows.

– film_grain_transfer_characteristics specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.

– When film_grain_transfer_characteristics is not present in the film grain characteristics SEI message, the value of film_grain_transfer_characteristics shall be inferred to be equal to transfer_characteristics.

film_grain_matrix_coefficients has the same semantics as specified in subclause E.2.1 for the matrix_coefficients syntax element, except as follows.

– film_grain_matrix_coefficients specifies the colour space of the film grain characteristics specified in the SEI message, rather than the colour space used for the coded video sequence.

– When film_grain_matrix_coefficients is not present in the film grain characteristics SEI message, the value of film_grain_matrix_coefficients shall be inferred to be equal to matrix_coefficients.

– The values allowed for film_grain_matrix_coefficients are not constrained by the value of chroma_format_idc.

The chroma_format_idc of the film grain characteristics specified in the film grain characteristics SEI message shall be inferred to be equal to 3 (4:4:4).

NOTE 3 – Because the use of a specific method is not required for performing film grain generation function used by the display process, a decoder may, if desired, down-convert the model information for chroma in order to simulate film grain for other chroma formats (4:2:0 or 4:2:2) rather than up-converting the decoded video (using a method not specified by this Recommendation | International Standard) before performing film grain generation.

blending_mode_id identifies the blending mode used to blend the simulated film grain with the decoded images as specified in Table D-6. blending_mode_id shall be in the range of 0 to 1, inclusive.

Table D-6 – blending_mode_id values

Value	Description
0	additive
1	multiplicative
2	reserved
3	reserved

Depending on blending_mode_id, the blending mode is specified as follows

– If blending_mode_id is equal to 0 the blending mode is additive as specified by

$$I_{grain}[\ x, y, c\ ] = \mathrm{Clip3}(\ 0, (\ 1 << \mathrm{filmGrainBitDepth}[\ c\ ]\ ) - 1, I_{decoded}[\ x, y, c\ ] + G[\ x, y, c\ ]\ ) \qquad \text{(D-16)}$$

– Otherwise (blending_mode_id is equal to 1), the blending mode is multiplicative as specified by

$$I_{grain}[\ x, y, c\ ] = \mathrm{Clip3}(\ 0, (\ 1 << \mathrm{filmGrainBitDepth}[\ c\ ]\ ) - 1, I_{decoded}[\ x, y, c\ ] * (\ 1 + G[\ x, y, c\ ]\ )\ ) \qquad \text{(D-17)}$$

where $I_{decoded}[\ x, y, c\ ]$ represents the sample value at coordinates x, y of the colour component c of the decoded image $I_{decoded}$, $G[\ x, y, c\ ]$ is the simulated film grain value at the same position and colour component, and filmGrainBitDepth[c] is the number of bits used for each sample in a fixed-length unsigned binary representation of the array $I_{grain}[\ x, y, c\ ]$.

log2_scale_factor specifies a scale factor used in the film grain characterization equations.

comp_model_present_flag[c] equal to 0 indicates that film grain is not modelled on the c-th colour component, where c equal to 0 refers to the luma component, c equal to 1 refers to the Cb component, and c equal to 2 refers to the Cr component. comp_model_present_flag[c] equal to 1 indicates that syntax elements specifying modelling of film grain on colour component c are present in the SEI message.

num_intensity_intervals_minus1[c] plus 1 specifies the number of intensity intervals for which a specific set of model values has been estimated.

NOTE 4 – The intensity intervals may overlap in order to simulate multi-generational film grain.

num_model_values_minus1[c] plus 1 specifies the number of model values present for each intensity interval in which the film grain has been modelled. The value of num_model_values_minus1[c] shall be in the range of 0 to 5, inclusive.

intensity_interval_lower_bound[c][i] specifies the lower bound of the interval i of intensity levels for which the set of model values applies.

intensity_interval_upper_bound[c][i] specifies the upper bound of the interval i of intensity levels for which the set of model values applies.

Depending on model_id, the selection of the sets of model values is specified as follows.

– If model_id is equal to 0, the average value of each block b of 16x16 samples in $I_{decoded}$, referred as b_{avg}, is used to select the sets of model values with index s[j] that apply to all the samples in the block:

```
for( i = 0, j = 0; i <= num_intensity_intervals_minus1; i++ )
    if( b_avg >= intensity_interval_lower_bound[ c ][ i ] && b_avg <= intensity_interval_upper_bound[ c ][ i ] ) {
      s[ j ] = i                                                                                    (D-18)
      j++
    }
```

– Otherwise (model_id is equal to 1), the sets of model values used to generate the film grain are selected for each sample value in $I_{decoded}$ as follows:

```
for( i = 0, j = 0; i <= num_intensity_intervals_minus1; i++ )
    if( I_decoded[ x, y, c ] >= intensity_interval_lower_bound[ c ][ i ] &&
        I_decoded[ x, y, c ] <= intensity_interval_upper_bound[ c ][ i ] ) {                        (D-19)
      s[ j ] = i
      j++
    }
```

Samples that do not fall into any of the defined intervals are not modified by the grain generation function. Samples that fall into more than one interval will originate multi-generation grain. Multi-generation grain results from adding the grain computed independently for each intensity interval.

comp_model_value[c][i][j] represents each one of the model values present for the colour component c and the intensity interval i. The set of model values has different meaning depending on the value of model_id. The value of comp_model_value[c][i][j] shall be constrained as follows, and may be additionally constrained as specified elsewhere in this subclause.

– If model_id is equal to 0, comp_model_value[c][i][j] shall be in the range of 0 to $2^{filmGrainBitDepth[\,c\,]} - 1$, inclusive.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][j] shall be in the range of $-2^{(\,filmGrainBitDepth[\,c\,] - 1\,)}$ to $2^{(\,filmGrainBitDepth[\,c\,] - 1\,)} - 1$, inclusive.

Depending on model_id, the synthesis of the film grain is modelled as follows.

– If model_id is equal to 0, a frequency filtering model enables simulating the original film grain for c = 0..2, x = 0..$PicWidthInSamples_L$, and y = 0..$PicHeightInSamples_L$ as specified by:

```
G[ x, y, c ] = ( comp_model_value[ c ][ s ][ 0 ] * Q[ x, y, c ] + comp_model_value[ c ][ s ][ 5 ] *
G[ x, y, c-1 ] ) >> log2_scale_factor                                                               (D-20)
```

where Q[c] is a two-dimensional random process generated by filtering 16x16 blocks gaussRv with random-value elements $gaussRv_{ij}$ generated with a normalized Gaussian distribution (independent and identically distributed Gaussian random variable samples with zero mean and unity variance) and where the value of an element G[x, y, c-1] used in the right-hand side of the equation is inferred to be equal to 0 when c-1 is less than 0.

NOTE 5 – A normalized Gaussian random value can be generated from two independent, uniformly distributed random values over the interval from 0 to 1 (and not equal to 0), denoted as uRv_0 and uRv_1, using the Box-Muller transformation specified by

$$\text{gaussRv}_{ij} = \sqrt{-2*\text{Ln}(\text{uRv}_0)}*\text{Cos}(2*\pi*\text{uRv}_1) \tag{D-21}$$

where Ln(x) is the natural logarithm of x (the base-e logarithm, where e is natural logarithm base constant 2.718 281 828...), Cos(x) is the trigonometric cosine function operating on an argument x in units of radians, and π is Archimedes' constant 3.141 592 653....

The band-pass filtering of blocks gaussRv may be performed in the discrete cosine transform (DCT) domain as follows:

```
for( y = 0; y < 16; y++ )
    for( x = 0; x < 16; x++ )
        if( ( x < comp_model_value[ c ][ s ][ 3 ]  &&  y < comp_model_value[ c ][ s ][ 4 ] )  ||
          x > comp_model_value[ c ][ s ][ 1 ]  ||  y > comp_model_value[ c ][ s ][ 2 ] )
            gaussRv[ x, y ] = 0
filteredRv = IDCT16x16( gaussRv )
```
(D-22)

where IDCT16x16(z) refers to a unitary inverse discrete cosine transformation (IDCT) operating on a 16x16 matrix argument z as specified by

$$\text{IDCT16x16}(z) = r * z * r^T \tag{D-23}$$

where the superscript T indicates a matrix transposition and r is the 16x16 matrix with elements r_{ij} specified by

$$r_{ij} = \frac{((i == 0)\,?\ 1 : \sqrt{2})}{4}\text{Cos}\left(\frac{i*(2*j+1)*\pi}{32}\right) \tag{D-24}$$

where Cos(x) is the trigonometric cosine function operating on an argument x in units of radians and π is Archimedes' constant 3.141 592 653.

Q[c] is formed by the frequency-filtered blocks filteredRv.

NOTE 6 – Coded model values are based on blocks of 16x16, but a decoder implementation may use other block sizes. For example, decoders implementing the IDCT on 8x8 blocks, should down-convert by a factor of two the set of coded model values comp_model_value[c][s][i] for i equal to 1..4.

NOTE 7 – To reduce the degree of visible blocks that can result from mosaicing the frequency-filtered blocks filteredRv, decoders may apply a low-pass filter to the boundaries between frequency-filtered blocks.

– Otherwise (model_id is equal to 1), an auto-regression model enables simulating the original film grain for c = 0..2, x = 0..$\text{PicWidthInSamples}_L$, and y = 0..$\text{PicHeightInSamples}_L$ as specified by

```
G[ x, y, c ] = ( comp_model_value[ c ][ s ][ 0 ] * n[ x, y, c ] +
    comp_model_value[ c ][ s ][ 1 ] * ( G[ x-1, y, c ] + ( ( comp_model_value[ c ][ s ][ 4 ] * G[ x, y-1, c ] )  >>
        log2_scale_factor ) ) +
    comp_model_value[ c ][ s ][ 3 ] * ( ( ( comp_model_value[ c ][ s ][ 4 ] * G[ x-1, y-1, c ] ) >>
        log2_scale_factor ) + G[ x+1, y-1, c ] ) +
    comp_model_value[ c ][ s ][ 5 ] * ( G[ x-2, y, c ] +
        ( ( comp_model_value[ c ][ s ][ 4 ] * comp_model_value[ c ][ s ][ 4 ] * G[ x, y-2, c ] ) >>
            ( 2 * log2_scale_factor ) ) ) +
    comp_model_value[ c ][ s ][ 2 ] * G[ x, y, c-1 ] ) >> log2_scale_factor
```
(D-25)

where n[x, y, c] is a random value with normalized Gaussian distribution (independent and identically distributed Gaussian random variable samples with zero mean and unity variance for each value of x, y, and c) and where the value of an element G[x, y, c] used in the right-hand side of the equation is inferred to be equal to 0 when any of the following conditions are true:

– x is less than 0,

– y is less than 0,

– x is greater than or equal to $\text{PicWidthInSamples}_L$,

– c is less than 0.

comp_model_value[c][i][0] provides the first model value for the model as specified by model_id. comp_model_value[c][i][0] corresponds to the standard deviation of the Gaussian noise term in the generation functions specified in Equations D-20 through D-23.

comp_model_value[c][i][1] provides the second model value for the model as specified by model_id. comp_model_value[c][i][1] shall be greater than or equal to 0 and less than 16.

When not present in the film grain characteristics SEI message, comp_model_value[c][i][1] shall be inferred as follows.

– If model_id is equal to 0, comp_model_value[c][i][1] shall be inferred to be equal to 8.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][1] shall be inferred to be equal to 0.

comp_model_value[c][i][1] is interpreted as follows.

– If model_id is equal to 0, comp_model_value[c][i][1] indicates the horizontal high cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][1] indicates the first order spatial correlation for neighbouring samples (x-1, y) and (x, y-1).

comp_model_value[c][i][2] provides the third model value for the model as specified by model_id. comp_model_value[c][i][2] shall be greater than or equal to 0 and less than 16.

When not present in the film grain characteristics SEI message, comp_model_value[c][i][2] shall be inferred as follows.

– If model_id is equal to 0, comp_model_value[c][i][2] shall be inferred to be equal to comp_model_value[c][i][1]

– Otherwise (model_id is equal to 0), comp_model_value[c][i][2] shall be inferred to be equal to 0.

comp_model_value[c][i][2] is interpreted as follows.

– If model_id is equal to 0, comp_model_value[c][i][2] indicates the vertical high cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][2] indicates the colour correlation between consecutive colour components.

comp_model_value[c][i][3] provides the fourth model value for the model as specified by model_id. comp_model_value[c][i][3] shall be greater than or equal to 0 and less than or equal to comp_model_value[c][i][1].

When not present in the film grain characteristics SEI message, comp_model_value[c][i][3] shall be inferred to be equal to 0.

comp_model_value[c][i][3] is interpreted as follows.

– If model_id is equal to 0, comp_model_value[c][i][3] indicates the horizontal low cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][3] indicates the first order spatial correlation for neighbouring samples (x-1, y-1) and (x+1, y-1).

comp_model_value[c][i][4] provides the fifth model value for the model as specified by model_id. comp_model_value[c][i][4] shall be greater than or equal to 0 and less than or equal to comp_model_value[c][i][2].

When not present in the film grain characteristics SEI message, comp_model_value[c][i][4] shall be inferred to be equal to model_id.

comp_model_value[c][i][4] is interpreted as follows.

– If model_id is equal to 0, comp_model_value[c][i][4] indicates the vertical low cut frequency to be used to filter the DCT of a block of 16x16 random values.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][4] indicates the aspect ratio of the modelled grain.

comp_model_value[c][i][5] provides the sixth model value for the model as specified by model_id. When not present in the film grain characteristics SEI message, comp_model_value[c][i][5] shall be inferred to be equal to 0.

comp_model_value[c][i][5] is interpreted as follows.

– If model_id is equal to 0, comp_model_value[c][i][5] indicates the colour correlation between consecutive colour components.

– Otherwise (model_id is equal to 1), comp_model_value[c][i][5] indicates the second order spatial correlation for neighbouring samples (x, y-2) and (x-2, y).

film_grain_characteristics_repetition_period specifies the persistence of the film grain characteristics SEI message and may specify a picture order count interval within which another film grain characteristics SEI message or the end of the coded video sequence shall be present in the bitstream. The value of film_grain_characteristics_repetition_period shall be in the range 0 to 16 384, inclusive.

film_grain_characteristics_repetition_period equal to 0 specifies that the film grain characteristics SEI message applies to the current decoded picture only.

film_grain_characteristics_repetition_period equal to 1 specifies that the film grain characteristics SEI message persists in output order until any of the following conditions are true.

– A new coded video sequence begins, or

– A picture in an access unit containing a film grain characteristics SEI message that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

film_grain_characteristics_repetition_period greater than 1 specifies that the film grain characteristics SEI message persists until any of the following conditions are true.

– A new coded video sequence begins, or

– A picture in an access unit containing a film grain characteristics SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + film_grain_characteristics_repetition_period.

film_grain_characteristics_repetition_period greater than 1 indicates that another film grain characteristics SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + film_grain_characteristics_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.21 Deblocking filter display preference SEI message semantics

This SEI message provides the decoder with an indication of whether the display of the cropped result of the deblocking filter process specified in subclause 8.7 or of the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 is preferred by the encoder for the display of each decoded picture that is output.

NOTE 1 – The display process is not specified in this Recommendation | International Standard. The means by which an encoder determines what to indicate as its preference expressed in a deblocking filter display preference SEI message is also not specified in this Recommendation | International Standard, and the expression of an expressed preference in a deblocking filter display preference SEI message does not impose any requirement on the display process.

deblocking_display_preference_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous deblocking filter display preference SEI message in output order. deblocking_display_preference_cancel_flag equal to 0 indicates that a display_prior_to_deblocking_preferred_flag and deblocking_display_preference_repetition_period follow.

NOTE 2 – In the absence of the deblocking filter display preference SEI message, or after the receipt of a deblocking filter display preference SEI message in which deblocking_display_preference_cancel_flag is equal to 1, the decoder should infer that the display of the cropped result of the deblocking filter process specified in subclause 8.7 is preferred over the display of the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 for the display of each decoded picture that is output.

display_prior_to_deblocking_preferred_flag equal to 1 indicates that the encoder preference is for the display process (which is not specified in this Recommendation | International Standard) to display the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 rather than the cropped result of the deblocking filter process specified in subclause 8.7 for each picture that is cropped and output as specified in Annex C. display_prior_to_deblocking_preferred_flag equal to 0 indicates that the encoder preference is for the

display process (which is not specified in this Recommendation | International Standard) to display the cropped result of the deblocking filter process specified in subclause 8.7 rather than the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 for each picture that is cropped and output as specified in Annex C.

NOTE 3 – The presence or absence of the deblocking filter display preference SEI message and the value of display_prior_to_deblocking_preferred_flag does not affect the requirements of the decoding process specified in this Recommendation | International Standard. Rather, it only provides an indication of when, in addition to fulfilling the requirements of this Recommendation | International Standard for the decoding process, enhanced visual quality might be obtained by performing the display process (which is not specified in this Recommendation | International Standard) in an alternative fashion. Encoders that use the deblocking filter display preference SEI message should be designed with an awareness that unless the encoder restricts its use of the DPB capacity specified in Annex A for the profile and level in use, some decoders may not have sufficient memory capacity for the storage of the result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 in addition to the storage of the result of the deblocking filter process specified in subclause 8.7 when reordering and delaying pictures for display, and such decoders would therefore not be able to benefit from the preference indication. By restricting its use of the DPB capacity, an encoder can be able to use at least half of the DPB capacity specified in Annex A while allowing the decoder to use the remaining capacity for storage of unfiltered pictures that have been indicated as preferable for display until the output time arrives for those pictures.

dec_frame_buffering_constraint_flag equal to 1 indicates that the use of the frame buffering capacity of the HRD decoded picture buffer (DPB) as specified by max_dec_frame_buffering has been constrained such that the coded video sequence will not require a decoded picture buffer with more than Max(1, max_dec_frame_buffering) frame buffers to enable the output of the decoded filtered or unfiltered pictures, as indicated by the deblocking filter display preference SEI messages, at the output times specified by the dpb_output_delay of the picture timing SEI messages. dec_frame_buffering_constraint_flag equal to 0 indicates that the use of the frame buffering capacity in the HRD may or may not be constrained in the manner that would be indicated by dec_frame_buffering_constraint_flag equal to 1.

For purposes of determining the constraint imposed when dec_frame_buffering_constraint_flag is equal to 1, the quantity of frame buffering capacity used at any given point in time by each frame buffer of the DPB that contains a picture shall be derived as follows:

- If both of the following criteria are satisfied for the frame buffer, the frame buffer is considered to use two frame buffers of capacity for its storage.
  - The frame buffer contains a frame or one or more fields that is marked as "used for reference", and
  - The frame buffer contains a picture for which both of the following criteria are fulfilled.
    - The HRD output time of the picture is greater than the given point in time, and
    - It has been indicated in a deblocking filter display preference SEI message that the encoder preference for the picture is for the display process to display the cropped result of the picture construction process prior to the deblocking filter process specified in subclause 8.5.12 rather than the cropped result of the deblocking filter process specified in subclause 8.7, and
- Otherwise, the frame buffer is considered to use one frame buffer of DPB capacity for its storage.

When dec_frame_buffering_constraint_flag is equal to 1, the frame buffering capacity used by all of the frame buffers in the DPB that contain pictures, as derived in this manner, shall not be greater than Max(1, max_dec_frame_buffering) during the operation of the HRD for the coded video sequence.

The value of dec_frame_buffering_constraint_flag shall be the same in all deblocking filter display preference SEI messages of the coded video sequence.

deblocking_display_preference_repetition_period specifies the persistence of the film grain characteristics SEI message and may specify a picture order count interval within which another film grain characteristics SEI message or the end of the coded video sequence shall be present in the bitstream. The value of deblocking_display_preference_repetition_period shall be in the range 0 to 16 384, inclusive.

deblocking_display_preference_repetition_period equal to 0 specifies that the deblocking filter display preference SEI message applies to the current decoded picture only.

deblocking_display_preference_repetition_period equal to 1 specifies that the deblocking filter display preference SEI message persists in output order until any of the following conditions are true.

- A new coded video sequence begins
- A picture in an access unit containing a deblocking filter display preference SEI message that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic).

deblocking_display_preference_repetition_period greater than 1 specifies that the deblocking filter display preference SEI message persists until any of the following conditions are true.

– A new coded video sequence begins

– A picture in an access unit containing a deblocking filter display preference SEI message is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + deblocking_display_preference_repetition_period.

deblocking_display_preference_repetition_period greater than 1 indicates that another deblocking filter display preference SEI message shall be present for a picture in an access unit that is output having PicOrderCnt( ) greater than PicOrderCnt(CurrPic) and less than or equal to PicOrderCnt(CurrPic) + deblocking_display_preference_repetition_period; unless the bitstream ends or a new coded video sequence begins without output of such a picture.

D.2.22 Stereo video information SEI message semantics

This SEI message provides the decoder with an indication that the entire coded video sequence consists of pairs of pictures forming stereo-view content.

The stereo video information SEI message shall not be present in any access unit of a coded video sequence unless a stereo video information SEI message is present in the first access unit of the coded video sequence.

field_views_flag equal to 1 indicates that all pictures in the current coded video sequence are fields and all fields of a particular parity are considered a left view and all fields of the opposite parity are considered a right view for stereo-view content. field_views_flag equal to 0 indicates that all pictures in the current coded video sequence are frames and alternating frames in output order represent a view of a stereo view. The value of field_views_flag shall be the same in all stereo video information SEI messages within a coded video sequence.

When the stereo video information SEI message is present and field_views_flag is equal to 1, the left view and right view of a stereo video pair shall be coded as a complementary field pair, the display time of the first field of the field pair in output order should be delayed to coincide with the display time of the second field of the field pair in output order, and the spatial locations of the samples in each individual field should be interpreted for display purposes as representing complete pictures as shown in Figure 6-1 rather than as spatially-distinct fields within a frame as shown in Figure 6-2.

NOTE – The display process is not specified in this Recommendation | International Standard.

top_field_is_left_view_flag equal to 1 indicates that the top fields in the coded video sequence represent a left view and the bottom fields in the coded video sequence represent a right view. top_field_is_left_view_flag equal to 0 indicates that the bottom fields in the coded video sequence represent a left view and the top fields in the coded video sequence represent a right view. When present, the value of top_field_is_left_view_flag shall be the same in all stereo video information SEI messages within a coded video sequence.

current_frame_is_left_view_flag equal to 1 indicates that the current picture is the left view of a stereo-view pair. current_frame_is_left_view_flag equal to 0 indicates that the current picture is the right view of a stereo-view pair.

next_frame_is_second_view_flag equal to 1 indicates that the current picture and the next picture in output order form a stereo-view pair, and the display time of the current picture should be delayed to coincide with the display time of the next picture in output order. next_frame_is_second_view_flag equal to 0 indicates that the current picture and the previous picture in output order form a stereo-view pair, and the display time of the current picture should not be delayed for purposes of stereo-view pairing.

left_view_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the left-view pictures of the coded video sequence refer to reference pictures that are right-view pictures. left_view_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the left-view pictures of the coded video sequence may or may not refer to reference pictures that are right-view pictures. Within a coded video sequence, the value of left_view_self_contained_flag in all stereo video information SEI messages shall be the same.

right_view_self_contained_flag equal to 1 indicates that no inter prediction operations within the decoding process for the right-view pictures of the coded video sequence refer to reference pictures that are left-view pictures. right_view_self_contained_flag equal to 0 indicates that some inter prediction operations within the decoding process for the right-view pictures of the coded video sequence may or may not refer to reference pictures that are left-view pictures. Within a coded video sequence, the value of right_view_self_contained_flag in all stereo video information SEI messages shall be the same.

D.2.23 Reserved SEI message semantics

This message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. Encoders conforming to this Recommendation | International Standard shall not send reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders conforming to this Recommendation | International Standard that encounter reserved SEI messages shall discard their content without effect on the decoding process, except as specified in future Recommendations | International Standards specified by ITU-T | ISO/IEC.

reserved_sei_message_payload_byte is a byte reserved for future use by ITU-T | ISO/IEC.

EAnnex E
Video usability information

(This annex forms an integral part of this Recommendation | International Standard)

This Annex specifies syntax and semantics of the VUI parameters of the sequence parameter sets.

VUI parameters are not required for constructing the luma or chroma samples by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Recommendation | International Standard (see Annex C for the specification of conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex E, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified by this Recommendation | International Standard. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in subclauses 7.3.2.1 and 7.4.2.1 and this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.1 VUI syntax

E.1.1 VUI parameters syntax

vui_parameters( ) {	C	Descriptor
aspect_ratio_info_present_flag	0	u(1)
if(aspect_ratio_info_present_flag) {		
aspect_ratio_idc	0	u(8)
if(aspect_ratio_idc = = Extended_SAR) {		
sar_width	0	u(16)
sar_height	0	u(16)
}		
}		
overscan_info_present_flag	0	u(1)
if(overscan_info_present_flag)		
overscan_appropriate_flag	0	u(1)
video_signal_type_present_flag	0	u(1)
if(video_signal_type_present_flag) {		
video_format	0	u(3)
video_full_range_flag	0	u(1)
colour_description_present_flag	0	u(1)
if(colour_description_present_flag) {		
colour_primaries	0	u(8)
transfer_characteristics	0	u(8)
matrix_coefficients	0	u(8)
}		
}		
chroma_loc_info_present_flag	0	u(1)
if(chroma_loc_info_present_flag) {		
chroma_sample_loc_type_top_field	0	ue(v)
chroma_sample_loc_type_bottom_field	0	ue(v)
}		
timing_info_present_flag	0	u(1)
if(timing_info_present_flag) {		
num_units_in_tick	0	u(32)
time_scale	0	u(32)
fixed_frame_rate_flag	0	u(1)
}		
nal_hrd_parameters_present_flag	0	u(1)
if(nal_hrd_parameters_present_flag)		
hrd_parameters( )		
vcl_hrd_parameters_present_flag	0	u(1)
if(vcl_hrd_parameters_present_flag)		
hrd_parameters( )		
if(nal_hrd_parameters_present_flag || vcl_hrd_parameters_present_flag)		
low_delay_hrd_flag	0	u(1)
pic_struct_present_flag	0	u(1)

bitstream_restriction_flag	0	u(1)
if(bitstream_restriction_flag) {		
motion_vectors_over_pic_boundaries_flag	0	u(1)
max_bytes_per_pic_denom	0	ue(v)
max_bits_per_mb_denom	0	ue(v)
log2_max_mv_length_horizontal	0	ue(v)
log2_max_mv_length_vertical	0	ue(v)
num_reorder_frames	0	ue(v)
max_dec_frame_buffering	0	ue(v)
}		
}		

E.1.2 HRD parameters syntax

hrd_parameters( ) {	C	Descriptor
cpb_cnt_minus1	0	ue(v)
bit_rate_scale	0	u(4)
cpb_size_scale	0	u(4)
for(SchedSelIdx = 0; SchedSelIdx <= cpb_cnt_minus1; SchedSelIdx++) {		
bit_rate_value_minus1[SchedSelIdx **]**	0	ue(v)
cpb_size_value_minus1[SchedSelIdx **]**	0	ue(v)
cbr_flag[SchedSelIdx **]**	0	u(1)
}		
initial_cpb_removal_delay_length_minus1	0	u(5)
cpb_removal_delay_length_minus1	0	u(5)
dpb_output_delay_length_minus1	0	u(5)
time_offset_length	0	u(5)
}		

E.2 VUI semantics

E.2.1 VUI parameters semantics

aspect_ratio_info_present_flag equal to 1 specifies that aspect_ratio_idc is present. aspect_ratio_info_present_flag equal to 0 specifies that aspect_ratio_idc is not present.

aspect_ratio_idc specifies the value of the sample aspect ratio of the luma samples. Table E-1 shows the meaning of the code. When aspect_ratio_idc indicates Extended_SAR, the sample aspect ratio is represented by sar_width and sar_height. When the aspect_ratio_idc syntax element is not present, aspect_ratio_idc value shall be inferred to be equal to 0.

Table E-1 – Meaning of sample aspect ratio indicator

aspect_ratio_idc	Sample aspect ratio	(informative) Examples of use
0	Unspecified	
1	1:1 (“square”)	1280x720 16:9 frame without horizontal overscan 1920x1080 16:9 frame without horizontal overscan (cropped from 1920x1088) 640x480 4:3 frame without horizontal overscan
2	12:11	720x576 4:3 frame with horizontal overscan 352x288 4:3 frame without horizontal overscan
3	10:11	720x480 4:3 frame with horizontal overscan 352x240 4:3 frame without horizontal overscan
4	16:11	720x576 16:9 frame with horizontal overscan 528x576 4:3 frame without horizontal overscan
5	40:33	720x480 16:9 frame with horizontal overscan 528x480 4:3 frame without horizontal overscan
6	24:11	352x576 4:3 frame without horizontal overscan 480x576 16:9 frame with horizontal overscan
7	20:11	352x480 4:3 frame without horizontal overscan 480x480 16:9 frame with horizontal overscan
8	32:11	352x576 16:9 frame without horizontal overscan
9	80:33	352x480 16:9 frame without horizontal overscan
10	18:11	480x576 4:3 frame with horizontal overscan
11	15:11	480x480 4:3 frame with horizontal overscan
12	64:33	528x576 16:9 frame without horizontal overscan
13	160:99	528x480 16:9 frame without horizontal overscan
14	4:3	1440x1080 16:9 frame without horizontal overscan
15	3:2	1280x1080 16:9 frame without horizontal overscan
16	2:1	960x1080 16:9 frame without horizontal overscan
17..254	Reserved	
255	Extended_SAR	

sar_width indicates the horizontal size of the sample aspect ratio (in arbitrary units).

sar_height indicates the vertical size of the sample aspect ratio (in the same arbitrary units as sar_width).

sar_width and sar_height shall be relatively prime or equal to 0. When aspect_ratio_idc is equal to 0 or sar_width is equal to 0 or sar_height is equal to 0, the sample aspect ratio shall be considered unspecified by this Recommendation | International Standard.

overscan_info_present_flag equal to 1 specifies that the overscan_appropriate_flag is present. When overscan_info_present_flag is equal to 0 or is not present, the preferred display method for the video signal is unspecified.

overscan_appropriate_flag equal to 1 indicates that the cropped decoded pictures output are suitable for display using overscan. overscan_appropriate_flag equal to 0 indicates that the cropped decoded pictures output contain visually important information in the entire region out to the edges of the cropping rectangle of the picture, such that the cropped decoded pictures output should not be displayed using overscan. Instead, they should be displayed using either an exact match between the display area and the cropping rectangle, or using underscan.

NOTE 1 – For example, overscan_appropriate_flag equal to 1 might be used for entertainment television programming, or for a live view of people in a videoconference, and overscan_appropriate_flag equal to 0 might be used for computer screen capture or security camera content.

video_signal_type_present_flag equal to 1 specifies that video_format, video_full_range_flag and colour_description_present_flag are present. video_signal_type_present_flag equal to 0, specify that video_format, video_full_range_flag and colour_description_present_flag are not present.

video_format indicates the representation of the pictures as specified in Table E-2, before being coded in accordance with this Recommendation | International Standard. When the video_format syntax element is not present, video_format value shall be inferred to be equal to 5.

Table E-2 – Meaning of video_format

video_format	Meaning
0	Component
1	PAL
2	NTSC
3	SECAM
4	MAC
5	Unspecified video format
6	Reserved
7	Reserved

video_full_range_flag indicates the black level and range of the luma and chroma signals as derived from E'_Y, E'_{PB}, and E'_{PR} or E'_R, E'_G, and E'_B analogue component signals.

When the video_full_range_flag syntax element is not present, the value of video_full_range_flag shall be inferred to be equal to 0.

colour_description_present_flag equal to 1 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are present. colour_description_present_flag equal to 0 specifies that colour_primaries, transfer_characteristics and matrix_coefficients are not present.

colour_primaries indicates the chromaticity coordinates of the source primaries as specified in Table E-3 in terms of the CIE 1931 definition of x and y as specified by ISO/CIE 10527.

When the colour_primaries syntax element is not present, the value of colour_primaries shall be inferred to be equal to 2 (the chromaticity is unspecified or is determined by the application).

Table E-3 – Colour primaries

Value	Primaries			Informative Remark
0	Reserved			For future use by ITU-T / ISO/IEC
1	primary	x	y	ITU-R Recommendation BT.709-5
	green	0.300	0.600	
	blue	0.150	0.060	
	red	0.640	0.330	
	white D65	0.3127	0.3290	
2	Unspecified			Image characteristics are unknown or are determined by the application.
3	Reserved			
4	primary	x	y	ITU-R Recommendation BT.470-6 System M
	green	0.21	0.71	
	blue	0.14	0.08	
	red	0.67	0.33	
	white C	0.310	0.316	
5	primary	x	y	ITU-R Recommendation BT.470-6 System B, G
	green	0.29	0.60	
	blue	0.15	0.06	
	red	0.64	0.33	
	white D65	0.3127	0.3290	
6	primary	x	y	Society of Motion Picture and Television Engineers 170M (1999)
	green	0.310	0.595	
	blue	0.155	0.070	
	red	0.630	0.340	
	white D65	0.3127	0.3290	
7	primary	x	y	Society of Motion Picture and Television Engineers 240M (1999)
	green	0.310	0.595	
	blue	0.155	0.070	
	red	0.630	0.340	
	white D65	0.3127	0.3290	
8	primary	x	y	Generic film (colour filters using Illuminant C)
	green	0.243	0.692 (Wratten 58)	
	blue	0.145	0.049 (Wratten 47)	
	red	0.681	0.319 (Wratten 25)	
	white C	0.310	0.316	
9-255	Reserved			For future use by ITU-T / ISO/IEC

transfer_characteristics indicates the opto-electronic transfer characteristic of the source picture as specified in Table E-4 as a function of a linear optical intensity input L_c with an analogue range of 0 to 1.

When the transfer_characteristics syntax element is not present, the value of transfer_characteristics shall be inferred to be equal to 2 (the transfer characteristics are unspecified or are determined by the application).

Table E-4 – Transfer characteristics

Value	Transfer Characteristic		Informative Remark
0	Reserved		For future use by ITU-T / ISO/IEC
1	$V = 1.099\ L_c^{0.45} - 0.099$ $V = 4.500\ L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	ITU-R Recommendation BT.709-5
2	Unspecified		Image characteristics are unknown or are determined by the application.
3	Reserved		For future use by ITU-T / ISO/IEC
4	Assumed display gamma 2.2		ITU-R Recommendation BT.470-6 System M
5	Assumed display gamma 2.8		ITU-R Recommendation BT.470-6 System B, G
6	$V = 1.099\ L_c^{0.45} - 0.099$ $V = 4.500\ L_c$	for $1 >= L_c >= 0.018$ for $0.018 > L_c >= 0$	Society of Motion Picture and Television Engineers 170M (1999)
7	$V = 1.1115\ L_c^{0.45} - 0.1115$ $V = 4.0\ L_c$	for $1 >= L_c >= 0.0228$ for $0.0228 > L_c >= 0$	Society of Motion Picture and Television Engineers 240M (1999)
8	$V = L_C$	for $1 > L_c >= 0$	Linear transfer characteristics
9	$V = 1.0 - \mathrm{Log10}(L_c) \div 2$ $V = 0.0$	for $1 >= L_c >= 0.01$ for $0.01 > L_c >= 0$	Logarithmic transfer characteristic (100:1 range)
10	$V = 1.0 - \mathrm{Log10}(L_c) \div 2.5$ $V = 0.0$	for $1 >= L_c >= 0.0031622777$ for $0.0031622777 > L_c >= 0$	Logarithmic transfer characteristic (316.22777:1 range)
11..255	Reserved		For future use by ITU-T / ISO/IEC

matrix_coefficients describes the matrix coefficients used in deriving luma and chroma signals from the green, blue, and red primaries, as specified in Table E-5.

matrix_coefficients shall not be equal to 0 unless both of the following conditions are true

– $\mathrm{BitDepth}_C$ is equal to $\mathrm{BitDepth}_Y$

– chroma_format_idc is equal to 3 (4:4:4)

The specification of the use of matrix_coefficients equal to 0 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

matrix_coefficients shall not be equal to 8 unless one or both of the following conditions are true

– $\mathrm{BitDepth}_C$ is equal to $\mathrm{BitDepth}_Y$

– $\mathrm{BitDepth}_C$ is equal to $\mathrm{BitDepth}_Y + 1$ and chroma_format_idc is equal to 3 (4:4:4)

The specification of the use of matrix_coefficients equal to 8 under all other conditions is reserved for future use by ITU-T | ISO/IEC.

When the matrix_coefficients syntax element is not present, the value of matrix_coefficients shall be inferred to be equal to 2.

The interpretation of matrix_coefficients is defined as follows.

– E'_R, E'_G, and E'_B are analogue with values in the range of 0 to 1.

– White is specified as having E'_R equal to 1, E'_G equal to 1, and E'_B equal to 1.

– Black is specified as having E'_R equal to 0, E'_G equal to 0, and E'_B equal to 0.

– If video_full_range_flag is equal to 0, the following equations apply.

  – If matrix_coefficients is equal to 1, 4, 5, 6, or 7, the following equations apply.

$$Y = Round((1 << (BitDepth_Y – 8)) * (219 * E'_Y + 16)) \quad (E\text{-}1)$$

$$Cb = Round((1 << (BitDepth_C – 8)) * (224 * E'_{PB} + 128)) \quad (E\text{-}2)$$

$$Cr = Round((1 << (BitDepth_C – 8)) * (224 * E'_{PR} + 128)) \quad (E\text{-}3)$$

– Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply.

$$R = (1 << (BitDepth_Y – 8)) * (219 * E'_R + 16) \quad (E\text{-}4)$$

$$G = (1 << (BitDepth_Y – 8)) * (219 * E'_G + 16) \quad (E\text{-}5)$$

$$B = (1 << (BitDepth_Y – 8)) * (219 * E'_B + 16) \quad (E\text{-}6)$$

– Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.

– Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, or 8), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.

– Otherwise (video_full_range_flag is equal to 1), the following equations apply.

– If matrix_coefficients is equal to 1, 4, 5, 6, or 7, the following equations apply.

$$Y = Round(((1 << BitDepth_Y) – 1) * E'_Y) \quad (E\text{-}7)$$

$$Cb = Round(((1 << BitDepth_C) – 1) * E'_{PB} + (1 << (BitDepth_C – 1)) \quad (E\text{-}8)$$

$$Cr = Round(((1 << BitDepth_C) – 1) * E'_{PR} + (1 << (BitDepth_C – 1)) \quad (E\text{-}9)$$

– Otherwise, if matrix_coefficients is equal to 0 or 8, the following equations apply.

$$R = ((1 << BitDepth_Y) – 1) * E'_R \quad (E\text{-}10)$$

$$G = ((1 << BitDepth_Y) – 1) * E'_G \quad (E\text{-}11)$$

$$B = ((1 << BitDepth_Y) – 1) * E'_B \quad (E\text{-}12)$$

– Otherwise, if matrix_coefficients is equal to 2, the interpretation of the matrix_coefficients syntax element is unknown or is determined by the application.

– Otherwise (matrix_coefficients is not equal to 0, 1, 2, 4, 5, 6, 7, or 8), the interpretation of the matrix_coefficients syntax element is reserved for future definition by ITU-T | ISO/IEC.

– If matrix_coefficients is not equal to 0 or 8, the following equations apply.

$$E'_Y = K_R * E'_R + (1 – K_R – K_B) * E'_G + K_B * E'_B \quad (E\text{-}13)$$

$$E'_{PB} = 0.5 * (E'_B – E'_Y) \div (1 – K_B) \quad (E\text{-}14)$$

$$E'_{PR} = 0.5 * (E'_R – E'_Y) \div (1 – K_R) \quad (E\text{-}15)$$

NOTE 2 – Then E'_Y is analogue with values in the range of 0 to 1, E'_{PB} and E'_{PR} are analogue with values in the range of -0.5 to 0.5, and white is equivalently given by $E'_Y = 1$, $E'_{PB} = 0$, $E'_{PR} = 0$.

– Otherwise, if matrix_coefficients is equal to 0, the following equations apply.

$$Y = \text{Round}(G) \quad \text{(E-16)}$$

$$Cb = \text{Round}(B) \quad \text{(E-17)}$$

$$Cr = \text{Round}(R) \quad \text{(E-18)}$$

– Otherwise (matrix_coefficients is equal to 8), the following applies.

  – If BitDepth_C is equal to BitDepth_Y, the following equations apply.

$$Y = \text{Round}(0.5 * G + 0.25 * (R + B)) \quad \text{(E-19)}$$

$$Cb = \text{Round}(0.5 * G – 0.25 (R + B)) \quad \text{(E-20)}$$

$$Cr = \text{Round}(0.5 * (R – B)) \quad \text{(E-21)}$$

NOTE 3 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-20 and E-21 may be referred to as Cg and Co, respectively. The inverse conversion for the above four equations should be computed as.

$$t = Y – Cb \quad \text{(E-22)}$$

$$G = Y + Cb \quad \text{(E-23)}$$

$$B = t – Cr \quad \text{(E-24)}$$

$$R = t + Cr \quad \text{(E-25)}$$

  – Otherwise (BitDepth_C is not equal to BitDepth_Y), the following equations apply.

$$Cr = \text{Round}(R) – \text{Round}(B) \quad \text{(E-26)}$$

$$t = \text{Round}(B) + (Cr >> 1) \quad \text{(E-27)}$$

$$Cb = \text{Round}(G) – t \quad \text{(E-28)}$$

$$Y = t + (Cb >> 1) \quad \text{(E-29)}$$

NOTE 4 – For purposes of the YCgCo nomenclature used in Table E-5, Cb and Cr of Equations E-28 and E-26 may be referred to as Cg and Co, respectively. The inverse conversion for the above four equations should be computed as.

$$t = Y – (Cb >> 1) \quad \text{(E-30)}$$

$$G = t + Cb \quad \text{(E-31)}$$

$$B = t – (Cr >> 1) \quad \text{(E-32)}$$

$$R = B + Cr \tag{E-33}$$

Table E-5 – Matrix coefficients

Value	Matrix	Informative remark
0	GBR	Typically referred to as RGB; see Equations E-16 to E-18
1	$K_R = 0.2126$; $K_B = 0.0722$	ITU-R Recommendation BT.709-5 and Society of Motion Picture and Television Engineers RP177 (1993)
2	Unspecified	Image characteristics are unknown or are determined by the application.
3	Reserved	For future use by ITU-T / ISO/IEC
4	$K_R = 0.30$; $K_B = 0.11$	United States Federal Communications Commission Title 47 Code of Federal Regulations (2003) 73.682 (a) (20)
5	$K_R = 0.299$; $K_B = 0.114$	ITU-R Recommendation BT.470-6 System B, G
6	$K_R = 0.299$; $K_B = 0.114$	Society of Motion Picture and Television Engineers 170M (1999)
7	$K_R = 0.212$; $K_B = 0.087$	Society of Motion Picture and Television Engineers 240M (1999)
8	YCgCo	See Equations E-19 to E-33
9-255	Reserved	For future use by ITU-T / ISO/IEC

chroma_loc_info_present_flag equal to 1 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are present. chroma_loc_info_present_flag equal to 0 specifies that chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present.

chroma_sample_loc_type_top_field and **chroma_sample_loc_type_bottom_field** specify the location of chroma samples for the top field and the bottom field as shown in Figure E-1. The value of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be in the range of 0 to 5, inclusive. When the chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field are not present, the values of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field shall be inferred to be equal to 0.

NOTE 5 – When coding progressive source material, chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field should have the same value.

Figure E-1 – Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field

timing_info_present_flag equal to 1 specifies that num_units_in_tick, time_scale and fixed_frame_rate_flag are present in the bitstream. timing_info_present_flag equal to 0 specifies that num_units_in_tick, time_scale and fixed_frame_rate_flag are not present in the bitstream.

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num_units_in_tick shall be greater than 0. A clock tick is the minimum interval of time that can be represented in the coded data. For example, when the clock frequency of a video signal is 60 000 ÷ 1001 Hz, time_scale may be equal to 60 000 and num_units_in_tick may be equal to 1001. See Equation C-1.

time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time_scale of 27 000 000. time_scale shall be greater than 0.

fixed_frame_rate_flag equal to 1 indicates that the temporal distance between the HRD output times of any two consecutive pictures in output order is constrained as follows. fixed_frame_rate_flag equal to 0 indicates that no such

constraints apply to the temporal distance between the HRD output times of any two consecutive pictures in output order.

For each picture n where n indicates the n-th picture (in output order) that is output and picture n is not the last picture in the bitstream (in output order) that is output, the value of $\Delta t_{fi,dpb}(n)$ is specified by

$$\Delta t_{fi,dpb}(n) = \Delta t_{o,dpb}(n) \div \text{DeltaTfiDivisor} \quad \text{(E-34)}$$

where $\Delta t_{o,dpb}(n)$ is specified in Equation C-13 and DeltaTfiDivisor is specified by Table E-6 based on the value of pic_struct_present_flag, field_pic_flag, and pic_struct for the coded video sequence containing picture n. Entries marked "-" in Table E-6 indicate a lack of dependence of DeltaTfiDivisor on the corresponding syntax element.

When fixed_frame_rate_flag is equal to 1 for a coded video sequence containing picture n, the value computed for $\Delta t_{fi,dpb}(n)$ shall be equal to t_c as specified in Equation C-1 (using the value of t_c for the coded video sequence containing picture n) when either or both of the following conditions are true for the following picture n_n that is specified for use in Equation C-13.

– picture n_n is in the same coded video sequence as picture n.

– picture n_n is in a different coded video sequence and fixed_frame_rate_flag is equal to 1 in the coded video sequence containing picture n_n and the value of num_units_in_tick ÷ time_scale is the same for both coded video sequences.

Table E-6 – Divisor for computation of $\Delta t_{fi,dpb}(n)$

pic_struct_present_flag	field_pic_flag	pic_struct	DeltaTfiDivisor
0	1	-	1
1	-	1	1
1	-	2	1
0	0	-	2
1	-	0	2
1	-	3	2
1	-	4	2
1	-	5	3
1	-	6	3
1	-	7	4
1	-	8	6

nal_hrd_parameters_present_flag equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance) are present. nal_hrd_parameters_present_flag equal to 0 specifies that NAL HRD parameters are not present.

NOTE 6 – When nal_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters, including the NAL sequence HRD parameter information and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

When nal_hrd_parameters_present_flag is equal to 1, NAL HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag.

The variable NalHrdBpPresentFlag is derived as follows.

– If any of the following is true, the value of NalHrdBpPresentFlag shall be set equal to 1.

  – nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1

  – the need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.

– Otherwise, the value of NalHrdBpPresentFlag shall be set equal to 0.

vcl_hrd_parameters_present_flag equal to 1 specifies that VCL HRD parameters (pertaining to all bitstream conformance) are present. vcl_hrd_parameters_present_flag equal to 0 specifies that VCL HRD parameters are not present.

NOTE 7 – When vcl_hrd_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the VCL HRD parameters and all buffering period and picture timing SEI messages, by some means not specified in this Recommendation | International Standard.

When vcl_hrd_parameters_present_flag is equal to 1, VCL HRD parameters (subclauses E.1.2 and E.2.2) immediately follow the flag.

The variable VclHrdBpPresentFlag is derived as follows.

– If any of the following is true, the value of VclHrdBpPresentFlag shall be set equal to 1.
  – vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1
  – the need for presence of buffering periods for VCL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of VclHrdBpPresentFlag shall be set equal to 0.

The variable CpbDpbDelaysPresentFlag is derived as follows.

– If any of the following is true, the value of CpbDpbDelaysPresentFlag shall be set equal to 1.
  – nal_hrd_parameters_present_flag is present in the bitstream and is equal to 1
  – vcl_hrd_parameters_present_flag is present in the bitstream and is equal to 1
  – the need for presence of CPB and DPB output delays to be present in the bitstream in picture timing SEI messages is determined by the application, by some means not specified in this Recommendation | International Standard.
– Otherwise, the value of CpbDpbDelaysPresentFlag shall be set equal to 0.

low_delay_hrd_flag specifies the HRD operational mode as specified in Annex C. When fixed_frame_rate_flag is equal to 1, low_delay_hrd_flag shall be equal to 0.

NOTE 8 – When low_delay_hrd_flag is equal to 1, "big pictures" that violate the nominal CPB removal times due to the number of bits used by an access unit are permitted. It is expected, but not required, that such "big pictures" occur only occasionally.

pic_struct_present_flag equal to 1 specifies that picture timing SEI messages (subclause D.2.2) are present that include the pic_struct syntax element. pic_struct_present_flag equal to 0 specifies that the pic_struct syntax element is not present in picture timing SEI messages. When pic_struct_present_flag is not present, its value shall be inferred to be equal to 0.

bitstream_restriction_flag equal to 1, specifies that the following coded video sequence bitstream restriction parameters are present. bitstream_restriction_flag equal to 0, specifies that the following coded video sequence bitstream restriction parameters are not present.

motion_vectors_over_pic_boundaries_flag equal to 0 indicates that no sample outside the picture boundaries and no sample at a fractional sample position whose value is derived using one or more samples outside the picture boundaries is used to inter predict any sample. motion_vectors_over_pic_boundaries_flag equal to 1 indicates that one or more samples outside picture boundaries may be used in inter prediction. When the motion_vectors_over_pic_boundaries_flag syntax element is not present, motion_vectors_over_pic_boundaries_flag value shall be inferred to be equal to 1.

max_bytes_per_pic_denom indicates a number of bytes not exceeded by the sum of the sizes of the VCL NAL units associated with any coded picture in the coded video sequence.

The number of bytes that represent a picture in the NAL unit stream is specified for this purpose as the total number of bytes of VCL NAL unit data (i.e., the total of the NumBytesInNALunit variables for the VCL NAL units) for the picture. The value of max_bytes_per_pic_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bytes_per_pic_denom the following applies.

– If max_bytes_per_pic_denom is equal to 0, no limits are indicated.

– Otherwise (max_bytes_per_pic_denom is not equal to 0), no coded picture shall be represented in the coded video sequence by more than the following number of bytes.

$$(PicSizeInMbs * RawMbBits) \div (8 * max_bytes_per_pic_denom) \quad \text{(E-35)}$$

When the max_bytes_per_pic_denom syntax element is not present, the value of max_bytes_per_pic_denom shall be inferred to be equal to 2.

max_bits_per_mb_denom indicates the maximum number of coded bits of macroblock_layer( ) data for any macroblock in any picture of the coded video sequence. The value of max_bits_per_mb_denom shall be in the range of 0 to 16, inclusive.

Depending on max_bits_per_mb_denom the following applies.

– If max_bits_per_mb_denom is equal to 0, no limit is specified.

– Otherwise (max_bits_per_mb_denom is not equal to 0), no coded macroblock_layer( ) shall be represented in the bitstream by more than the following number of bits.

$$(128 + RawMbBits) \div max_bits_per_mb_denom \quad \text{(E-36)}$$

Depending on entropy_coding_mode_flag, the bits of macroblock_layer( ) data are counted as follows.

– If entropy_coding_mode_flag is equal to 0, the number of bits of macroblock_layer( ) data is given by the number of bits in the macroblock_layer( ) syntax structure for a macroblock.

– Otherwise (entropy_coding_mode_flag is equal to 1), the number of bits of macroblock_layer( ) data for a macroblock is given by the number of times read_bits(1) is called in subclauses 9.3.3.2.2 and 9.3.3.2.3 when parsing the macroblock_layer( ) associated with the macroblock.

When the max_bits_per_mb_denom is not present, the value of max_bits_per_mb_denom shall be inferred to be equal to 1.

log2_max_mv_length_horizontal and **log2_max_mv_length_vertical** indicate the maximum absolute value of a decoded horizontal and vertical motion vector component, respectively, in ¼ luma sample units, for all pictures in the coded video sequence. A value of n asserts that no value of a motion vector component shall exceed the range from -2^n to 2^n-1, inclusive, in units of ¼ luma sample displacement. The value of log2_max_mv_length_horizontal shall be in the range of 0 to 16, inclusive. The value of log2_max_mv_length_vertical shall be in the range of 0 to 16, inclusive. When log2_max_mv_length_horizontal is not present, the values of log2_max_mv_length_horizontal and log2_max_mv_length_vertical shall be inferred to be equal to 16.

NOTE 9 – The maximum absolute value of a decoded vertical or horizontal motion vector component is also constrained by profile and level limits as specified in Annex A.

num_reorder_frames indicates the maximum number of frames, complementary field pairs, or non-paired fields that precede any frame, complementary field pair, or non-paired field in the coded video sequence in decoding order and follow it in output order. The value of num_reorder_frames shall be in the range of 0 to max_dec_frame_buffering, inclusive. When the num_reorder_frames syntax element is not present, the value of num_reorder_frames value shall be inferred to be equal to max_dec_frame_buffering.

max_dec_frame_buffering specifies the required size of the HRD decoded picture buffer (DPB) in units of frame buffers. The coded video sequence shall not require a decoded picture buffer with size of more than Max(1, max_dec_frame_buffering) frame buffers to enable the output of decoded pictures at the output times specified by dpb_output_delay of the picture timing SEI messages. The value of max_dec_frame_buffering shall be in the range of num_ref_frames to MaxDpbSize (as specified in subclause A.1.1 or A.3.1), inclusive. When the max_dec_frame_buffering syntax element is not present, the value of max_dec_frame_buffering shall be inferred to be equal to MaxDpbSize.

E.2.2 HRD parameters semantics

cpb_cnt_minus1 plus 1 specifies the number of alternative CPB specifications in the bitstream. The value of cpb_cnt_minus1 shall be in the range of 0 to 31, inclusive. When low_delay_hrd_flag is equal to 1, cpb_cnt_minus1 shall be equal to 0. When cpb_cnt_minus1 is not present, it shall be inferred to be equal to 0.

bit_rate_scale (together with bit_rate_value_minus1[SchedSelIdx]) specifies the maximum input bit rate of the SchedSelIdx-th CPB.

cpb_size_scale (together with cpb_size_value_minus1[SchedSelIdx]) specifies the CPB size of the SchedSelIdx-th CPB.

bit_rate_value_minus1[SchedSelIdx **]** (together with bit_rate_scale) specifies the maximum input bit rate for the SchedSelIdx-th CPB. bit_rate_value_minus1[SchedSelIdx] shall be in the range of 0 to 2^{32} - 2, inclusive. For any SchedSelIdx > 0, bit_rate_value_minus1[SchedSelIdx] shall be greater than bit_rate_value_minus1[SchedSelIdx - 1]. The bit rate in bits per second is given by

$$\text{BitRate[SchedSelIdx]} = (\text{ bit_rate_value_minus1[SchedSelIdx]} + 1) * 2^{(6 + \text{bit_rate_scale})} \qquad \text{(E-37)}$$

When the bit_rate_value_minus1[SchedSelIdx] syntax element is not present, the value of BitRate[SchedSelIdx] shall be inferred as follows.

– If profile_idc is equal to 66, 77, or 88, BitRate[SchedSelIdx] shall be inferred to be equal to 1000 * MaxBR for VCL HRD parameters and to be equal to 1200 * MaxBR for NAL HRD parameters, where MaxBR is specified in subclause A.1.1.

– Otherwise, BitRate[SchedSelIdx] shall be inferred to be equal to cpbBrVclFactor * MaxBR for VCL HRD parameters and to be equal to cpbBrNalFactor * MaxBR for NAL HRD parameters, where cpbBrVclFactor, cpbBrNalFactor, and MaxBR are specified in subclause A.1.2.

cpb_size_value_minus1[SchedSelIdx **]** is used together with cpb_size_scale to specify the SchedSelIdx-th CPB size. cpb_size_value_minus1[SchedSelIdx] shall be in the range of 0 to 2^{32} - 2, inclusive. For any SchedSelIdx greater than 0, cpb_size_value_minus1[SchedSelIdx] shall be less than or equal to cpb_size_value_minus1[SchedSelIdx -1].

The CPB size in bits is given by

$$\text{CpbSize[SchedSelIdx]} = (\text{ cpb_size_value_minus1[SchedSelIdx]} + 1) * 2^{(4 + \text{cpb_size_scale})} \qquad \text{(E-38)}$$

When the cpb_size_value_minus1[SchedSelIdx] syntax element is not present, the value of CpbSize[SchedSelIdx] shall be inferred as follows.

– If profile_idc is equal to 66, 77, or 88, CpbSize[SchedSelIdx] shall be inferred to be equal to 1000 * MaxCPB for VCL HRD parameters and to be equal to 1200 * MaxCPB for NAL HRD parameters, where MaxCPB is specified in subclause A.1.1.

– Otherwise, CpbSize[SchedSelIdx] shall be inferred to be equal to cpbBrVclFactor * MaxCPB for VCL HRD parameters and to be equal to cpbBrNalFactor * MaxCPB for NAL HRD parameters, where cpbBrVclFactor, cpbBrNalFactor, and MaxCPB are specified in subclause A.1.2.

cbr_flag[SchedSelIdx **]** equal to 0 specifies that to decode this bitstream by the HRD using the SchedSelIdx-th CPB specification, the hypothetical stream delivery scheduler (HSS) operates in an intermittent bit rate mode. cbr_flag[SchedSelIdx] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When the cbr_flag[SchedSelIdx] syntax element is not present, the value of cbr_flag shall be inferred to be equal to 0.

initial_cpb_removal_delay_length_minus1 specifies the length in bits of the initial_cpb_removal_delay[SchedSelIdx] and initial_cpb_removal_delay_offset[SchedSelIdx] syntax elements of the buffering period SEI message. The length of initial_cpb_removal_delay[SchedSelIdx] and of initial_cpb_removal_delay_offset[SchedSelIdx] is initial_cpb_removal_delay_length_minus1 + 1. When the initial_cpb_removal_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the initial_cpb_removal_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the initial_cpb_removal_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

cpb_removal_delay_length_minus1 specifies the length in bits of the cpb_removal_delay syntax element. The length of the cpb_removal_delay syntax element of the picture timing SEI message is cpb_removal_delay_length_minus1 + 1. When the cpb_removal_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the cpb_removal_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the cpb_removal_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

dpb_output_delay_length_minus1 specifies the length in bits of the dpb_output_delay syntax element. The length of the dpb_output_delay syntax element of the picture timing SEI message is dpb_output_delay_length_minus1 + 1. When the dpb_output_delay_length_minus1 syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the dpb_output_delay_length_minus1 parameters shall be equal in both hrd_parameters( ) syntax structures. When the dpb_output_delay_length_minus1 syntax element is not present, it shall be inferred to be equal to 23.

time_offset_length greater than 0 specifies the length in bits of the time_offset syntax element. time_offset_length equal to 0 specifies that the time_offset syntax element is not present. When the time_offset_length syntax element is present in more than one hrd_parameters( ) syntax structure within the VUI parameters syntax structure, the value of the time_offset_length parameters shall be equal in both hrd_parameters( ) syntax structures. When the time_offset_length syntax element is not present, it shall be inferred to be equal to 24.

GAnnex G
Scalable video coding

(This annex forms an integral part of this Recommendation | International Standard)

This Annex specifies SVC.

SVC is an extension of the technical specification in clauses 1 through 9 and Annexes B through E. A decoder conforming to a profile including specifications of this Annex processes bitstreams according to the specification in this Annex. Parts of the decoding process in this Annex are specified by invoking processes in clauses 1 through 9 and Annexes B through E.

[Ed. (JR/HS/MW/TW) : The current text is a draft, the following points still need to be clarified, and finished

(A) Double-check and improve decoding process description

(B) Interlace

- most of the interlace seems to be highly redundant at a first glance

- also it seems to be very complicated to read

- is it possible to specify some subprocesses for deriving special parameters

- We think these parts really needs an improvement

- complicated, long-term item

(C) Syntax

- check syntax diagrams

- is it possible to simplify the syntax for PR slices

(D) Semantics

- clean-up, checking

- need to add several additional constraints

- are the inference rules are ok - should they be specified there ??? (in case they are only needed for a special process, they could be specified in F.8.XX)

(E) Parsing

- check, clean-up

(F) Annexes A-E may need some additional updates

- check, clean-up

]

G.1 Scope

This Annex specifies an extension of ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 referred to as SVC.

G.2 Normative References

The specifications in clause 2 shall apply.

G.3 Definitions

For the purposes of this Annex, the definitions as specified in clause 3 shall apply. Additionally, the following definitions are specified. These definitions are either not present in clause 3 or extend definitions in clause 3. When a definition is extended, only the extension of the definition is given in the text below.

G.3.1. **scalable layer**: A set of NAL units within an access unit with the same value of dependency_id.

G.3.2. **layer base**: A set of NAL units within a scalable layer with a value of quality_level being equal to 0.

G.3.3. **layer enhancement**: A set of NAL units within a scalable layer with a value of quality_level being larger or equal to 1.

G.3.4. **inter-layer coding**: Coding of a block, macroblock, slice, or picture that uses inter-layer prediction.

G.3.5. **inter-layer prediction**: A prediction derived from syntax elements or decoded samples from coded pictures that are part of a different layer than the current layer.

G.3.6. **intra-layer coding**: Coding of a block, macroblock, slice, or picture that uses intra-layer prediction.

G.3.7. **intra-layer prediction**: A prediction derived from syntax elements or decoded samples from other coded pictures that are part of the same layer as the current layer.

G.3.8. **coded slice**: a slice that contains slice data of a coded picture.

G.3.9. **suffix NAL unit**: A NAL unit that immediately follows another NAL unit in decoding order and contains descriptive information of the preceding NAL unit, which is referred to as the associated NAL unit. A suffix NAL unit shall have nal_ref_idc equal to 20 or 21, shall have dependency_id and quality_level both equal to 0, and shall not containg a coded slice. A suffix NAL unit belongs to the same coded picture as the associated NAL unit.

G.4 Abbreviations

The specifications in clause 4 shall apply.

G.5 Conventions

The specifications in clause 5 shall apply.

G.6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships

The specifications in subclauses 6.1, 6.2, and 6.3 shall apply. Additionally, the following derivation processes for base data are specified.

G.6.1 Derivation process for base quality slices

This process can only be invoked when the value of the syntax element slice_type is equal to PR.

Input to this process is

– a macroblock address mbAddr

– a variable baseQualityLevel

Output of this process is a base quality slice baseQualitySlice.

The slice baseQualitySlice is represented by its transform coefficients lists LumaLevel[ ], LumaLevel8x8[ ], Intra16x16DCLevel, Intra16x16ACLevel[ ], ChromaDCLevel[ ], and ChromaACLevel[ ][ ] for each macroblock as well as by its syntax elements and its derived motion data.

The slice baseQualitySlice, for which all of the following conditions are true, shall be the output.

– The syntax element dependency_id is equal to baseQualityDependencyId with baseQualityDependencyId being the syntax element dependency_id of the base quality slice baseQualitySlice.

– The syntax element quality_level of the base quality slice baseQualitySlice is equal to baseQualityLevel.

NOTE – The syntax elements dependency_id and quality_level are either transmitted in the NAL unit header or inferred.

– The value of PicOrderCnt(baseQualitySlice) is equal to the value of PicOrderCnt(CurrPic).

– The slice baseQualitySlice covers the macroblock with macroblock address mbAddr.

G.6.2 Derivation process for base macroblock

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

– a macroblock address mbAddr,

– a luma location (xP, yP) relative to the upper-left luma sample in the macroblock mbAddr.

Outputs of this process are

– a macroblock address mbAddrBase identifying the base macroblock,

– the base luma location (xB, yB) corresponding to luma location (xP, yP).

The position of the upper-left luma sample of the macroblock mbAddr is derived by invoking the inverse macroblock scanning process as specified in subclause 6.4.1 with mbAddr as input and the output being assigned to (xM, yM).

The base luma location (xB, yB) and the base macroblock address mbAddrBase are derived as follows.

– If SpatialScalabilityType is less than or equal to 3, the following applies.

  – The base luma location (xB, yB) is derived as follows.

$$xB = ((xP + xM - ScaledBaseLeftOffset) * BasePicWidth + ScaledBaseWidth / 2) / ScaledBaseWidth \quad \text{(G-1)}$$

$$yB = ((yP + yM - ScaledBaseTopOffset) * BasePicHeight + ScaledBaseHeight / 2) / ScaledBaseHeight \quad \text{(G-2)}$$

  – The base macroblock address mbAddrBase is derived as follows.

    – If any of the following conditions is true, mbAddrBase is marked as not available.

      – xB is less than 0, or xB is greater than or equal to BasePicWidth.

      – yB is less than 0, or yB is greater than or equal to BasePicHeight.

    – Otherwise, the following applies.

$$mbAddrBase = (yB / 16) * BasePicWidthInMbs + (xB / 16) \quad \text{(G-3)}$$

– Otherwise (SpatialScalabilityType is greater than 3), the following applies.

  – Let (vblXB, vblYB) be a virtual base layer luma location derived as follows.

    – Let scaledHeight and scaledTopOffset be temporary variables that are derived as follows.

      – If macroblock mbAddr is a field macroblock, the following applies.

$$scaledHeight = ScaledBaseHeight / 2 \quad \text{(G-4)}$$

$$scaledTopOffset = ScaledBaseTopOffset / 2 \quad \text{(G-5)}$$

      – Otherwise (macroblock mbAddr is a frame macroblock), the following applies.

$$scaledHeight = ScaledBaseHeight \quad \text{(G-6)}$$

$$scaledTopOffset = ScaledBaseTopOffset \quad \text{(G-7)}$$

- Let y0 be a temporary variable that is derived as follows.
  - If MbaffFrameFlag is equal to 1 and macroblock mbAddr is a field macroblock, the following applies.

    y0 = (yP + yM) / 2 - scaledTopOffset (G-8)

  - Otherwise (MbaffFrameFlag is equal to 0 or macroblock mbAddr is a frame macroblock), the following applies.

    y0 = yP + yM – scaledTopOffset (G-9)

- Let baseHeight, baseOffset and baseFactor be temporary variables that are derived as follows.
  - Let baseProgFrameFlag be a flag derived as follows.
    - If any of the following conditions is true, baseProgFrameFlag is set equal to 1,
      - BaseFrameMbsOnlyFlag is equal to 1,
      - BaseMbaffFrameFlag is equal to 0 and BaseFieldPicFlag is equal to 0.
    - Otherwise, baseProgFrameFlag is set equal to 0.
  - Let frameFieldFlag be a flag, which is derived as follows.
    - If the macroblock mbAddr is a field macroblock and any of the following conditions is true, frameFieldFlag is set equal to 1.
      - BaseMbaffFrameFlag is equal to 1.
      - baseProgFrameFlag is equal to 1 and ScaledBaseHeight is less than (2 * BasePicHeight).
    - Otherwise, frameFieldFlag is set equal to 0.
  - Depending on frameFieldFlag the following applies.
    - If frameFieldFlag is equal to 1, the following applies.
      - The variables baseHeight and basefactor are set equal to

        baseHeight = BasePicHeight / 2 (G-10)
        baseFactor = 2 (G-11)

      - The variable baseOffset is derived as follows.
        - If any of the following conditions is true, baseOffset is set equal to 0.
          - MbaffFrameFlag is equal to 1 and (yM / 16) is equal to 0.
          - bottom_field_flag is equal to 1.
        - Otherwise, baseOffset is set equal to 16.
    - Otherwise (frameFieldFlag is equal to 0), the variables baseHeight, baseFactor, and baseOffset are to equal to

      baseHeight = BasePicHeight (G-12)
      baseFactor = 1 (G-13)
      baseOffset = 0 (G-14)

- (vblXB, vblYB) is derived as follows.

  vblXB = ((xP + xM – ScaledBaseLeftOffset) * BasePicWidth + ScaledBaseWidth / 2) / ScaledBaseWidth (G-15)
  vblYB = baseOffset + baseFactor * (y0 * baseHeight + scaledHeight / 2) / scaledHeight (G-16)

- Let vblMbAddrBase be a virtual base layer macroblock address derived as follows.
  - If any of the following conditions is true, vblMbAddrBase is marked as not available.
    - vblXB is less than 0, or vblXB is greater than or equal to BasePicWidth.
    - vblYB is less than 0, or vblYB is greater than or equal to BasePicHeight.
  - Otherwise, the following applies.
    - If MbaffFrameFlag is equal to 1, vblMbAddrBase is derived by

$$\text{vblMbAddrBase} = 2 * ((\text{vblYB} / 32) * \text{BasePicWidthInMbs} + (\text{vblXB} / 16)) + (\text{vblYB} \% 32) / 16 \qquad \text{(G-17)}$$

    - Otherwise (MbaffFrameFlag is equal to 0), vblMbAddrBase is derived by

$$\text{vblMbAddrBase} = (\text{vblYB} / 16) * \text{BasePicWidthInMbs} + (\text{vblXB} / 16) \qquad \text{(G-18)}$$

- The process for virtual base layer macroblock conversion as specified in subclause G.6.5 is invoked with mbAddr, vblMbAddrBase, (vblXB, vblYB), and mbTypeBase as the inputs and the outputs are assigned to mbAddrBase and (xB, yB).

G.6.3 Derivation process for base partitions

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

- a macroblock address mbAddr,
- a luma location (xP, yP) relative to the upper-left luma sample in the macroblock mbAddr,
- an array mbTypeBase specifying the macroblock types of the base macroblocks,
- an array subMbTypeBase specifying the sub-macroblock types of the base macroblocks.

Outputs of this process are

- a macroblock address mbAddrBase identifying the base macroblock,
- a macroblock partition index mbPartIdxBase identifying the base macroblock partition inside the base macroblock mbAddrBase,
- a sub-macroblock partition index subMbPartIdxBase identifying the base sub-macroblock partition inside the macroblock partition mbPartIdxBase of the base macroblock mbAddrBase.

The derivation process for base macroblocks as specified in subclause G.6.2 is invoked with mbAddr and (xP, yP) as the inputs and the outputs are assigned to mbAddrBase and (xB, yB).

The base macroblock partition index mbPartIdxBase and the base sub-macroblock partition index subMbPartIdxBase are derived as follows.

- If mbAddrBase is marked as not available, mbPartIdxBase and subMbPartIdxBase are marked as not available.
- Otherwise, if mbTypeBase[mbAddrBase] specifies an I macroblock, mbPartIdxBase and subMbPartIdxBase are set equal to 0.
- Otherwise, if mbTypeBase[mbAddrBase] is equal to B_Skip or B_Direct_16x16, the following applies.
  - mbPartIdxBase is set equal to 2 * ((yB % 16) / 8) + (xB % 16) / 8.
  - The base sub-macroblock partition index subMbPartIdxBase is derived as follows.
    - If direct_8x8_inference_flag is equal to 1, subMbPartIdxBase is set equal to 0.
    - Otherwise (direct_8x8_inference_flag is equal to 0), subMbPartIdxBase is set equal to 2 * ((yB % 8) / 4) + (xB % 8) / 4.
- Otherwise, the following applies.

- The macroblock partition index of the macroblock partition in the macroblock given by mbTypeBase[mbAddrBase] covering the location (xB % 16, yB % 16) is assigned to mbPartIdxBase.
- The base sub-macroblock partition index subMbPartIdxBase is derived as follows.
  - If NumMbPart(mbTypeBase[mbAddrBase]) is equal to 4, the following applies.
    - If subMbTypeBase[mbAddrBase][mbPartIdxBase] is equal to B_Direct_8x8, subMbPartIdxBase is set equal to 2 * ((yB % 8) / 4) + (xB % 8) / 4.
    - Otherwise, the sub-macroblock partition index of the sub-macroblock partition covering the location (xB % 16, yB % 16) is assigned to subMbPartIdxBase.
  - Otherwise (NumMbPart(mbTypeBase[mbAddrBase]) is not equal to 4), subMbPartIdxBase is set equal to 0.

G.6.4 Derivation process for macroblock type and sub-macroblock type in inter-layer prediction

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

- a macroblock address mbAddr,
- an array mbTypeBase specifying the macroblock types of the base macroblocks,
- an array subMbTypeBase specifying the sub-macroblock types of the base macroblocks,
- two arrays predFlagL0Base and predFlagL1Base containing arrays of prediction utilization flags for each base macroblock.

Outputs of this process are

- a macroblock type mbType,
- a list of sub-macroblock types subMbType with 4 elements (if applicable),
- an 4x4 array partInfo indicating the corresponding base layer macroblock address, macroblock partition index and sub-macroblock partition index of each 4x4 block of the macroblock mbAddr.

Let partInfo be a 4x4 array.

For each (x, y) with x, y = 0..3, the following applies.

- The derivation process for base partitions as specified in subclause G.6.3 is invoked with mbAddr, (4 * x + 1, 4 * y + 1), mbTypeBase, and subMbTypeBase as input and mbAddrBase, mbPartIdxBase, and subMbPartIdxBase as output.
- The element partInfo[x, y] is derived as follows.
  - If mbAddrBase is marked as not available, partInfo[x, y] is marked as not available.
  - Otherwise, the following applies.
    - If macroblock mbAddrBase is intra coded,

      partInfo[x, y] = –1 (G-19)

    - Otherwise,

      partInfo[x, y] = 16 * mbAddrBase + 4 * mbPartIdxBase + subMbPartIdxBase (G-20)

When SpatialScalabilityType is greater than 2, the following applies.

- For (x0, y0) with x0, y0 = 0..1, the following applies.
  - For (x, y) with x, y = 0..1, the following applies.
    - When partInfo[2 * x0 + x, 2 * y0 + y] is equal to –1, the following applies.

- If partInfo[2 * x0 + 1 – x, 2 * y0 + y] is not equal to –1,

  partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * x0 + 1 – x, 2 * y0 + y] (G-21)

- Otherwise, if partInfo[2 * x0 + x, 2 * y0 + 1 – y] is not equal to –1,

  partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * x0 + x, 2 * y0 + 1 – y] (G-22)

- Otherwise, if partInfo[2 * x0 + 1 – x, 2 * y0 + 1 –y] is not equal to –1,

  partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * x0 + 1 – x, 2 * y0 + 1 – y] (G-23)

- Otherwise, partInfo[2 * x0 + x, 2 * y0 + y] is not modified.

– For (x0, y0) with x0, y0 = 0..1, the following applies.
  - When partInfo[2 * x0, 2 * y0] is equal to –1, the following applies.
    - If partInfo[2 * (1 – x0), 2 * y0] is not equal to –1, the following applies for x, y = 0..1.

      partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * (1 – x0) + x, 2 * y0 + y] (G-24)

    - Otherwise, if partInfo[2 * x0, 2 * (1 – y0)] is not equal to –1, the following applies for x, y = 0..1.

      partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * x0 + x, 2 * (1 – y0) + y] (G-25)

    - Otherwise, if partInfo[2 * (1 – x0), 2 * (1 – y0)] is not equal to –1, the following applies for x, y = 0..1.

      partInfo[2 * x0 + x, 2 * y0 + y] = partInfo[2 * (1 – x0) + x, 2 * (1 – y0) + y] (G-26)

    - Otherwise, the array partInfo is not modified.

The variable mbType is derived as follows.

– If any of the variables partInfo[x, y] is marked as not available, mbType is marked as not available.

– Otherwise, if partInfo[x, y] is equal to –1 for all x, y = 0..3, mbType is set equal to I_BL.

– Otherwise, the following applies.
  - Let partSize, partPredMode0, and partPredMode1 be temporary variables that are derived as follows.
    - If partInfo[x, y], with x, y = 0..3, is equal to partInfo[0, 0], the following applies

      partSize = 16x16
      partPredMode0 = 2 * predFlagL1Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4]
      + predFlagL0Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4] (G-27)

    - Otherwise, if partInfo[x, y], with x = 0..3 and y = 0..1, is equal to partInfo[0, 0] and partInfo[x, y], with x = 0..3 and y = 2..3, is equal to partInfo[0, 2], the following applies.

      partSize = 16x8
      partPredMode0 = 2 * predFlagL1Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4]
      + predFlagL0Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4] (G-28)
      partPredMode1 = 2 * predFlagL1Base[partInfo[3, 3] / 16][(partInfo[3, 3] % 16) / 4]
      + predFlagL0Base[partInfo[3, 3] / 16][(partInfo[3, 3] % 16) / 4] (G-29)

- Otherwise, if partInfo[x, y], with x = 0..1 and y = 0..3, is equal to partInfo[0, 0] and partInfo[x, y], with x = 2..3 and y = 0..3, is equal to partInfo[2, 0], the following applies

  partSize = 8x16
  partPredMode0 = 2 * predFlagL1Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4]
  + predFlagL0Base[partInfo[0, 0] / 16][(partInfo[0, 0] % 16) / 4] (G-30)
  partPredMode1 = 2 * predFlagL1Base[partInfo[3, 3] / 16][(partInfo[3, 3] % 16) / 4]
  + predFlagL0Base[partInfo[3, 3] / 16][(partInfo[3, 3] % 16) / 4] (G-31)

- Otherwise, partSize is set equal to 8x8.

- Depending on slice_type, partSize, partPredMode0, and partPredMode1, mbType is derived as specified by Table G-1.
- When mbType is equal to P_8x8 or B_8x8, for mbPartIdx = 0..3, the following applies.
  - Let x0, y0, and partPredMode be temporary variables that are derived by

    x0 = 2 * (mbPartIdx % 2) (G-32)
    y0 = 2 * (mbPartIdx / 2) (G-33)

  - partPredMode is derived as follows.
    - Let useFlagL0 be a temporary variable derived as follows.
      - If for all (x, y) with x, y = 0..1, predFlagL0Base[partInfo[x0 + x, y0 + y] / 16][(partInfo[x0 + x, y0 + y] % 16) / 4] is equal to 0, useFlagL0 is set to 0.
      - Otherwise, useFlagL0 is set to 1.
    - Let useFlagL1 be a temporary variable derived as follows.
      - If for all (x, y) with x, y = 0..1, predFlagL1Base[partInfo[x0 + x, y0 + y] / 16][(partInfo[x0 + x, y0 + y] % 16) / 4] is equal to 0, useFlagL1 is set to 0.
      - Otherwise, useFlagL1 is set to 1.
    - partPredMode is derived as

      partPredMode = 2 * useFlagL1 + useFlagL0 (G-34)

  - Let subPartSize be a temporary variable that is derived as follows.
    - If partInfo[x0 + x, y0 + y], with x, y = 0..1, is equal to partInfo[x0, y0], subPartSize is set equal to 8x8.
    - Otherwise, if partInfo[x0, y0] is equal to partInfo[x0 + 1, y0] and partInfo[x0, y0 + 1] is equal to partInfo[x0 + 1, y0 + 1], subPartSize is set equal to 8x4.
    - Otherwise, if partInfo[x0, y0] is equal to partInfo[x0, y0 + 1] and partInfo[x0 + 1, y0] is equal to partInfo[x0 + 1, y0 + 1], subPartSize is set equal to 4x8.
    - Otherwise, subPartSize is set equal to 4x4.
  - Depending on slice_type, subPartSize, partPredMode, subMbType[mbPartIdx] is derived as specified by Table G-2.

Table G-1 – Derivation of mackroblock types.

slice_type	partSize	partPredMode0	partPredMode1	mbType	slice_type	partSize	partPredMode0	partPredMode1	mbType
B	16x16	< 2	na	B_L0_16x16	B	16x8	2	3	B_L1_Bi_16x8
B	16x16	2	na	B_L1_16x16	B	8x16	2	3	B_L1_Bi_8x16
B	16x16	3	na	B_Bi_16x16	B	16x8	3	< 2	B_Bi_L0_16x8
B	16x8	< 2	< 2	B_L0_L0_16x8	B	8x16	3	< 2	B_Bi_L0_8x16
B	8x16	< 2	< 2	B_L0_L0_8x16	B	16x8	3	2	B_Bi_L1_16x8
B	16x8	2	2	B_L1_L1_16x8	B	8x16	3	2	B_Bi_L1_8x16
B	8x16	2	2	B_L1_L1_8x16	B	16x8	3	3	B_Bi_Bi_16x8
B	16x8	< 2	2	B_L0_L1_16x8	B	8x16	3	3	B_Bi_Bi_8x16
B	8x16	< 2	2	B_L0_L1_8x16	B	8x8	na	na	B_8x8
B	16x8	2	< 2	B_L1_L0_16x8	P	16x16	na	na	P_L0_16x16
B	8x16	2	< 2	B_L1_L0_8x16	P	16x8	na	na	P_L0_L0_16x8
B	16x8	< 2	3	B_L0_Bi_16x8	P	8x16	na	na	P_L0_L0_8x16
B	8x16	< 2	3	B_L0_Bi_8x16	P	8x8	na	na	P_8x8

Table G-2 – Derivation of sub-macroblock types.

slice_type	subPartSize	partPredMode	subMbType[mbPartIdx]	slice_type	subPartSize	partPredMode	subMbType[mbPartIdx]
B	8x8	< 2	B_L0_8x8	B	4x8	3	B_Bi_4x8
B	8x8	2	B_L1_8x8	B	4x4	< 2	B_L0_4x4
B	8x8	3	B_Bi_8x8	B	4x4	2	B_L1_4x4
B	8x4	< 2	B_L0_8x4	B	4x4	3	B_Bi_4x4
B	8x4	2	B_L1_8x4	P	8x8	na	P_L0_8x8
B	8x4	3	B_Bi_8x4	P	8x4	na	P_L0_8x4
B	4x8	< 2	B_L0_4x8	P	4x8	na	P_L0_4x8
B	4x8	2	B_L1_4x8	P	4x4	na	P_L0_4x4

G.6.5 Virtual base layer macroblock conversion

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

- a macroblock address mbAddr,
- a virtual base macroblock address vblMbAddrBase,
- a virtual base luma position (vblXB, vblYB),
- an array mbTypeBase specifying the macroblock types of the base macroblocks,

Outputs of this process are

- a base macroblock address mbAddrBase,
- a base layer luma position (xB, yB),

Let baseProgFrameFlag be a flag derived as follows.

- If any of the following conditions is true, baseProgFrameFlag is set equal to 1,
  - BaseFrameMbsOnlyFlag is equal to 1,
  - BaseMbaffFrameFlag is equal to 0 and BaseFieldPicFlag is equal to 0.
- Otherwise, baseProgFrameFlag is set equal to 0.

The macroblock address mbAddrBase and the base layer luma location (xB, yB) are derived as follows.

- If vblMbAddrBase is marked as not available, mbAddrBase is marked as not available.
- Otherwise, if all the following conditions are true, the process for Field-to-Frame base macroblock conversion as specified in subclause G.6.5.1 is invoked with vblMbAddrBase, (vblXB, vblYB), and mbTypeBase as the inputs and the outputs are assigned to mbAddrBase, and (xB, yB).
  - BaseMbaffFrameFlag is equal to 1,
  - the base macroblock vblMbAddrBase is a field macroblock,
  - frame_mbs_only_flag is equal to 1 or the macroblock mbAddr is a frame macroblock.
- Otherwise, the following applies.
  - Let frameFieldFlag be a variable that is derived as follows.
    - If any of the following conditions is true, frameFieldFlag is set equal to 1.
      - BaseFrameMbsOnlyFlag is equal to 0, the macroblock vblMbAddrBase is a frame macroblock, and the macroblock mbAddr is a field macroblock.
      - baseProgFrameFlag os equal to 1, the macroblock mbAddr is a field macroblock, and FrameHeightInMbs is less than (2 * BasePicWidthInMbs).
    - Otherwise, frameFieldFlag is set equal to 0.
  - Depending on frameFieldFlag the following applies.
    - If frameFieldFlag is equal to 1, the following applies.
      - Let bottomMbFlag be a temporary variable defined as follows.
        - If one of the following conditions is true, bottomMbFlag is set equal to 1.
          - bottom_field_flag is equal to 1,
          - MbaffFrameFlag is equal to 1 and (mbAddr % 2) is equal to 1.
        - Otherwise, bottomMbFlag is set equal to 0.
      - The process for Frame-to-Field base macroblock conversion as specified in subclause G.6.5.2 is invoked with bottomMbFlag, vblMbAddrBase, (vblXB, vblYB), and mbTypeBase as the inputs and the outputs are assigned to mbAddrBase, and (xB, yB).

– Otherwise (frameFieldFlag is equal to 0), mbAddrBase is set to vblMbAddrBase and (xB, yB) is set equal to (vblXB, vblYB).

G.6.5.1 Field-to-Frame base macroblock conversion

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

– a virtual base macroblock address vblMbAddrBase,

– a virtual base layer luma position (vblXB, vblYB),

– an array mbTypeBase specifying the macroblock types of the base macroblocks,

Outputs of this process are

– a base macroblock address mbAddrBase,

– a base layer luma position (xB, yB),

Let mbAddrBaseTop and mbAddrBaseBot be temporary variables defined as follows.

mbAddrBaseTop = vblMbAddrBase – (vblMbAddrBase % 2) (G-35)
mbAddrBaseBot = vblMbAddrBase + 1 – (vblMbAddrBase % 2) (G-36)

Depending on the macroblocks mbAddrBaseTop and mbAddrBaseBot, the following applies.

– If mbTypeBase[mbAddrBaseTop] and mbTypeBase[mbAddrBaseBot] both specify an I macroblock, the following applies.

mbAddrBase = vblMbAddrBase (G-37)
xB = vblXB (G-38)
yB = vblYB (G-39)

– Otherwise (mbTypeBase[mbAddrBaseTop] or mbTypeBase[mbAddrBaseBot] does not specify an I macroblock), the following applies.

  – Let y be a temporary variable, which is set equal to (vblYB % 16) / 4.

  – The variable mbAddrBase is derived as follows.

    – If one of the following conditions is true, mbAddrBase is set equal to mbAddrBaseTop.

      – y is less than 2 and base_bottom_field_coincided_flag is equal to 0

      – y is greater than or equal to 2 and base_bottom_field_coincided_flag is equal to 1

    – Otherwise, mbAddrBase is set equal to mbAddrBaseBot.

  – (xB, yB) is derived as follows.

xB = vblXB (G-40)
yB = 16 * (mbAddrBase / BasePicWidthInMbs) + 4 * (y / 2 + 2 * (vblMbAddrBase % 2)) (G-41)

G.6.5.2 Frame-to-Field base macroblock conversion

This process can only be invoked when the value of the syntax element layer_base_flag is equal to 0.

Inputs to this process are

– a flag bottomMbFlag indicating if the macroblock must be converted into a bottom or a top field macroblock,

– a virtual base macroblock address vblMbAddrBase,

– a virtual base layer luma position (vblXB, vblYB),

– an array mbTypeBase specifying the macroblock types of the base macroblocks,

Outputs of this process are

– a base macroblock address mbAddrBase,

– a base layer luma position (xB, yB),

Let mbAddrBaseTop and mbAddrBaseBot be temporary variables defined as follows. [Ed. Note(MW): mbAddrBase is not available yet. vblMbAddrBase instead?]

mbAddrBaseTop = vblMbAddrBase – (mbAddrBase % 2) (G-42)
mbAddrBaseBot = vblMbAddrBase + 1 – (mbAddrBase % 2) (G-43)

Depending on the macroblocks mbAddrBaseTop and mbAddrBaseBot, the following applies.

– If mbTypeBase[mbAddrBaseTop] and mbTypeBase[mbAddrBaseBot] both specify an I macroblock, the following applies.

mbAddrBase = vblMbAddrBase (G-44)
xB = vblXB (G-45)
yB = vblYB (G-46)

– Otherwise (mbTypeBase[mbAddrBaseTop] or mbTypeBase[mbAddrBaseBot] does not specify an I macroblock), the following applies.

  – The variables mbAddrBaseTop and mbAddrBaseBot are modified as follows.

    – If mbTypeBase[mbAddrBaseTop] specifies an I macroblock, mbAddrBaseTop is set to mbAddrBaseBot,

    – Otherwise if mbTypeBase[mbAddrBaseBot] specifies an I macroblock, mbAddrBaseBot is set to mbAddrBaseTop,

    – Otherwise mbAddrBaseTop and mbAddrBaseBot are not modified.

  – Let y be a temporary variable, which is set equal to (vblYB % 16) / 4.

  – The variable mbAddrBase is derived as follows.

    – If y is lower than 2, mbAddrBase is set equal to mbAddrBaseTop,

    – Otherwise (y is greater than or equal to 2), mbAddrBase is set equal to mbAddrBaseBot.

  – (xB, yB) is derived as follows.

xB = vblXB (G-47)
yB = 16 * (mbAddrBase / BasePicWidthInMbs) + 4 * (2 * (y % 2) +
(bottomMbFlag + base_frame_and_bottom_field_coincided_flag) % 2) (G-48)

G.7 Syntax and semantics

This clause specifies syntax and semantics for NAL unit types that are specified for SVC.

G.7.1 Method of specifying syntax in tabular form

The specifications in subclause 7.1 shall apply.

G.7.2 Specification of syntax functions, categories, and descriptors

The specifications in subclauses 7.2 shall apply. Additionally, the following syntax functions and descriptor are specified.

[Ed. (TW) Format those syntax functions that are needed to parse the NAL unit payloads for SVC]

next_nalu_not_belongs_to_current_PRF( ) is specified as follows.

- If any of the following conditions are true, the return value of next_nalu_not_belongs_to_current_PRF( ) is TRUE.
  - the syntax element dependency_id of the current NAL unit is not equal to the syntax element dependency_id of the following NAL unit
  - the syntax element quality_level of the current NAL unit is not equal to the syntax element quality_level of the following NAL unit
  - the syntax element first_mb_in_slice of the current NAL unit is not equal to the syntax element first_mb_in_slice of the following NAL unit
  - the syntax element slice_type of the current NAL unit is not equal to the syntax element slice_type of the following NAL unit
  - the following NAL unit belongs to a new access unit
  - the syntax element last_fragment_flag is equal to 1
- Otherwise the return value of next_nalu_not_belongs_to_current_PRF( ) is FALSE

swap_buffers(cBuffer0, cBuffer1) is specified as follows.

- A temporary buffer pointer cBufferPtr is set equal to cBuffer0
- cBuffer0 is set equal to cBuffer1
- cBuffer1 is set equal to cBufferPtr

spatial_scalability_type( ) is specified as follows.

- Let the variable interlaceFlag be derived as follows.
  - If one of the following conditions is true, interlaceFlag is equal to 0
    - frame_mbs_only_flag is equal to 1
    - frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and field_pic_flag is equal to 0
  - Otherwise, interlaceFlag is equal to 1.
- Let the variable baseInterlaceFlag be derived as follows,
  - If one of the following conditions is true, baseInterlaceFlag is equal to 0
    - BaseFrameMbsOnlyFlag is equal to 1
    - BaseMbaffFrameFlag is equal to 0 and BaseFieldPicFlag is equal to 0
  - Otherwise, baseInterlaceFlag is equal to 1
- The return value of spatial_scalability_type( ) is derived as follows.
  - If all the following conditions are true, the return value of spatial_scalability_type ( ) is equal to 0.
    - extended_spatial_scalability is less than 2
    - ScaledBaseWidth is equal to BasePicWidth
    - ScaledBaseHeight is equal to BasePicHeight
    - ScaledBaseLeftOffset % 16 is equal to 0
    - ScaledBaseTopOffset % 16 is equal to 0
    - chroma_format_idc is equal to BaseChromaFormatIdc
    - chroma_phase_x_plus1 is equal to base_chroma_phase_x_plus1
    - chroma_phase_y_plus1 is equal to base_chroma_phase_y_plus1
    - interlaceFlag is equal to 0

    - baseInterlaceFlag is equal to 0
  - Otherwise, if all the following condition are true, the return value of spatial_scalability_type ( ) is equal to 1.
    - extended_spatial_scalability is equal to 0
    - ScaledBaseWidth is equal to 2 * BasePicWidth
    - ScaledBaseHeight is equal to 2 * BasePicHeight
    - chroma_format_idc is equal to BaseChromaFormatIdc
    - interlaceFlag is equal to 0
    - baseInterlaceFlag is equal to 0
  - Otherwise, if all the following condition are true, the return value of spatial_scalability_type ( ) is equal to 2.
    - extended_spatial_scalability is less than 2
    - ScaledBaseWidth is equal to 2 * BasePicWidth
    - ScaledBaseHeight is equal to 2 * BasePicHeight
    - ScaledBaseLeftOffset % 16 is equal to 0
    - ScaledBaseTopOffset % 16 is equal to 0
    - chroma_format_idc is equal to BaseChromaFormatIdc
    - interlaceFlag is equal to 0
    - baseInterlaceFlag is equal to 0
  - Otherwise, if all the following condition are true, the return value of spatial_scalability_type ( ) is equal to 3.
    - interlaceFlag is equal to 0
    - baseInterlaceFlag is equal to 0
  - Otherwise, if baseInterlaceFlag is equal to 0, the return value of spatial_scalability_type ( ) is equal to 4.
  - Otherwise, if interlaceFlag is equal to 0, the return value of spatial_scalability_type ( ) is equal to 5.
  - Otherwise, the return value of spatial_scalability_type ( ) is equal to 6.

intra_base_quality_mb(mbAddr) is specified as follows.

- The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to (quality_level – 1) as input and the output is assigned to baseQualitySlice.
- The return value of intra_base_quality_mb(mbAddr) is derived as follows.
  - If the macroblock with address mbAddr in the slice baseQualitySlice is an I macroblock, the return value of intra_base_quality_mb is TRUE.
  - Otherwise, the return value of intra_base_quality_mb is FALSE.

base_quality_mb_part_pred_mode(mbAddr, mbPartIdx) is specified as follows.

- The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to (quality_level – 1) as input and the output is assigned to baseQualitySlice.
- Let predFlagL0Base and predFlagL1Base be the values of the prediction utilization flags predFlagL0 and predFlagL1, respectively, of the macroblock or sub-macroblock partition specified by mbAddr\mbPartIdx in the slice baseQualitySlice.
- The return value of base_quality_mb_part_pred_mode(mbAddr, mbPartIdx) is derived as follows.
  - If predFlagL0Base is equal to 1 and predFlagL1Base is equal to 1, the return value of base_quality_mb_part_pred_mode(mbAddr, mbPartIdx) is BiPred.

- Otherwise, if predFlagL0Base is equal to 1, the return value of base_quality_mb_part_pred_mode(mbAddr, mbPartIdx) is Pred_L0.
- Otherwise, the return value of base_quality_mb_part_pred_mode(mbAddr, mbPartIdx) is Pred_L1.

more_cabac_data_in_nal_unit( ) is specified as follows.

- If more_rbsp_data( ) returns true, the return value of this function is TRUE.
- Otherwise, the return value of this function is FALSE.

base_quality_mb_transform_size_specified(mbAddr) is specified as follows.

- For any value i with i = 0..quality_level – 1 the following applies.
  - The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].
- Let mbTypeBaseQuality be the syntax element mb_type of the macroblock mbAddr in the slice baseQualitySlice[0].
- Let subMbTypeBaseQuality[k] with k = 0..3 be the syntax element sub_mb_type[k] of the macroblock mbAddr in the slice baseQualitySlice[0].
- The return value of base_quality_mb_transform_size_specified(mbAddr) is derived as follows.
  - If any of the following conditions are true, the return value of base_quality_mb_transform_size_specified(mbAddr) is equal to TRUE.
    - MbPartPredMode(mbTypeBaseQuality, 0) is equal to Intra_4x4, Intra_8x8, or Intra_16x16.
    - mbTypeBaseQuality is equal to P_8x8 or P_8x8ref0, and any of the variables subMbTypeBaseQuality[k] with k = 0..3 is not equal to P_L0_8x8.
    - mbTypeBaseQuality is equal to B_8x8 and any of the following conditions is true.
      - NumSubMbPart(subMbTypeBaseQuality[k]) is greater than 1 for any k = 0...3.
      - SubMbPredMode(subMbTypeBaseQuality[k]) is equal to Direct for any k = 0..3, and direct_8x8_inference_flag is equal to 0.
    - The syntax element transform_size_8x8_flag was transmitted for the macroblock with macroblock address mbAddr in any of the slices baseQualitySlice[i] with i = 0.. quality_level – 1.
  - Otherwise, the return value of base_quality_mb_transform_size_specified(mbAddr) is equal to FALSE.

base_quality_transform_size_8x8_flag(mbAddr) is specified as follows.

- The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to quality_level – 1 as input and the output is assigned to baseQualitySlice.
- The return value of base_quality_transform_size_8x8_flag(mbAddr) is equal to the syntax element transform_size_8x8_flag of the macroblock with macroblock address mbAddr in the base quality slice baseQualitySlice.

base_mb_sig(mbAddr) is specified as follows.

- For any value i with i = 0..quality_level – 1 the following applies.
  - The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].
- The return value of base_mb_sig(mbAddr) is derived as follows.
  - If any of the following conditions is true for any of the slices baseQualitySlice[i] with i = 0..quality_level – 1, the return value of base_mb_sig(mbAddr) is equal to TRUE.
    - The variable CodedBlockPatternLuma is greater than 0.
    - The variable CodedBlockPatternChroma is greater than 0.
  - Otherwise, the return value of base_mb_sig(mbAddr) is equal to FALSE.

base_luma_cbp_bit(mbAddr, i8x8) is specified as follows.

- The derivation process for base quality slice in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to (quality_level – 1) as input and the output is assigned to baseQualitySlice.
- Let baseCodedBlockPatternLuma be equal to CodeBlockPatternLuma of the Macroblock with address mbAddr in the slice baseQualitySlice.
- The return value of base_luma_cbp_bit(mbAddr, i8x8) is ((baseCodedBlockPatternLuma >> i8x8) & 1).

base_luma_level(mbAddr, i8x8, i4x4, scanIdx) is specified as follows.

- A variable i is set equal to 0.
- A variable result is set equal to 0.
- The return value of base_luma_level(mbAddr, i8x8, i4x4, scanIdx) is derived as follows.
  a. The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].
  b. When the transform coefficient level LumaLevel[4 * i8x8 + i4x4][scanIdx] of the macroblock mbAddr is not equal to 0, the following applies
     - The variable result is set equal to LumaLevel[4 * i8x8 + i4x4][scanIdx].
     - The derivation process is continued with step d.
  c. When the variable i is less than (quality_level – 1), the following applies.
     - The variable i is incremented by 1: i = i + 1.
     - The derivation process is continued with step a.
  d. The return value of base_luma_level(mbAddr, i8x8, i4x4, scanIdx) is set equal to the value of the variable result

base_luma_level8x8(mbAddr, i8x8, scanIdx8x8) is specified as follows.

- A variable i is set equal to 0.
- A variable result is set equal to 0.
- The return value of base_luma8x8_level(mbAddr, i8x8, scanIdx8x8) is derived as follows.
  a. The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].
  b. When the transform coefficient level LumaLevel8x8[i8x8][scanIdx8x8] of the macroblock mbAddr is not equal to 0, the following applies
     - The variable result is set equal to LumaLevel8x8[i8x8][scanIdx8x8].
     - The derivation process is continued with step d.
  c. When the variable i is less than (quality_level – 1), the following applies.
     - The variable i is incremented by 1: i = i + 1.
     - The derivation process is continued with step a.
  d. The return value of base_luma8x8_level(mbAddr, i8x8, scanIdx8x8) is set equal to the value of the variable result.

base_chromaDC_level(mbAddr, iCbCr, dcIdx) is specified as follows.

- A variable i is set equal to 0.
- A variable result is set equal to 0.
- The return value of base_chromaDC_level(mbAddr, iCbCr, dcIdx) is derived as follows.
  a. The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].

b. When the transform coefficient level ChromaDCLevel[iCbCr][dcIdx] of the macroblock mbAddr is not equal to 0, the following applies

– The variable result is set equal to ChromaDCLevel[iCbCr][dcIdx].

– The derivation process is continued with step d.

c. When the variable i is less than (quality_level – 1), the following applies.

– The variable i is incremented by 1: i = i + 1.

– The derivation process is continued with step a.

d. The return value of base_chromaDC_level(mbAddr, iCbCr, dcIdx) is set equal to the value of the variable result

base_chromaAC_level(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is specified as follows.

– A variable i is set equal to 0.

– A variable result is set equal to 0.

– The return value of base_chromaAC_level(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is derived as follows.

a. The derivation process for base quality slices in subclause G.6.1 is invoked with mbAddr and baseQualityLevel equal to i as input and the output is assigned to baseQualitySlice[i].

b. When the transform coefficient level ChromaACLevel[iCbCr][4 * i8x8 + i4x4][scanIdx] of the macroblock mbAddr is not equal to 0, the following applies

– The variable result is set equal to ChromaACLevel[iCbCr][4 * i8x8 + i4x4][scanIdx].

– The derivation process is continued with step d.

c. When the variable i is less than (quality_level – 1), the following applies.

– The variable i is incremented by 1: i = i + 1.

– The derivation process is continued with step a.

d. The return value of base_chromaAC_level(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is set equal to the value of the variable result

sig_luma_coef(mbAddr, i8x8, i4x4, scanIdx) is specified as follows.

– The return value of sig_luma_coef(mbAddr, i8x8, i4x4, scanIdx) is derived as follows.

– If any of the following is true, the return value of sig_luma_coef(mbAddr, i8x8, i4x4, scanIdx) is set equal to 0

– The transform coefficient level LumaLevel[4 * i8x8 + i4x4][scanIdx] is equal to zero

– The value of base_luma_level(mbAddr, i8x8, i4x4, scanIdx) is not equal to zero

– Otherwise, the return value of sig_luma_coef(mbAddr, i8x8, i4x4, scanIdx) is set equal to 1

sig_luma_coef8x8(mbAddr, i8x8, scanIdx8x8) is specified as follows.

– The return value of sig_luma_coef8x8(mbAddr, i8x8, scanIdx8x8) is derived as follows.

– If any of the following is true, the return value of sig_luma_coef8x8(mbAddr, i8x8, scanIdx8x8) is set equal to 0

– the transform coefficient level LumaLevel8x8[i8x8][scanIdx8x8] is equal to zero

– the value of base_luma_level8x8(mbAddr, i8x8, scanIdx8x8) is not equal to zero

– Otherwise, the return value of sig_luma_coef8x8(mbAddr, i8x8, scanIdx8x8) is set equal to 1

sig_chromaDC_coef(mbAddr, iCbCr, dcIdx) is specified as follows.

– The return value of sig_chromaDC_coef(mbAddr, iCbCr, dcIdx) is derived as follows.

- If any of the following is true, the return value of sig_chromaDC_coef(mbAddr, iCbCr, dcIdx) is set equal to 0
  - the transform coefficient level ChromaDCLevel[iCbCr][dcIdx] is equal to zero
  - the value of base_chromaDC_level(mbAddr, iCbCr, dcIdx) is not equal to zero
- Otherwise, the return value of sig_chromaDC_coef(mbAddr, iCbCr, dcIdx) is set equal to 1

sig_chromaAC_coef(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is specified as follows.

- The return value of sig_chromaAC_coef(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is derived as follows.
  - If any of the following is true, the return value of sig_chromaAC_coef(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is set equal to 0
    - the transform coefficient level ChromaACLevel[iCbCr][4 * i8x8 + i4x4][scanIdx] is equal to zero
    - the value of base_chromaAC_level(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is not equal to zero
  - Otherwise, the return value of sig_chromaAC_coef(mbAddr, iCbCr, i8x8, i4x4, scanIdx) is set equal to 1

in_crop_window(mbAddr) is specified as follows.

- The variables mbIdxX, mbIdxY0 and mbIdxY1 are derived as follows.
  - mbIdxX is set equal to CurrMbAddr % PicWidthInMbs
  - Depending on MbaffFrameFlag, the following applies.
    - If MbaffFrameFlag is equal to 0,
      - mbIdxY0 and mbIdxY1 are both set equal to CurrMbAddr / PicWidthInMbs
    - Otherwise,
      - mbIdxY0 is set equal to 2 * ((CurrMbAddr / PicWidthInMbs) / 2)
      - mbIdxY1 is set equal to mbIdxY0 + 1
- The variable mbHeight is derived as follows.
  - If field_pic_flag is equal to 1, mbHeight is set equal to 32,
  - Otherwise, mbHeight is set equal to 16.
- If all of the following conditions are true, the return value of in_crop_window ( ) is equal to TRUE.
  - layer_base_flag is equal to 0
  - mbIdxX is greater than or equal to (ScaledBaseLeftOffset + 15) / 16
  - mbIdxX is less than (ScaledBaseLeftOffset + ScaledBaseWidth) / 16
  - mbIdxY0 is greater than or equal to (ScaledBaseTopOffset + mbHeight - 1) / mbHeight
  - mbIdxY1 is less than (ScaledBaseTopOffset + ScaledBaseHeight) / mbHeight
- Otherwise, the return value of in_crop_window ( ) is equal to FALSE.

[Ed. (HS): the next paragraph belong to a different subject… Add a sentence before it.]

- ge(v): context-adaptive variable-length entropy-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause G.9.1.
- tu(v): truncated unary coded syntax element with the left bit first. The parsing process for this descriptor is specified in the section describing semantics of sig_vlc_selector. [Ed (JR) Add refrence]

G.7.3 Syntax in tabular form

The specifications in subclause 7.3 shall apply. Additionally, the following syntax structures are specified.

G.7.3.1 NAL unit header SVC extension syntax

nal_unit_header_svc_extension( ) {	C	Descriptor
reserved_zero_two_bits	All	u(2)
priority_id	All	u(6)
temporal_level	All	u(3)
dependency_id	All	u(3)
quality_level	All	u(2)
reserved_zero_one_bit	All	u(1)
layer_base_flag	All	u(1)
use_base_prediction_flag	All	u(1)
discardable_flag	All	u(1)
fragmented_flag	All	u(1)
last_fragment_flag	All	u(1)
fragment_order	All	u(2)
nalUnitHeaderBytes += 3		
}		

G.7.3.2 Sequence parameter set SVC extension syntax

seq_parameter_set_svc_extension( ) {	C	Descriptor
interlayer_deblocking_filter_control_present_flag	0	u(1)
extended_spatial_scalability	0	u(2)
if (chroma_format_idc > 0) {		
chroma_phase_x_plus1	0	u(2)
chroma_phase_y_plus1	0	u(2)
}		
if(extended_spatial_scalability = = 1) {		
scaled_base_left_offset	0	se(v)
scaled_base_top_offset	0	se(v)
scaled_base_right_offset	0	se(v)
scaled_base_bottom_offset	0	se(v)
}		
fgs_coding_mode	2	u(1)
if(fgs_coding_mode = = 0) {		
groupingSizeMinus1	2	ue(v)
} else {		
numPosVector = 0		
do {		
if(numPosVector = = 0) {		
scanIndex0Minus1	2	ue(v)
}		
else {		

deltaScanIndexMinus1[numPosVector]	2	ue(v)
}		
numPosVector ++		
} while(scanPosVectLuma[numPosVector – 1] < 15)		
}		
}		

G.7.3.3 Slice layer in scalable extension syntax

slice_layer_in_scalable_extension_rbsp( ) {	**C**	**Descriptor**
if (dependency_id = = 0 && quality_level = = 0) {		
if (nal_ref_idc != 0) {		
store_base_rep_flag	2	u(1)
if (use_base_representation_flag && nal_unit_type != 21)		
dec_ref_pic_marking_base( )		
}		
} else {		
slice_header_in_scalable_extension( )	2	
if(slice_type != PR)		
slice_data_in_scalable_extension( )	2 | 3 | 4	
else {		
if(fragmented_flag = = 1) {		
if(fragment_order = = 0)		
NumBytesInPRF = 0		
while(entropy_coding_mode_flag && !byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
if (fragment_order ! = 0) {		
while(rbtmp_byte[NumBytesInPRF] = = 0x00		
NumBytesInPRF--		
NumBytesInPRF--		
}		
while(more_rbsp_trailing_data( ))		
rbtmp_byte[NumBytesInPRF++ **]**	2 | 3 | 4	b(8)
if(next_nalu_not_belongs_to_current_PRF( ))		
swap_buffers(rbtmp_byte, rbsp_byte)		
}		
if (fragmented_flag = = 0 || next_nalu_not_belongs_to_current_PRF ( ))		
progressive_refinement_slice_data_in_scalable_extension( )	2 | 3 | 4	
}		
if (fragmented_flag = = 0 || next_nalu_not_belongs_to_current_PRF ( ))		
rbsp_slice_trailing_bits( )	2	
}		
}		

G.7.3.4 Slice header in scalable extension

slice_header_in_scalable_extension( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type = = 21)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type = = 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type = = 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0 **]**	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1 **]**	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type = = EB)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type != PR) {		
if(slice_type = = EP || slice_type = = EB) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type = = EB)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering( )	2	
if (!layer_base_flag) {		
base_id	2	ue(v)
adaptive_prediction_flag	2	u(1)
}		
if((weighted_pred_flag && slice_type = = EP) || (weighted_bipred_idc = = 1 && slice_type = = EB)) {		
if(adaptive_prediction_flag)		
base_pred_weight_table_flag	2	u(1)
if(layer_base_flag || base_pred_weight_table_flag = = 0)		
pred_weight_table( )		
}		

if(nal_ref_idc != 0) {		
dec_ref_pic_marking( )	2	
if (use_base_representation_flag && nal_unit_type != 21)		
dec_ref_pic_marking_base( )		
}		
if(entropy_coding_mode_flag && slice_type != EI)		
cabac_init_idc	2	ue(v)
}		
if(slice_type != PR || fragment_order = = 0) {		
slice_qp_delta	2	se(v)
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(interlayer_deblocking_filter_control_present_flag) {		
disable_interlayer_deblocking_filter_idc	2	ue(v)
if(disable_interlayer_deblocking_filter_idc != 1) {		
interlayer_slice_alpha_c0_offset_div2	2	se(v)
interlayer_slice_beta_offset_div2	2	se(v)
}		
}		
}		
if(slice_type != PR)		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
if(slice_type != PR && extended_spatial_scalability > 0) {		
if (chroma_format_idc > 0) {		
base_chroma_phase_x_plus1	2	u(2)
base_chroma_phase_y_plus1	2	u(2)
}		
if(extended_spatial_scalability = = 2) {		
scaled_base_left_offset	2	se(v)
scaled_base_top_offset	2	se(v)
scaled_base_right_offset	2	se(v)
scaled_base_bottom_offset	2	se(v)
}		
}		
if(slice_type = = PR && fragment_order = = 0) {		
num_mbs_in_slice_minus1	2	ue(v)
luma_chroma_sep_flag	2	u(1)
store_base_rep_flag	2	u(1)
if (use_base_prediction_flag) {		
adaptive_ref_fgs_flag	2	u(1)

if(adaptive_ref_fgs_flag) {		
max_diff_ref_scale_for_zero_base_block	2	u(5)
max_diff_ref_scale_for_zero_base_coeff	2	u(5)
}		
}		
motion_refinement_flag	2	u(1)
}		
if(slice_type != PR) {		
if(BaseFrameMbsOnlyFlag && !frame_mbs_only_flag && !field_pic_flag)		
base_frame_and_bottom_field_coincided_flag	2	u(1)
else if(frame_mbs_only_flag && !BaseFrameMbsOnlyFlag && !BaseFieldPicFlag)		
base_bottom_field_coincided_flag	2	u(1)
}		
SpatialScalabilityType = spatial_scalability_type( )		
}		

G.7.3.4.1 Decoded reference picture marking syntax for decoded base pictures

dec_ref_pic_marking_base( ) {	**C**	**Descriptor**
base_adaptive_ref_pic_marking_mode_flag	2 | 5	u(1)
if(base_adaptive_ref_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2	ue(v)
if(memory_management_control_operation == 1)		
difference_of_pic_nums_minus1	2	ue(v)
if(memory_management_control_operation == 2)		
long_term_pic_num	2	ue(v)
} while(memory_management_control_operation != 0)		
}		

G.7.3.5 Slice data in scalable extension syntax

slice_data_in_scalable_extension( ) {	C	Descriptor
while(entropy_coding_mode_flag && !byte_aligned( ))		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i = 0; i < mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
moreDataFlag = more_rbsp_data( )		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && (CurrMbAddr % 2 == 0 || (CurrMbAddr % 2 == 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) | ae(v)
macroblock_layer_in_scalable_extension( )	2 | 3 | 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data( )		
else {		
if(slice_type != I)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && (CurrMbAddr % 2 == 0))		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = ! end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

G.7.3.6 Macroblock layer in scalable extension syntax

macroblock_layer_in_scalable_extension( ) {	C	Descriptor
if(in_crop_window(CurrMbAddr) && adaptive_prediction_flag)		
base_mode_flag	2	u(1) | ae(v)
if(! base_mode_flag)		
mb_type	2	ue(v) | ae(v)
if(! base_mode_flag && mb_type == I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	2	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i **]**	2	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i **]**	2	u(v)
} else {		
if(! base_mode_flag) {		
NoSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) == 4) {		
sub_mb_pred_in_scalable_extension(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type [mbPartIdx]) > 1)		
NoSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
NoSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type == I_NxN)		
transform_size_8x8_flag	2	u(1) | ae(v)
mb_pred_in_scalable_extension(mb_type)	2	
}		
}		
if(base_mode_flag || MbPartPredMode(mb_type, 0) != Intra_16x16) {		
coded_block_pattern	2	me(v) | ae(v)
if(CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && (base_mode_flag || (mb_type != I_NxN && NoSubMbPartSizeLessThan8x8Flag && !(MbPartPredMode(mb_type, 0) == B_Direct_16x16 && !direct_8x8_inference_flag))))		
transform_size_8x8_flag	2	u(1) | ae(v)
}		
if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || (! base_mode_flag && MbPartPredMode(mb_type, 0) == Intra_16x16)) {		
mb_qp_delta	2	se(v) | ae(v)
residual_in_scalable_extension( )	3 | 4	
}		
}		

}		

G.7.3.6.1 Macroblock prediction in scalable extension syntax

mb_pred_in_scalable_extension(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) = = Intra_4x4 || MbPartPredMode(mb_type, 0) = = Intra_8x8 || MbPartPredMode(mb_type, 0) = = Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) = = Intra_4x4)		
for(luma4x4BlkIdx = 0; luma4x4BlkIdx < 16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag[luma4x4BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra4x4_pred_mode_flag**[** luma4x4BlkIdx **]**)		
rem_intra4x4_pred_mode[luma4x4BlkIdx **]**	2	u(3) | ae(v)
}		
if(MbPartPredMode(mb_type, 0) = = Intra_8x8)		
for(luma8x8BlkIdx = 0; luma8x8BlkIdx < 4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag[luma8x8BlkIdx **]**	2	u(1) | ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode[luma8x8BlkIdx **]**	2	u(3) | ae(v)
}		
if(chroma_format_idc != 0)		
intra_chroma_pred_mode	2	ue(v) | ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
if(layer_base_flag == 0) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx **]**	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx **]**	2	u(1) | ae(v)
}		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1 && ! motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0 && ! motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)

}		
}		

G.7.3.6.2 Sub-macroblock prediction in scalable extension syntax

sub_mb_pred_in_scalable_extension(MbType) {	C	Descriptor
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ue(v) | ae(v)
if(layer_base_flag == 0) {		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx]	2	u(1) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx]	2	u(1) | ae(v)
}		
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1 && !motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0 && !motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	se(v) | ae(v)
}		

G.7.3.6.3 Residual in scalable extension syntax

residual_in_scalable_extension( ) {	C	Descriptor
if (adaptive_prediction_flag && slice_type != I && slice_type != IE && (base_mode_flag || (MbPartPredType(mb_type, 0) != Intra_16x16 && MbPartPredType(mb_type, 0) != Intra_8x8 && MbPartPredType(mb_type, 0) != Intra_4x4))) {		
residual_prediction_flag	3 | 4	u(1) | ae(v)
if (residual_prediction_flag && base_mode_flag)		
smoothed_reference_flag	3 | 4	u(1) | ae(v)
}		
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
if(! base_mode_flag && MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block_cabac(Intra16x16DCLevel, 16)	3	
for(i8x8 = 0; i8x8 < 4; i8x8++) /* each luma 8x8 block */		
if(!transform_size_8x8_flag)		
for(i4x4 = 0; i4x4 < 4; i4x4++) { /* each 4x4 sub-block of block */		
if(CodedBlockPatternLuma & (1 << i8x8))		
if(! base_mode_flag && MbPartPredMode(mb_type, 0) = = Intra_16x16)		
residual_block(Intra16x16ACLevel[i8x8 * 4 + i4x4], 15)	3	
else		
residual_block(LumaLevel[i8x8 * 4 + i4x4], 16)	3 | 4	
else if(! base_mode_flag && MbPartPredMode(mb_type, 0) = = Intra_16x16)		
for(i = 0; i < 15; i++)		
Intra16x16ACLevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
LumaLevel[i8x8 * 4 + i4x4][i] = 0		
}		
else if(CodedBlockPatternLuma & (1 << i8x8))		
residual_block(LumaLevel8x8[i8x8], 64)	3 | 4	
else		
for(i = 0; i < 64; i++)		
LumaLevel8x8[i8x8][i] = 0		
if(chroma_format_idc != 0) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
if(CodedBlockPatternChroma & 3) /* chroma DC residual present */		
residual_block(ChromaDCLevel[iCbCr], 4 * NumC8x8)	3 | 4	
else		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[iCbCr][i] = 0		

for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
if(CodedBlockPatternChroma & 2) /* chroma AC residual present */		
residual_block(ChromaACLevel[iCbCr][i8x8*4+i4x4], 15)	3 | 4	
else		
for(i = 0; i < 15; i++)		
ChromaACLevel[iCbCr][i8x8*4+i4x4][i] = 0		
}		

G.7.3.7 Progressive refinement slice data in scalable extension syntax

progressive_refinement_slice_data_in_scalable_extension( ) {	C	Descriptor
PRsliceCompletelyDecoded = 0		
startMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
lastMbAddr = startMbAddr		
for(i = 0; i < num_mbs_in_slice_minus1; i++)		
lastMbAddr = NextMbAddr(lastMbAddr)		
CurrMbAddr = startMbAddr		
while(CurrMbAddr <= lastMbAddr) { /* initialisation */		
for(i8x8 = 0; i8x8 < 4; i8x8++) {		
for(scanIdx8x8 = 0; scanIdx8x8 < 64; scanIdx8x8++)		
LumaLevel8x8[i8x8][scanIdx8x8] = NotCoded		
for(i4x4 = 0; i4x4 < i; i4x4++) {		
coded_block_flag_luma[4 * i8x8 + i4x4] = NotCoded		
for(scanIdx = 0; scanIdx < 16; scanIdx++)		
LumaLevel[4*i8x8 + i4x4][scanIdx] = NotCoded		
}		
}		
for(i = 0; i < 6; i++)		
coded_block_pattern_bit[i] = NotCoded		
mb_qp_delta = NotCoded		
if(transform_8x8_mode_flag) {		
if(base_quality_mb_transform_size_specified(CurrMbAddr)		
transform_size_8x8_flag = base_quality_transform_size_8x8_flag(CurrMbAddr)		
else		
transform_size_8x8_flag = NotCoded		
} else		
transform_size_8x8_flag = 0		
TransformSize8x8Flag = (transform_size_8x8_flag = = NotCoded ? 0 : transform_size_8x8_flag)		
if(chroma_format_idc != 0) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
for(iCbCr = 0; iCbCr < 2; iCbCr++) {		
coded_block_flag_chromaDC[iCbCr] = NotCoded		
for(dcIdx = 0; dcIdx < 4 * NumC8x8; dcIdx++)		
ChromaDCLevel[iCbCr][dcIdx] = NotCoded		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++) {		
coded_block_flag_chromaAC[iCbCr][4 * i8x8 + i4x4] = NotCoded		
for(scanIdx = 0; scanIdx < 15; scanIdx++)		
ChromaACLevel[iCbCr][4 * i8x8 + i4x4][scanIdx] = NotCoded		
}		

}		
}		
MotionDataCodedFlag = 0		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
}		
scanIdxLuma = 0		
scanIdxChromaDC = 0		
scanIdxChromaAC = 1		
cycleNum = 0		
chromaStartCycle = 1		
if(entropy_coding_mode_flag = = 0) {		
num_end_vals /* luma EOB offsets */	3 | 4	te(v)
num_end_vals = (num_end_vals + 1) % 4		
for(i = 0; i < num_end_vals; i++)		
eobShiftLuma[i] = 15		
eobShiftLuma[num_end_vals **]**	3 | 4	ue(v)
startLevel = eobShiftLuma[num_end_vals] – 1		
for(i = 0; i < 14 – num_end_vals && startLevel > 0;) {		
eob_run	3 | 4	ue(v)
for(j = 0; j < eob_run; j++)		
eobShiftLuma[num_end_vals + 1 + i + j] = startLevel		
startLevel--		
i += eob_run		
}		
num_end_vals /* chroma EOB offsets */	3 | 4	te(v)
num_end_vals = (num_end_vals + 1) % 4		
for(i = 0; i < num_end_vals; i++)		
eobShiftLuma[i] = 15		
eob_shift_chroma[num_end_vals **]**	3 | 4	ue(v)
startLevel = eob_shift_chroma[num_end_vals] – 1		
for(i = 0; i < 13 – num_end_vals && startLevel > 0;) {		
eob_run	3 | 4	ue(v)
for(j = 0; j < eob_run; j++)		
eob_shift_chroma[num_end_vals + 1 + i + j] = startLevel		
startLevel--		
i += eob_run		
}		
for(i = 0; i < 16; i ++)		
sig_vlc_selector[i] = 0		
num_sig_vlc_selectors	3 | 4	u(4)
for(i = 0; i < num_sig_vlc_selectors; i ++)		
sig_vlc_selector[i]	3 | 4	tu(v)
init_luma_cbp_flag	3 | 4	u(1)
current_luma_cbp_flag = init_luma_cbp_flag		
luma_cbp_run = 0		

init_chroma_cbp_flag	3 | 4	u(1)
current_chroma_cbp = (init_chroma_cbp_flag) ? 1 : 0		
chroma_cbp_run = 0		
chroma_cbp_coded_flag = 0		
}		
while (scanIdxLuma < 16 || scanIdxChromaDC < 4 || scanIdxChromaAC < 16) {		
allowChromaDCFlag = (cycleNum = = 0) || ((cycleNum >= chromaStartCycle) && ((chromaStartCycle – cycleNum) % 2 = = 0))		
allowChromaACFlag = (cycleNum > 0) && ((cycleNum = = chromaStartCycle) || ((cycleNum >= chromaStartCycle) && ((cycleNum – chromaStartCycle) % 3 = = 1)))		
CurrMbAddr = startMbAddr		
while(CurrMbAddr <= lastMbAddr) {		
if(motion_refinement_flag && ! MotionDataCodedFlag && ! intra_base_quality_mb(CurrMbAddr)) {		
motion_data_in_progressive_refinement_slices( )	3 | 4	
MotionDataCodedFlag = 1		
}		
if(entropy_coding_mode_flag = = 0) {		
if(scanIdxLuma == 0)		
mb_luma_cbp(CurrMbAddr, luma_cbp_run, current_luma_cbp_flag)	3 | 4	
if(!chroma_cbp_coded_flag && (allowChromaDCFlag || allowChromaACFlag))		
mb_chroma_cbp(chroma_cbp_run, current_chroma_cbp)	3 | 4	
}		
for(blkIdx4x4 = 0; blkIdx4x4 < 16; blkIdx4x4++) { /* luma */		
if (scanIdxLuma < 16) {		
currScanPos = scanIdxLuma		
while(currScanPos < scanPosVectLuma[cycleNumLuma + 1]) {		
if (LumaLevel[blkIdx4x4][currPosScan] = = NotCoded) {		
if(! TranformSize8x8Flag) {		
if (base_luma_level(CurrMbAddr, blkIdx4x4 / 4, blkIdx4x4 % 4, currScanPos) = = 0)		
luma_coefficient(blkIdx4x4, currScanPos)	3 | 4	
else		
luma_coefficient_ref(blkIdx4x4, currScanPos)	3 | 4	
}		
else {		
i8x8 = blkIdx4x4 / 8 * 2 + (blkIdx4x4 % 4) / 2		
offset = 2*((blkIdx4x4 / 4)%2) + ((blkIdx4x4%4)%2)		
if (base_luma_level8x8(CurrMbAddr, i8x8, currScanPos *4 + offset) = = 0)		
luma_coefficient(blkIdx4x4, currScanPos)	3 | 4	
else		
luma_coefficient_ref(blkIdx4x4, currScanPos)	3 | 4	

}		
}		
currScanPos++		
}		
}		
}		
if(chroma_format_idc != 0 && (luma_chroma_sep_flag == 0 || scanIdxLuma == 16)) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
if(allowChromaDCFlag) /* chromaDC */		
for(iCbCr = 0; iCbCr < 2; iCbCr++) {		
if(scanIdxChromaDC < 4) {		
currScanPos = scanIdxChromaDC		
while(currScanPos < scanPosVectChromaDC[cycleNumChromaDC + 1]) {		
if(ChromaDCLevel[iCbCr][currScanPos] == NotCoded) {		
if(base_chroma_DC_level(CurrMbAddr, iCbCr, currScanPos) == 0)		
chromaDC_coefficient(iCbCr)	3 | 4	
else		
chromaDC_coefficient_ref(iCbCr, currScanPos)	3 | 4	
}		
currScanPos++		
}		
}		
}		
if(allowChromaACFlag) /* chromaAC */		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(blkIdx4x4 = 0; blkIdx4x4 < 4*NumC8x8; blkIdx4x4++) {		
if(scanIdxChromaAC < 16) {		
currScanPos = scanIdxChromaAC		
while(currScanPos < scanPosVectLuma[cycleNumChromaAC + 1]) {		
if(ChromaACLevel[iCbCr][currScanPos] == NotCoded) {		
if(base_chroma_AC_level(CurrMbAddr, iCbCr, currScanPos) == 0)		
chromaAC_coefficient(iCbCr, blkIdx4x4)	3 | 4	
else		
chromaAC_coefficient_ref(iCbCr, blkIdx4x4, currScanPos)	3 | 4	
}		
currScanPos++		
}		
}		
}		
}		
CurrMbAddr = NextMbAddr(CurrMbAddr)		
}		

scanIdxLuma = scanPosVectLuma[cycleNumLuma + 1]		
if(allowChromaDCFlag)		
scanIdxChromaDC = scanPosVectChromaDC[cycleNumChromaDC + 1]		
if(allowChromaACFlag)		
scanIdxChromaAC = scanPosVectLuma[cycleNumChromaAC + 1]		
if(cycleNum = = 0 && luma_chroma_sep_flag = = 0) {		
cycle_start_symbol	3 | 4	u(1) | ae(v)
if(cycle_start_symbol = = 1) {		
cycle_start_symbol_ext	3 | 4	u(1) | ae(v)
chromaStartCycle = 2 + cycle_start_symbol_ext		
}		
}		
if(scanIdxLuma < 16)		
cycleNumLuma ++		
if(allowChromaDCFlag && scanIdxChromaDC < 4)		
cycleNumChromaDC ++		
if(allowChromaACFlag && scanIdxChromaAC < 16)		
cycleNumChromaAC ++		
if(entropy_coding_mode_flag = = 0)		
if (&& (! chroma_cbp_coded_flag) && (allowChromaDCFlag || allowChromaACFlag))		
chroma_cbp_coded_flag = 1		
cycleNum++		
}		
if(entropy_coding_mode_flag && more_cabac_data_in_nal_unit( )) {		
end_of_slice_flag	2	ae(v)
PRsliceCompletelyDecoded = end_of_slice_flag		
} else		
PRsliceCompletelyDecoded = 1		
}		

G.7.3.7.1 Motion data in progressive refinement slice data syntax

motion_data_in_progressive_refinement_slices( ) {	C	Descriptor
base_mode_flag	2	u(1) | ae(v)
if(! base_mode_flag) {		
mb_type	2	ue(v) | ae(v)
noSubMbPartSizeLessThan8x8Flag = 1		
if(NumMbPart(mb_type) = = 4) {		
sub_mb_pred_in_progressive_refinement_slice(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type [mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else		
mb_pred_in_progressive_refinement_slice(mb_type)	2	
residual_prediction_flag	3 | 4	u(1) | ae(v)
if(! residual_prediction_flag && noSubMbPartSizeLessThan8x8Flag = = 1)		
transform_size_8x8_flag = NotCoded		
else if(noSubMbPartSizeLessThan8x8Flag = = 0)		
transform_size_8x8_flag = 0		
}		
}		

G.7.3.7.1.1 Macroblock prediction in progressive refinement slice data syntax

mb_pred_in_progressive_refinement_slice(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L1 && base_quality_mb_part_pred_mode(CurrMbAddr, mbPartIdx) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0 && base_quality_mb_part_pred_mode(CurrMbAddr, mbPartIdx) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1 && ! motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0 && ! motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	ae(v)
}		
}		

G.7.3.7.1.2 Sub-macroblock in progressive refinement slice data syntax

sub_mb_pred_in_progressive_refinement_slice(MbType) {	C	Descriptor
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1 && base_quality_mb_part_pred_mode(CurrMbAddr, mbPartIdx) != Pred_L1)		
motion_prediction_flag_l0[mbPartIdx]	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0 && base_quality_mb_part_pred_mode(CurrMbAddr, mbPartIdx) != Pred_L0)		
motion_prediction_flag_l1[mbPartIdx]	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1 && !motion_prediction_flag_l0[mbPartIdx])		
ref_idx_l0[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0 && !motion_prediction_flag_l1[mbPartIdx])		
ref_idx_l1[mbPartIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** subMbPartIdx **][** compIdx **]**	2	ae(v)
}		

G.7.3.7.2 Macroblock Luma Coded Block Pattern in progressive slice data CAVLC syntax

mb_luma_cbp(mbAddr, luma_cbp_run, current_luma_cbp_flag) {		
for(i8x8 = 0; i8x8 < 4; i8x8 ++) {		
if(base_luma_cbp_bit(mbAddr, i8x8) != 0)		
coded_block_pattern_bit[i8x8] = base_luma_cbp_bit(mbAddr, i8x8)		
else {		
if(luma_cbp_run == 0) {		
current_luma_cbp_flag = 1 - current_luma_cbp_flag		
luma_cbp_run_minus1	3 | 4	ue(v)
luma_cbp_run = luma_cbp_run_minus1 + 1		
}		
coded_block_pattern_bit[i8x8] = current_luma_cbp_flag		
luma_cbp_run--		
}		
}		
}		

G.7.3.7.3 Macroblock Chroma Coded Block Pattern in progressive slice data CAVLC syntax

mb_chroma_cbp(chroma_cbp_run, current_chroma_cbp) {		
if(chroma_cbp_run == 0) {		
chroma_cbp_transition_flag	3 | 4	u(1)
current_chroma_cbp = (current_chroma_cbp + chroma_cbp_transition_flag + 1) % 3		
chroma_cbp_run_minus1	3 | 4	ue(v)
chroma_cbp_run = chroma_cbp_run_minus1 + 1		
}		
coded_block_pattern_bit[4] = (current_chroma_cbp != 0)		
coded_block_pattern_bit[5] = (current_chroma_cbp > 1)		
chroma_cbp_run --		
}		

G.7.3.7.4 Luma coefficient in progressive refinement slice data syntax

luma_coefficient(blkIdx4x4, inScanIdx) {	C	Descriptor
i8x8 = blkIdx4x4 / 4		
i4x4 = blkIdx4x4 % 4		
if(more_cabac_data_in_nal_unit( ) && coded_block_pattern_bit[i8x8] = = NotCoded) {		
if(entropy_coding_mode_flag)		
coded_block_pattern_bit[i8x8 **]**	3 | 4	ae(v)
if(coded_block_pattern_bit[i8x8] = = 0) {		
for(j = 0; j < 4; j++)		
for(i = 0; i < 16; i++)		
LumaLevel[4 * i8x8 + j][i] = 0		
for(i = 0; i < 64; i++)		
LumaLevel8x8[i8x8][i] = 0		
} else {		
if(more_cabac_data_in_nal_unit( ) && ! base_mb_sig(CurrMbAddr) && mb_qp_delta = = NotCoded)		
mb_qp_delta	2	se(v) | ae(v)
if(more_cabac_data_in_nal_unit( ) && transform_size_8x8_flag = = NotCoded) {		
transform_size_8x8_flag	2	u(1) | ae(v)
TransformSize8x8Flag = transform_size_8x8_flag		
}		
}		
}		
if(coded_block_pattern_bit[i8x8] = = 1) {		
firstCoeff = 0		
if(more_cabac_data_in_nal_unit( ) && coded_block_flag_luma[blkIdx4x4] = = NotCoded) {		
coded_block_flag_luma[blkIdx4x4 **]**	3 | 4	ge(v) | ae(v)
firstCoeff = 1		
if(coded_block_flag_luma[blkIdx4x4] = = 0)		
for(i = 0; i < 16; i++) {		
LumaLevel[blkIdx4x4][i] = 0		
LumaLevel8x8[i8x8][i4x4 + 4 * i] = 0		
}		
}		
if(coded_block_flag_luma[blkIdx4x4] = = 1) {		
stride = 1 + 3 * TransformSize8x8Flag		
stopIdx = 16 * stride		
scanIdx = inScanIdx / 4		
numRemain = 0		
minScanPos = 16		
for(idx = inScanIdx; idx < stopIdx; idx += stride) {		
if((TransformSize8x8Flag && base_luma_level8x8(CurrMbAddr, i8x8, idx) = = 0) || (! TransformSize8x8Flag && base_luma_level(CurrMbAddr, i8x8, idx) = = 0))		

numRemain++		
if(TransformSize8x8Flag && minScanPos = = 16 && ((base_luma_level8x8(CurrMbAddr, 0, idx) = = 0) || (base_luma_level8x8(CurrMbAddr, 1, idx) = = 0) || (base_luma_level8x8(CurrMbAddr, 2, idx) = = 0) || (base_luma_level8x8(CurrMbAddr, 3, idx) = = 0)))		
minScanPos = idx		
}		
residual_mag = 0		
numCoef = 1		
if(entropy_coding_mode_flag = = 0) {		
sigVlcSelector = sig_vlc_selector[scanIdx]		
if(TransformSize8x8Flag && (numRemain = = 2 || (minScanPos > 10 && numRemain = = 3))) {		
eo8b_symbol	2	u(1)
if(eo8b_symbol)		
numCoef = 0		
}		
if(numCoef = = 1)		
significant_coefficient_cavlc(scanIdx, firstCoeff, numRemain, sigVlcSelector, eobShiftLuma[scanIdx + 1], numCoef, trmCoef)	3 | 4	
} else		
significant_coefficient_cabac(scanIdx, firstCoeff, numCoef, trmCoef)	3 | 4	
if(numCoef = = 0) {		
for(i = inScanIdx; i < stopIdx; i += stride)		
if(! TransformSize8x8Flag)		
LumaLevel[blkIdx4x4][i] = 0		
else		
LumaLevel8x8[i8x8][i]		
residual_mag = ! entropy_coding_mode_flag		
} else {		
if(! TransformSize8x8Flag) {		
for(i = 0; i < numCoef; i++)		
LumaLevel[blkIdx4x4][i + scanIdx] = 0		
LumaLevel[blkIdx4x4][numCoef + scanIdx] = trmCoef		
} else {		
for(i = 0; i < numCoef; i++)		
LumaLevel8x8[i8x8][inScanIdx + i] = 0		
LumaLevel8x8[i8x8][inScanIdx + numCoef] = trmCoef		
}		
if(! entropy_coding_mode_flag && (numCoef = = numRemain)) {		
residual_mag = 1		
coeff_sig_vlc_symbol	3 | 4	ge(v)
trmCoef = coeff_sig_vlc_symbol		
}		
}		
if(residual_mag) { /* magnitudes greater than 1 in CAVLC */		
mag1_count = 0		

for(i = 0; i < 16; i++)		
if((! TransformSize8x8Flag && sig_luma_coef(CurrMbAddr, i8x8, i4x4, i)) || (TransformSize8x8Flag && sig_luma_coef8x8(CurrMbAddr, i8x8, i4x4 + 4 * i)))		
mag1_count++		
if(trmCoef < mag1_count * 2) {		
max_mag = (trmCoef % 2) + 2		
mag2_count = (trmCoef / 2) + 1		
} else {		
max_mag = (trmCoef / mag1_count) + 2		
mag2_count = (trmCoef % mag1_count) + 1		
}		
mag1_remain = mag1_count		
mag2_remain = mag2_count		
for(i = 0; i < 16 && mag2_remain > 0; i++)		
if((! TransformSize8x8Flag && sig_luma_coef(CurrMbAddr, i8x8, i4x4, i)) || (TransformSize8x8Flag && sig_luma_coef8x8(CurrMbAddr, i8x8, i4x4 + 4 * i))) {		
if(mag2_remain < mag1_remain)		
sig_mag_flag	3 | 4	u(1)
else		
sig_mag_flag = 1		
if(sig_mag_flag) {		
if(! TransformSize8x8Flag)		
LumaLevel[blkIdx4x4][i] = LumaLevel[blkIdx4x4][i] < 0 ? SigMag1 : SigMag0		
else		
LumaLevel8x8[i8x8][i4x4 + 4 * i] = LumaLevel8x8[i8x8][i4x4 + 4 * i] < 0 ? SigMag1 : SigMag0		
mag2_remain--		
}		
mag1_remain--		
}		
mag2_remain = mag2_count		
seen_max = 0		
for(i = 0; i < 16 && mag2_remain > 0; i++)		
if((! TransformSize8x8Flag && (LumaLevel[blkIdx4x4][i] = = SigMag1 || LumaLevel[blkIdx4x4][i] = = SigMag0)) || (TransformSize8x8Flag && (LumaLevel8x8[i8x8][i4x4 + 4 * i] = = SigMag1 || LumaLevel8x8[i8x8][i4x4 + 4 * i] = = SigMag0))) {		
if(seen_max || mag2_remain > 1) {		
coef_mag = 2		
for(j = 1; j < max_mag; j++) {		
coef_mag_ref_bit	3 | 4	u(1)
if(coef_mag_ref_bit)		
coef_mag++		
else		

j = max_mag		
}		
} else		
coef_mag = max_mag		
if(! TransformSize8x8)		
LumaLevel[blkIdx4x4][i] = (LumaLevel[blkIdx4x4][i] = = SigMag1) ? –coef_mag : coef_mag		
else		
LumaLevel8x8[i8x8][i4x4 + 4 * i] = (LumaLevel8x8[i8x8][i4x4 + 4 * i] = = SigMag1) ? –coeff_mag : coeff_mag		
mag2_remain--		
seen_max = (coef_mag = = max_mag) ? 1 : seen_max		
}		
}		
}		
}		
}		
}		

G.7.3.7.5 Chroma DC coefficient in progressive slice data syntax

chromaDC_coefficient(iCbCr) {	C	Descriptor
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
dcIdx = 4		
dcIdxStart = 4		
numRemain = 0		
for(idx = 0; idx < 4; idx++) {		
if(ChromaDCLevel[iCbCr][idx] = = NotCoded && base_chromaDC_level(CurrMbAddr, iCbCr, idx) = = 0) {		
if(dcIdx = = 4)		
dcIdx = idx		
if(idx > = dcIdx)		
numRemain++		
}		
}		
firstCoeff = 0		
if(more_cabac_data_in_nal_unit( ) && coded_block_pattern_bit[4] = = NotCoded) {		
firstCoeff = 1		
if(entropy_coding_mode_flag)		
coded_block_pattern_bit[4 **]**	3 | 4	ae(v)
if(coded_block_pattern_bit[4] = = 0) {		
for(j = 0; j < 2; j++) {		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[j][i] = 0		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
for(scanIdx = 0; scanIdx < 15; scanIdx++)		
ChromaACLevel[j][4 * i8x8 + i4x4][scanIdx] = 0		
}		
} else {		
if(more_cabac_data_in_nal_unit( ) && ! base_mb_sig(CurrMbAddr) && mb_qp_delta = = NotCoded)		
mb_qp_delta	2	se(v) | ae(v)
}		
}		
if(coded_block_pattern_bit[4] = = 1) {		
if(more_cabac_data_in_nal_unit( ) && coded_block_flag_chromaDC[iCbCr] = = NotCoded) {		
coded_block_flag_chromaDC[iCbCr **]**	3 | 4	u(v) | ae(v)
if(coded_block_flag_chromaDC[iCbCr] == 0)		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[iCbCr][i] = 0		
}		
if(coded_block_flag_chromaDC[iCbCr] = = 1) {		
residual_mag = 0		

if(! entropy_coding_mode_flag) {		
sigVlcSelector = sig_vlc_selector[dcIdx]		
significant_coefficient_cavlc(firstCoeff, numRemain, sigVlcSelector, eob_shift_chroma[dcIdx + 1], numCoef, trmCoef)	3 | 4	
} else		
significant_coefficient_cabac(firstCoeff, numCoef, trmCoef)	3 | 4	
if(numCoef = = 0) {		
for(i = dcIdx; i < 4; i++)		
ChromaDCLevel[iCbCr][i] = 0		
residual_mag = ! entropy_coding_mode_flag		
} else {		
for(i = 0; i < numCoef; i++)		
ChromaDCLevel[iCbCr][i + dcIdx] = 0		
ChromaDCLevel[iCbCr][numCoef + dcIdx] = trmCoef		
eob = 1		
for(i = dcIdx + numCoef + 1; i < 4; i++)		
eob &= base_chromaDC_level(CurrMbAddr, iCbCr, i) ! = 0		
if(! entropy_coding_flag && eob) {		
residual_mag = 1		
coeff_sig_vlc_symbol	3 | 4	ge(v)
trmCoef = coeff_sig_vlc_symbol		
}		
}		
if(residual_mag) { /* magnitudes greater than 1 in CAVLC */		
mag1_count = 0		
for(i = 0; i < 4; i++)		
if(sig_chromaDC_coef(CurrMbAddr, iCbCr, i))		
mag1_count++		
if(trmCoef < mag1_count * 2) {		
max_mag = (trmCoef % 2) + 2		
mag2_count = (trmCoef / 2) + 1		
} else {		
max_mag = (trmCoef / mag1_count) + 2		
mag2_count = (trmCoef % mag1_count) + 1		
}		
mag1_remain = mag1_count		
mag2_remain = mag2_count		
for(i = 0; i < 4 && mag2_remain > 0; i++)		
if(sig_chromaDC_coef (CurrMbAddr, iCbCr, i)) {		
if(mag2_remain < mag1_remain)		
sig_mag_flag	3 | 4	u(1)
else		
sig_mag_flag = 1		
if(sig_mag_flag) {		
ChromaDCLevel[iCbCr][i] = ChromaDCLevel[iCbCr][i] < 0 ? SigMag1 : SigMag0		
mag2_remain--		
}		

mag1_remain--		
}		
mag2_remain = mag2_count		
seen_max = 0		
for(i = 0; i < 4 && mag2_remain > 0; i++)		
if(ChromaDCLevel[iCbCr][i] = = SigMag1 | | ChromaDCLevel[iCbCr][i] = = SigMag0) {		
if(seen_max | | mag2_remain > 1) {		
coef_mag = 2		
for(j = 1; j < max_mag; j++) {		
coef_mag_ref_bit	3 | 4	u(1)
if(coef_mag_ref_bit)		
coef_mag++		
else		
j = max_mag		
}		
} else		
coef_mag = max_mag		
ChromaDCLevel[iCbCr][i] = (ChromaDCLevel[iCbCr][i] = = SigMag1) ? –coef_mag : coef_mag		
mag2_remain--		
seen_max = (coef_mag = = max_mag) ? 1 : seen_max		
}		
}		
}		
}		
}		

G.7.3.7.6 Chroma AC coefficient in progressive refinement slice data syntax

chromaAC_coefficient(iCbCr, blkIdx4x4) {	C	Descriptor
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
i8x8 = blkIdx4x4 / 4		
i4x4 = blkIdx4x4 % 4		
scanIdx = 16		
numRemain = 0		
for(idx = 0; idx < 16; idx++) {		
if(ChromaACLevel[iCbCr][blkIdx4x4][idx] == NotCoded && base_chromaAC_level(CurrMbAddr, iCbCr, i8x8, i4x4, idx) = = 0) {		
if(scanIdx = = 16)		
scanIdx = idx		
if(idx > = scanIdx)		
numRemain++		
}		
}		
if(more_cabac_data_in_nal_unit( ) && coded_block_pattern_bit[4] = = NotCoded) {		
if(entropy_coding_mode_flag)		
coded_block_pattern_bit[4 **]**	3 | 4	ae(v)
if(coded_block_pattern_bit[4] = = 0) {		
for(j = 0; j < 2; j++) {		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaACLevel[j][i] = 0		
for(i = 0; i < NumC8x8; i++)		
for(k = 0; k < 4; k++)		
for(l = 0; l < 15; l++)		
ChromaACLevel[j][4 * i + k][l] = 0		
}		
} else {		
if(more_cabac_data_in_nal_unit( ) && ! base_mb_sig(CurrMbAddr) && mb_qp_delta = = NotCoded)		
mb_qp_delta	2	se(v) | ae(v)
}		
}		
firstCoeff = 0		
if(coded_block_pattern_bit[4] = = 1 && more_cabac_data_in_nal_unit( ) && coded_block_pattern_bit[5] = = NotCoded) {		
firstCoeff = 1		
if(entropy_coding_mode_flag)		
coded_block_pattern_bit[5 **]**	3 | 4	ae(v)
if(coded_block_pattern_bit[5] = = 0) {		
for(j = 0; j < 2; j++)		
for(i = 0; i < NumC8x8; i++)		

for(k = 0; k < 4; k++)		
for(l = 0; l < 15; l++)		
ChromaACLevel[j][4 * i + k][l] = 0		
}		
}		
if(coded_block_pattern_bit[5] = = 1) {		
if(more_cabac_data_in_nal_unit( ) && coded_block_flag_chromaAC[iCbCr][blkIdx4x4] = = NotCoded) {		
coded_block_flag_chromaAC[iCbCr **][** blkIdx4x4 **]**	3 | 4	ge(v) | ae(v)
if(coded_block_flag_chromaAC[iCbCr][blkIdx4x4] = = 0)		
for(k = 0; k < 15; k++)		
ChromaACLevel[iCbCr][blkIdx4x4][k] = 0		
}		
if(coded_block_flag_chromaAC[iCbCr][blkIdx4x4] = = 1) {		
residual_mag = 0		
if(entropy_coding_mode_flag = = 0)		
sigVlcSelector = sig_vlc_selector[scanIdx]		
significant_coefficient_cavlc(firstCoeff, signumRemain, VlcSelector, eob_shift_chroma[scanIdx + 1], numCoef, trmCoef)	3 | 4	
else		
significant_coefficient_cabac(firstCoeff, numCoef, trmCoef)	3 | 4	
if(numCoef = = 0) {		
for(i = scanIdx; i < 16; i++)		
ChromaACLevel[iCbCr][blkIdx4x4][i] = 0		
residual_mag = ! entropy_coding_mode_flag		
} else {		
for(i = 0; i < numCoef; i++)		
ChromaACLevel[iCbCr][blkIdx4x4][i + scanIdx] = 0		
ChromaACLevel[iCbCr][blkIdx4x4][numCoef + scanIdx] = trmCoef		
eob = 1		
for(i = scanIdx + numCoef + 1; i < 16; i++)		
eob &= base_chromaAC_level(CurrMbAddr, iCbCr, i8x8, i4x4, i) ! = 0		
if(entropy_coding_mode_flag = = 0 && eob) {		
residual_mag = 1		
coeff_sig_vlc_symbol	3 | 4	ge(v)
trmCoef = coeff_sig_vlc_symbol		
}		
}		
if(residual_mag) { /* magnitudes greater than 1 in CAVLC */		
mag1_count = 0		
for(i = 1; i < 16; i++)		
if(sig_chromaAC_coef(CurrMbAddr, iCbCr, i8x8, i4x4, i))		
mag1_count++		
if(trmCoef < mag1_count * 2) {		
max_mag = (trmCoef % 2) + 2		
mag2_count = (trmCoef / 2) + 1		
} else {		

max_mag = (trmCoef / mag1_count) + 2		
mag2_count = (trmCoef % mag1_count) + 1		
}		
mag1_remain = mag1_count		
mag2_remain = mag2_count		
for(i = 1; i < 16 && mag2_remain > 0; i++)		
if(sig_chromaAC_coef (CurrMbAddr, iCbCr, i8x8, i4x4, i)) {		
if(mag2_remain < mag1_remain)		
sig_mag_flag	3 | 4	u(1)
else		
sig_mag_flag = 1		
if(sig_mag_flag) {		
ChromaACLevel[iCbCr][blkIdx4x4][i] = ChromaACLevel[iCbCr][blkIdx4x4][i] < 0 ? SigMag1 : SigMag0		
mag2_remain--		
}		
mag1_remain--		
}		
mag2_remain = mag2_count		
seen_max = 0		
for(i = 1; i < 16 && mag2_remain > 0; i++)		
if(ChromaACLevel[iCbCr][blkIdx4x4][i] = = SigMag1 | | LumaLevel[blkIdx4x4][i] = = SigMag0) {		
if(seen_max | | mag2_remain > 1) {		
coef_mag = 2		
for(j = 1; j < max_mag; j++) {		
coef_mag_ref_bit	3 | 4	u(1)
if(coef_mag_ref_bit)		
coef_mag++		
else		
j = max_mag		
}		
} else		
coef_mag = max_mag		
ChromaACLevel[iCbCr][blkIdx4x4][i] = (ChromaACLevel[iCbCr][blkIdx4x4][i] = = SigMag1) ? –coef_mag : coef_mag		
mag2_remain--		
seen_max = (coef_mag = = max_mag) ? 1 : seen_max		
}		
}		
}		
}		
}		

G.7.3.7.7 Significant coefficient in progressive slice data CABAC syntax

significant_coefficient_cabac (firstCoeff, numCoef, trmCoef) {	C	Descriptor
numCoef = 0		
if(! firstCoeff && more_cabac_data_in_nal_unit( ))		
last_significant_coeff_flag	3 | 4	ae(v)
while(! last_significant_coeff_flag && ! significant_coeff_flag) {		
if(more_cabac_data_in_nal_unit( )) {		
significant_coeff_flag	3 | 4	ae(v)
numCoef++		
}		
if(significant_coeff_flag) {		
if(more_cabac_data_in_nal_unit( ))		
coeff_sign_flag	3 | 4	ae(v)
if(more_cabac_data_in_nal_unit( )) {		
coeff_abs_level_minus1	3 | 4	ae(v)
trmCoef = (coeff_abs_level_minus1 + 1) * (1 – 2 * coeff_sign_flag)		
}		
}		
}		

G.7.3.7.8 Significant coefficient in progressive slice data CAVLC syntax

significant_coefficient_cavlc (firstCoeff, numRemain, cutoffSym, eobShift, numCoef, termCoef) {	C	Descriptor
coeff_sig_vlc_symbol	3 | 4	ge(v)
eob = 0		
if(! firstCoeff)		
eob = (coef_sig_vlc_symbol > numRemain) | | (coef_sig_vlc_symbol = = eobShift)		
if(! eob) {		
numCoef = coef_sig_vlc_symbol + (firstCoeff | | coef_sig_vlc_symbol < = eobShift)		
coeff_sign_flag	3 | 4	u(1)
termCoef = 1 – 2 * coeff_sign_flag		
} else		
termCoef = coef_sig_vlc_symbol - numRemain		

G.7.3.7.9 Luma coefficient refinement in progressive refinement slice data syntax

luma_coefficient_ref(blkIdx4x4, scanIdx) {	C	Descriptor
i8x8 = blkIdx4x4 / 4		
i4x4 = blkIdx4x4 % 4		
scanIdx8x8 = 4 * scanIdx + i4x4		
if((! TransformSize8x8Flag && base_luma_level(CurrMbAddr, i8x8, i4x4, scanIdx) != 0) || (TransformSize8x8Flag && base_luma_level8x8(CurrMbAddr, i8x8, scanIdx8x8) != 0)) {		
coeffLevel = 0		
if(! TransformSize8x8Flag) {		
bLSign = (base_luma_level(CurrMbAddr, i8x8, i4x4, scanIdx) < 0 ? 1 : 0)		
coeff_refinement(bLSign, LumaLevel[blkIdx4x4][scanIdx])	3 | 4	
} else {		
bLSign = (base_luma_level8x8(CurrMbAddr, i8x8, scanIdx8x8) < 0 ? 1 : 0)		
coeff_refinement(bLSign, LumaLevel8x8[i8x8][scanIdx8x8])	3 | 4	
}		
}		
}		

G.7.3.7.10 Chroma DC coefficient refinement in progressive refinement slice data syntax

chromaDC_coefficient_ref(iCbCr, dcIdx) {	C	Descriptor
if(base_chromaDC_level(CurrMbAddr, iCbCr, dcIdx)) {		
bLSign = base_chromaDC_level(CurrMbAddr, iCbCr, dcIdx) < 0 ? 1 : 0		
coeff_refinement(bLSign, ChromaDCLevel[iCbCr][dcIdx])	3 | 4	
}		
}		

G.7.3.7.11 Chroma AC coefficient refinement in progressive refinement slice data syntax

chromaAC_coefficient_ref(iCbCr, blkIdx4x4, scanIdx) {	C	Descriptor
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
i8x8 = blkIdx4x4 / 4		
i4x4 = blkIdx4x4 % 4		
if(base_chromaAC_level(CurrMbAddr, iCbCr, i8x8, i4x4, scanIdx) != 0) {		
bLSign = (base_chromaAC_level(CurrMbAddr, iCbCr, i8x8, i4x4, scanIdx) < 0 ? 1 : 0)		
coeff_refinement(bLSign, ChromaACLevel[iCbCr][blkIdx4x4][dcIdx])	3 | 4	
}		
}		

G.7.3.7.12 Coefficient refinement in progressive refinement slice data syntax

coeff_refinement(bLSign, coeff_level) {		
if((entropy_coding_mode = = 1 && more_cabac_data_in_nal_unit( )) || (entropy_coding_mode = = 0 && more_rbsp_data( ))) {		
coeff_refinement_flag	3 | 4	ae(v) | ge(v)
if(coeff_refinement_flag)		
if(entropy_coding_mode = = 1 && more_cabac_data_in_nal_unit( ))		
coeff_refinement_direction_flag	3 | 4	ae(v) | ge(v)
}		
coeffLevel = 0		
if(coeff_refinement_flag)		
coeffLevel = ((coeff_refinement_direction_flag ^ bLSign) ? –1 : 1)		
}		

G.7.4 Semantics

The specifications in subclause 7.4 shall apply. Additionally, the following semantics are specified.

G.7.4.1 NAL unit header SVC extension semantics

When nal_unit_type is equal to 20 or 21 for a NAL unit containing a slice of a coded picture, the NAL unit shall not have dependency_id and quality_level both equal to 0.

When the value of nal_unit_type is equal to 20 or 21 for a NAL unit that does not contain a slice of a coded picture, the NAL unit is a suffix NAL unit that shall have dependency_id and quality_level both equal to 0. A suffix NAL unit with nal_unit_type equal to 20 shall be associated with a NAL unit with nal_unit_type equal to 1 and shall be the next NAL unit after the associated NAL unit in decoding order. A suffix NAL unit with nal_unit_type equal to 21 shall be associated with a NAL unit with NAL unit type equal to 5 and shall be the next NAL unit after the associated NAL unit in decoding order.

When nal_unit_type is equal to 21 for a NAL unit containing a slice of a scalable layer, the value of nal_unit_type shall be equal to 21 in all other VCL NAL units of the same scalable layer. Such a scalable layer is referred to as an IDR scalable layer.

Let the dependency_id of an IDR scalable layer be equal to idrDid. Let the target NAL units be those NAL units that have dependency_id less or equal to idrDid, and that are in the access unit containing the IDR scalable layer or in a subsequent access unit in decoding order of the access unit containing the IDR scalable layer. The decoding of any of

the target NAL units shall be independent of any NAL unit that is in an access unit preceding the access unit containing the IDR scalable layer in decoding order.

When nal_unit_type is equal to 5 for any NAL unit in an access unit, the access unit shall not contain any NAL unit with nal_unit_type equal to 1-4 or 20. An access unit, which contains a NAL unit with nal_unit_type equal to 5 is referred to as IDR access unit.

reserved_zero_two_bits shall be equal to 0. The values of reserved_zero_two_bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_two_bits.

priority_id specifies a priority identifier for the NAL unit. When the NAL unit is a suffix NAL unit, the priority identifier also apply to the associated NAL unit, which is the previous NAL unit in decoding order.

Let pId be any value larger or equal to Pmin, where Pmin is the mininum value of priority_id in the coded video sequence. The bitstream that would be obtained by discarding all NAL units with priority_id greater than pId shall be conforming to this Recommendation | International Standard

NOTE – The syntax element priority_id is not used by the decoding process specified in this Recommendation | International Standard; The syntax element priority_id may be used as determined by the application.

reserved_zero_one_bit shall be equal to 0. The values of reserved_zero_one_bit may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_zero_one_bit.

temporal_level specifies a temporal level for the NAL unit. temporal_level, if present, shall be identical for all the NAL units in an access unit. When temporal_level is not present for a NAL unit, the value of temporal_level for that NAL unit shall be inferred to be equal to the value of temporal_level of the associated suffix NAL unit.

The value of temporal_level shall be equal to 0 for an access unit containing any NAL unit with nal_unit_type equal to 5 or 21.

The decoding of any access unit with temporal_level equal to currTl shall be independent of all access units with temporal_level greater than currTl.Let tId be any value larger or equal to Tmin, where Tmin is the mininum value of temporal_level in the coded video sequence. The bitstream that would be obtained by discarding all NAL units with temporal_level greater than tId shall be conforming to this Recommendation | International Standard

dependency_id specifies a dependency identifier for the NAL unit. When dependency_id is not present, it shall be inferred to be equal to 0. The dependency_id is used in the decoding process (subclause G.8) and for identifying base pictures that are used for inter-layer prediction of motion and/or texture data. The value of dependency_id shall be equal to 0 for a suffix NAL unit.

Let dId be any value larger or equal to Dmin, where Dmin is the mininum value of dependency_id in the coded video sequence. The bitstream that would be obtained by discarding all NAL units with dependency _id greater than dId shall be conforming to this Recommendation | International Standard

quality_level specifies a quality level for the current NAL unit. When quality_level is not present, it shall be inferred to be equal to 0. The quality_level is used in connection with the variable PRsliceCompletelyDecoded of previous NAL units in decoding order for determining whether a NAL unit containing a PR slice can be decoded. The value of quality_level shall be equal to 0 for a suffix NAL unit.**layer_base_flag** equal to 1 specifies that no inter-layer prediction (of coding mode, motion, sample value, and/or residual prediction) is used for the current slice. layer_base_flag equal to 0 specifies that an inter-layer prediction (of coding mode, motion, sample value, and/or residual prediction) may be used for the current slice when signalled in the macroblock layer.

use_base_prediction_flag equal to 1 specifies that the base representation of the reference pictures is used during the inter prediction proces.

discardable_flag equal to 1 specifies that the content of the NAL unit (currDependencyId = dependency_id) is not used in the decoding process of NAL units with dependency_id > currDependencyId. discardable_flag equal to 0 indicates that the content of the NAL unit (currDependencyId = dependency_id) is used in the decoding process of NAL units with dependency_id > currDependencyId. When the NAL unit is a suffix NAL unit, the semantics of discardable_flag also apply to the associated NAL unit, which is the previous NAL unit in decoding order.

Let enhDId and bId be the values of the syntax elements dependency_id and base_id, respectively, for any NAL unit with quality_level equal to 0 inside the current access unit. For any of these NAL units with enhDId greater than dependency_id and bId greater or equal to 0, the following applies.

– The variables DependencyIdBase, QualityLevelBase, and FragmentOrderBase are derived as follows.

  – DependencyIdBase = bId >> 4

  - QualityLevelBase = (bId >> 2) & 3.
  - FragmentOrderBase = bId & 3
- When DependencyIdBase is equal to dependency_id, the following applies,
  - When quality_level is smaller than QualityLevelBase, or when quality_level is equal to QualityLevelBase and fragment_order not greater than FragmentOrderBase, discardable_flag shall be equal to 0.
  - When quality_level is greater than QualityLevelBase, or if quality_level is equal to QualityLevelBase and fragment_order greater than FragmentOrderBase, discardable_flag shall be equal to 1.

When discardable_flag is equal to 1, the NAL unit shall not be referenced by the syntax element base_id of any other NAL unit of the same access unit.

fragmented_flag equal to 1 specifies that the current NAL unit is fragmented. fragmented_flag equal to 0 specifies that the current NAL unit is not fragmented. If fragmented_flag is not present it shall be inferred to be equal to 0.

When fragmented_flag is equal to 1, the NAL unit cannot be parsed independently; in this case the RBSP bytes of the slice data of all NAL units with identical values of first_mb_in_slice, slice_type and fragmented_flag shall be stored in a temporary RBSP byte buffer in increasing order of fragment order. The parsing process is started when a NAL unit with last_fragment_flag equal to 1 is received or when the next NAL unit is not a NAL unit with identical values of first_mb_in_slice, slice_type and fragmented_flag, or when the next NAL unit belongs to another access unit.

last_fragment_flag equal to 1 specifies that the current NAL unit is the last fragment of a progressive refinement slice and that the parsing process can be started. last_fragment_flag equal to 0 specifies that zero or more NAL units containing fraction of the current progressive refinement slice may follow.

If not present last_fragment_flag shall be inferred to be equal to 0 for a fragmented progressive refinement slice with fragment_order equal to 0 and shall be inferred to be equal to 1 otherwise.

fragment_order specifies the order in which the NAL units with fragmented_flag equal to 1 shall be ordered before the parsing process is started. If fragment_order is not present it should be inferred to be equal to 0.

A NAL unit with fragmentOrder = fragment_order greater than 0 shall immediately follow a NAL unit with identical values of first_mb_in_slice, slice_type and fragmented_flag, and a value of fragment_order equal to fragmentOrder – 1.

G.7.4.1.1 Order of VCL NAL units and association to coded pictures in the scalable extension

The order of the VCL NAL units within a coded picture in the scalable extension is constrained as follows.

- A NAL unit with dependency_id equal to dependencyId shall not precede any NAL unit with dependency_id less than dependencyId.
- A NAL unit with dependency_id equal to dependencyId and quality_level equal to qualityLevel shall not precede any NAL unit with dependency_id equal to dependencyId and quality_level less than qualityLevel.
- A NAL unit with dependency_id equal to dependencyId, quality_level equal to qualityLevel, first_mb_in_slice equal to firstMbInSlice, and fragment_order equal to fragmentOrder shall directly precede the NAL unit with dependency_id equal to dependencyId, quality_level equal to qualityLevel, first_mb_in_slice equal to firstMbInSlice, and fragmentOrder equal to fragmentOrder + 1 (when present).

G.7.4.1.2 Detection of first VCL NAL units of a primary coded picture / access unit

[Ed. (HS): We need to add a process similar to 7.4.1.2.4 of the main body, but with considering the structure of SVC access units.

First ideas (to be refined, worked out):

- dependency_id of the current coded slice NAL unit is less than dependency_id of the previous coded slice NAL unit in the bit-stream
- dependency_id of the current coded slice NAL unit is equal to dependency_id of the previous coded slice NAL unit in the bit-stream and any of the following conditions is true
  - quality_level of the current coded slice NAL unit is less than quality_level of the previous coded slice NAL unit in the bit-stream

– quality_level of the current coded slice NAL units is equal to 0, quality_level of the previous coded slice NAL unit in the bit-stream is equal to 0, and any of the conditions specified in subclause 7.4.1.2.4 is fulfilled.

]

G.7.4.2 Sequence parameter set SVC extension semantics

interlayer_deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter for interlayer prediction is present in the slice header. deblocking_filter_control_present_flag equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter for interlayer prediction is not present in the slice headers and their inferred values are in effect.

extended_spatial_scalability specifies the presence of syntax elements related to geometrical parameters for the base layer upsampling. When extended_spatial_scalability is equal to 0, no geometrical parameter is present in the bitstream. When extended_spatial_scalability is equal to 1, geometrical parameters are present in the sequence parameter set. When extended_spatial_scalability is equal to 2, geometrical parameters are present in slice_data_in_scalable_extension. The value of 3 is reserved for extended_spatial_scalability. When extended_spatial_scalability is not present, it shall be inferred to be equal to 0.

scaled_base_left_offset specifies the horizontal offset between the upper-left pixel of an upsampled base layer picture and the upper-left pixel of a picture of the current layer in units of two luma samples. When scaled_base_left_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseLeftOffset is defined as follow:

ScaledBaseLeftOffset = 2 * scaled_base_left_offset (G-49)

The variable ScaledBaseLeftOffsetC is defined as follow:

ScaledBaseLeftOffsetC = ScaledBaseLeftOffset / SubWidthC (G-50)

scaled_base_top_offset specifies vertical offset of the upper-left pixel of an upsampled base layer picture and the upper-left pixel of a picture of the current layer in units of two luma samples. When frame_mbs_only_flag is 0, scaled_base_top_offset represents the offset in a field picture. When scaled_base_top_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseTopOffset is defined as follow:

ScaledBaseTopOffset = 2 * scaled_base_top_offset * (2 – frame_mbs_only_flag) (G-51)

The variable ScaledBaseTopOffsetC is defined as follow:

ScaledBaseTopOffsetC = ScaledBaseTopOffset / SubHeightC (F-52)

scaled_base_right_offset specifies the horizontal offset between the bottom-right pixel of an upsampled based layer picture and the bottom-right pixel of a picture of the current layer in units of two luma samples. When scaled_base_right_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseRightOffset is defined as follow:

ScaledBaseRightOffset = 2 * scaled_base_right_offset (G-53)

The variable ScaledBaseWidth is defined as follow:

ScaledBaseWidth= PicWidthInMbs * 16 – ScaledBaseLeftOffset – ScaledBaseRightOffset (G-54)

The variable ScaledBaseWidthC is defined as follow:

ScaledBaseWidthC = ScaledBaseWidth / SubWidthC (G-55)

scaled_base_bottom_offset specifies the vertical offset between the bottom-right pixel of an upsampled based layer picture and the bottom-right pixel of a picture of the current layer in units of two luma samples. When frame_mbs_only_flag is 0, scaled_base_bottom_offset represents the offset in a field picture. When scaled_base_bottom_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseBottomOffset is defined as follow:

ScaledBaseBottomOffset = 2 * scaled_base_bottom_offset * (2 – frame_mbs_only_flag) (G-56)

The variable ScaledBaseHeight is defined as follow:

ScaledBaseHeight = PicHeightInMbs * 16 –
(ScaledBaseTopOffset + ScaledBaseBottomOffset) / (1 + field_pic_flag) (G-57)

The variable ScaledBaseHeightC is defined as follow:

ScaledBaseHeightC = ScaledBaseHeight / SubHeightC (G-58)

NOTE – When extended_spatial_scalability is equal to 2, these geometrical parameters are present in the slice data in scalable extension semantics.

NOTE – The values of scaled_base_bottom_offset and scaled_base_top_offset shall both be even when base_frame_mbs_only_flag is equal to 0.

chroma_phase_x_plus1 specifies the horizontal phase shift of the chroma components in units of quarter sampling space in the horizontal direction of a picture of the current layer. When chroma_phase_x_plus1 is not present, it shall be inferred to be equal to 0. The chroma_phase_x_plus1 is in range 0..1, the values of 2 and 3 are reserved. [Ed. Note (SSun) the reserved bit shall be removed later.]

chroma_phase_y_plus1 specifies the vertical phase shift of the chroma components in units of quarter sampling space in the vertical direction of a picture of the current layer. When chroma_phase_y_plus1 is not present, it shall be inferred to be equal to 1. The chroma_phase_y_plus1 is in range 0..2, the value of 3 is reserved.

NOTE – The chroma type specified in the vui_parameters should be consistent with the chroma phase parameters chroma_phase_x_plus1 and chroma_phase_y_plus1 in the same sequence_parameter_set.

fgs_coding_mode specifies the vector scanPosVectLuma containing the scanning positions to reach at each coding cycle of a progressive refinement slice for the luminance blocks and the AC blocks of each chrominance. A value of 0 specifies that the vector is inferred using the syntax element goupingSizeMinus1. A value of 1 specifies that the vector is inferred using the syntax elements scanIndex0Minus1 and deltaScanIndexMinus1[numPosVect] The vector scanPosVectChromaDC for the DC block of each chrominance is inferred by [0, 1, 2, 3] for any value of fgs_coding_mode.

groupingSizeMinus1 specifies the number of scanning positions grouped for each cycle, minus 1. For the cycle i , the element scanPosVectLuma[i] of the vector is then equal to scanPosVectLuma[i – 1] + groupingSizeMinus1 + 1 and scanPosVectLuma[0] equals groupingSizeMinus1. When groupingSizeMinus1 is not present it shall be inferred to be equal to 1.

scanIndex0Minus1 specifies the scanning position to reach for the first cycle when fgs_coding_mode is equal to 1. The first element scanPosVectLuma[0] of the vector is then equal to scanIndex0Minus1 + 1.

deltaScanIndexMinus1 specifies the difference between the position to reach at cycle i – 1 and the position at cycle i, minus 1. For the cycle i, the element scanPosVectLuma[i] of the vector is then equal to scanPosVectLuma[i – 1] + deltaScanIndexMinus1[i – 1] + 1.

The scanning index are given for 4x4 blocks, scanning index for 8x8 blocks are derived by scanPosVectLuma[i] * 4 + 3.

G.7.4.3 Slice layer in scalable extension semantics

store_base_rep_flag equal to 1 specifies that an additional representation of the current picture has to be stored for the prediction of future picture. store_base_rep_flag equal to 0 specifies that no special processing is done on the current picture. When this syntax element is not present, it shall be inferred to be equal to 0.

cabac_alignment_one_bit is a bit equal to 1. [Ed. (JR) is it necessary to repeat a syntax element description when it appears twice ?]

rbtmp_byte[i **]** is the i-th byte of a progressive_refinement_slice_data_in_scalable_extension( ) payload, starting from the end of the slice header. The rbtmp_byte[ ] is used to append sub-parts of the slice data of a progressive refinement slice before parsing its syntax elements

G.7.4.4 Slice header in scalable extension semantics

The semantic specified in subclauses 7.4.3 shall apply. Additionally, the following semantics are specified.

When present, the value of the slice header in scalable extension syntax elements pic_parameter_set_id, frame_num, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], base_id, and slice_group_change_cycle, base_chroma_phase_x_plus1, base_chroma_phase_y_plus1, scaled_base_left_offset, scaled_base_top_offset, scaled_base_right_offset, scaled_base_bottom_offset shall be identical for all slice data NAL units in a scalable layer.

slice_type specifies the coding type of the slice according to Table G-3.

Table G-3 – Name association to slice_type for NAL units with nal_unit_type equal to 20 or 21.

slice_type	Name of slice_type
0	EP (P slice in scalable extension)
1	EB (B slice in scalable extension)
2	EI (I slice in scalable extension)
3	PR (progressive refinement slice)

When nal_unit_type is equal to 21, slice_type shall be equal to 2 or 3.

When num_ref_frames is equal to 0, slice_type shall be equal to 2 or 3.

When fragmented_flag is equal to 1, slice_type shall be equal to 3.

When slice_type is equal to 3, the following applies.

– Let dId be the value of dependency_id, and let setOfBaseSlices be the set of slices that is represented by the NAL units with dependency_id equal to dId and quality_level equal to 0 inside the current access unit.

– When the current slice covers more than one slice in setOfBaseSlices, the following constraints shall be fulfilled for all of the covered slices in setOfBaseSlices.

  – The values of slice_type, num_ref_l0_active_minus1 (when present), and num_ref_l1_active_minus1 (when present) shall be identical for all covered slices.

  – Either the syntax structure ref_pic_list_reordering( ) shall not be present in all covered slices, or all syntax elements inside the syntax structure ref_pic_list_reordering( ) shall be identical for all covered slices.

  – Either the syntax structure pred_weight_table( ) shall not be present in all covered slices, or all syntax elements inside the syntax structure pred_weight_table( ) shall be identical for all covered slices.

– A variable BaseSliceType is set equal to slice_type of the slices in setOfBaseSlices that are covered by the current slice.

– the syntax element fragmented_flag shall be equal to 1

base_id specifies the value of dependency_id, quality_level and fragment_order for base pictures that are used for inter-layer prediction of coding mode, motion, samples values, and/or residual values of the current slice. When this syntax element is not present, it shall be inferred to be equal to -1.

If this syntax element is present, the variables DependencyIdBase, QualityLevelBase, and FragmentOrderBase are derived as follows.

– DependencyIdBase = base_id >> 4

– QualityLevelBase = (base_id >> 2) & 3.

– FragmentOrderBase = base_id & 3

Otherwise (the syntax element base_id is not present), DependencyIdBase , QualityLevelBase, and FragmentOrderBase are set equal to –1.

If base_id is greater or equal to 0, for any other coded slice NAL units in the access unit with the same value of DependencyIdBase derived from the base_id syntax element in the slice header, the value of base_id shall be equal to the value of base_id of the current NAL unit.

If base_id is greater or equal to 0, the variables BasePicWidthInMbs, BaseChromaFormatIdc, BasePicWidth, BasePicHeight, BasePicWidthC, BasePicHeightC, BaseMbWidthC, and BaseMbHeightC are defined as follows.

– BasePicWidthInMbs is equal to basePicWidthInMbsMinus1 + 1, with basePicWidthInMbsMinus1 being the syntax element pic_width_in_mbs_minus1 of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase

– BaseChromaFormatIdc is equal to the syntax element chroma_format_idc of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BasePicWidth is equal to variable PicWidthInSamplesL of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BasePicHeight is equal to variable PicHeightInSamplesL of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BasePicWidthC is equal to variable PicWidthInSamplesC of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BasePicHeightC is equal to variable PicHeightInSamplesC of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BaseMbWidthC is equal to variable MbWidthC of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BaseMbHeightC is equal to variable MbHeightC of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BaseFrameMbsOnlyFlag is equal to frame_mbs_only_flag of the active sequence parameter set for the pictures with dependency_id equal to DependencyIdBase.

– BaseFieldPicFlag is equal to field_pic_flag of the picture with dependency_id equal to DependencyIdBase in the current access unit.

– BaseBottomFieldFlag is equal to bottom_field_flag of the picture with dependency_id equal to DependencyIdBase in the current access unit.

– BaseMbaffFrameFlag is equal to MbaffFrameFlag of the picture with dependency_id equal to DependencyIdBase in the current access unit.

adaptive_prediction_flag specifies the presence of syntax elements in the macroblock layer in scalable extension. When this syntax element is not present, it shall be inferred to be equal to 0.

base_pred_weight_table_flag equal to 1 specifies that the variables for weighted prediction are inferred. When base_pred_weight_table_flag is not present, it shall be inferred as follows.

– If layer_base_flag is equal to 1, base_pred_weight_table_flag shall be inferred to be equal to 0.

– Otherwise (layer_base_flag is equal to 0), base_pred_weight_table_flag shall be inferred to be equal to 1.

When base_pred_weight_table_flag equal to 1, the following applies.

– Let dId be the value of dependency_id, and let setOfBaseSlices be the set of slices that is represented by the NAL units with dependency_id equal to dId and quality_level equal to 0 inside the current access unit.

– When the current slice covers more than one slice in setOfBaseSlices, the following constraints shall be fulfilled for all of the covered slices in setOfBaseSlices.

  – The values of slice_type, num_ref_l0_active_minus1 (when present), and num_ref_l1_active_minus1 (when present) shall be identical for all covered slices.

  – Either the syntax structure ref_pic_list_reordering( ) shall not be present in all covered slices, or all syntax elements inside the syntax structure ref_pic_list_reordering( ) shall be identical for all covered slices.

- Either the syntax structure pred_weight_table( ) shall not be present in all covered slices, or all syntax elements inside the syntax structure pred_weight_table( ) shall be identical for all covered slices.

– A variable BaseSliceType is set equal to slice_type of the slices in setOfBaseSlices.

disable_interlayer_deblocking_filter_idc specifies whether the operation of the deblocking filter for interlayer prediction shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When disable_interlayer_deblocking_filter_idc is not present in the slice header, the value of disable_interlayer_deblocking_filter_idc shall be inferred to be equal to 0.

The value of disable_interlayer_deblocking_filter_idc shall be in the range of 0 to 2, inclusive.

interlayer_slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{C0} deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as:

InterlayerFilterOffsetA = interlayer_slice_alpha_c0_offset_div2 << 1 (G-59)

The value of interlayer_slice_alpha_c0_offset_div2 shall be in the range of -6 to +6, inclusive. When interlayer_slice_alpha_c0_offset_div2 is not present in the slice header, the value of interlayer_slice_alpha_c0_offset_div2 shall be inferred to be equal to 0.

interlayer_slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations controlled by the macroblocks within the slice. From this value, the offset that is applied when addressing the β table of the deblocking filter shall be computed as:

InterlayerFilterOffsetB = interlayer_slice_beta_offset_div2 << 1 (G-60)

The value of interlayer_slice_beta_offset_div2 shall be in the range of -6 to +6, inclusive. When interlayer_slice_beta_offset_div2 is not present in the slice header the value of interlayer_slice_beta_offset_div2 shall be inferred to be equal to 0.

base_chroma_phase_x_plus1 specifies the horizontal phase shift of the chroma components in units of quarter sampling space in the horizontal direction of the pictures with dependency_id equal to DependencyIdBase. When base_chroma_phase_x_plus1 is not present, it shall be inferred to be equal to chroma_phase_x_plus1.

base_chroma_phase_y_plus1 specifies the vertical phase shift of the chroma components in units of quarter sampling space in the vertical direction of the pictures with dependency_id equal to DependencyIdBase. When base_chroma_phase_y_plus1 is not present, it shall be inferred to be equal to chroma_phase_y_plus1.

scaled_base_left_offset specifies the horizontal offset between the upper-left pixel of the upsampled base layer picture and the upper-left pixel of the current picture in units of two luma samples of current picture. When scaled_base_left_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseLeftOffset is defined as follow:

ScaledBaseLeftOffset = 2 * scaled_base_left_offset (G-61)

The variable ScaledBaseLeftOffsetC is defined as follow:

ScaledBaseLeftOffsetC = ScaledBaseLeftOffset / SubWidthC (G-62)

scaled_base_top_offset specifies vertical offset of the upper-left pixel of the upsampled base layer picture and the upper-left pixel of the current picture in units of two luma samples of current picture. When frame_mbs_only_flag is 0, scaled_base_top_offset represents the offset in a field picture. When scaled_base_top_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseTopOffset is defined as follow:

ScaledBaseTopOffset = 2 * scaled_base_top_offset * (2 – frame_mbs_only_flag) (G-63)

The variable ScaledBaseTopOffsetC is defined as follow:

ScaledBaseTopOffsetC = ScaledBaseTopOffset / SubHeightC (F-64)

scaled_base_right_offset specifies the horizontal offset between the bottom-right pixel of the upsampled based layer picture and the bottom-right pixel of the current picture in units of two luma samples of current picture. When scaled_base_right_offset is not present, it shall be inferred to be equal to 0.

The variable ScaledBaseRightOffset is defined as follow:

ScaledBaseRightOffset = 2 * scaled_base_right_offset (G-65)

The variable ScaledBaseWidth is defined as follow:

ScaledBaseWidth= PicWidthInMbs * 16 – ScaledBaseLeftOffset – ScaledBaseRightOffset (G-66)

The variable ScaledBaseWidthC is defined as follow:

ScaledBaseWidthC = ScaledBaseWidth / SubWidthC (G-67)

scaled_base_bottom_offset specifies the vertical offset between the bottom-right pixel of the upsampled based layer picture and the bottom-right pixel of the current picture in units of two luma samples of current picture. When frame_mbs_only_flag is 0, scaled_base_bottom_offset represents the offset in a field picture. When scaled_base_bottom_offset is not present, it shall be inferred to be equal to 0.

NOTE – These geometrical parameters, if present in the slice data in scalable extension semantics, shall apply to the current slice, and the parameters shall be consistent for slices of the same layer within an access unit. These parameters shall be associated to the current picture if the picture is to be used as reference picture

NOTE – The values of scaled_base_bottom_offset and scaled_base_top_offset shall both be even when base_frame_mbs_only_flag is equal to 0.

The variable ScaledBaseBottomOffset is defined as follow:

ScaledBaseBottomOffset = 2 * scaled_base_bottom_offset * (2 – frame_mbs_only_flag) (G-68)

The variable ScaledBaseHeight is defined as follow:

ScaledBaseHeight = PicHeightInMbs * 16
(ScaledBaseTopOffset + ScaledBaseBottomOffset) / (1 + field_pic_flag) (G-69)

The variable ScaledBaseHeightC is defined as follow:

ScaledBaseHeightC = ScaledBaseHeight / SubHeightC (G-70)

base_frame_and_bottom_field_coincided_flag equal to 0 indicates the frame picture with dependency_id equal to DependencyIdBase is coincided with a top field picture. base_frame_and_bottom_field_coincided_flag equal to 1 indicates the frame picture with dependency_id equal to DependencyIdBase is coincided with a bottom field picture. When base_frame_and_bottom_field_coincided_flag is not present, it shall be inferred to be equal to 0. [Shijun: some modification is needed if independent parsing is enforced. Same to the next one.]

base_bottom_field_coincided_flag equal to 0 indicates the top field of the frame picture with dependency_id equal to DependencyIdBase is coincided with the current frame picture. base_bottom_field_coincided_flag equal to 1 indicates the bottom field of the frame picture with dependency_id equal to DependencyIdBase is coincided with the current frame picture. When base_bottom_field_coincided_flag is not present, it shall be inferred to be equal to 0.

num_mbs_in_slice_minus1 plus 1 specifies the number of macroblocks in the progressive refinement slice.

luma_chroma_sep_flag equal to 0 specifies that any chroma transform coefficient levels to be decoded for the current macroblock will immediately follow the luma transform coefficient levels. luma_chroma_sep_flag equal to 1 specifies that all luma transform coefficient levels for the slice are to be decoded first, followed by all chroma transform coefficient values.

store_base_rep_flag equal to 1 specifies that an additional representation of the current picture has to be stored for the prediction of future picture. store_base_rep_flag equal to 0 specifies that no special processing is done on the current picture. When this syntax element is not present, it shall be inferred to be equal to 0.

adaptive_ref_fgs_flag equal to 1 specifies that adaptive reference is used in decoding the progressive slice of a key picture. When key_pic_flag is equal to 0, adaptive_ref_fgs_flag shall be equal to 0.

max_diff_ref_scale_for_zero_base_block specifies the maximum scaling factor to be used for scaling the differential reference signal in constructing the Inter prediction samples used in decoding the progressive slice of a key picture, when the transform block in the base layer does not have any nonzero coefficients. The value of max_diff_ref_scale_for_zero_base_block shall be in the range of 0 to 31, inclusive.

A variable MaxDiffRefScaleZeroBaseBlock is derived as follows.

– If max_diff_ref_scale_for_zero_base_block is equal to 0, the variable MaxDiffRefScaleZeroBaseBlock is set equal to 0.

– Otherwise (max_diff_ref_scale_for_zero_base_block is not equal to 0), the variable MaxDiffRefScaleZeroBaseBlock is set equal to (max_diff_ref_scale_for_zero_base_block + 1).

max_diff_ref_scale_for_zero_base_coeff specifies the maximum scaling factor to be used for scaling the differential reference signal in constructing the Inter prediction samples used in decoding the progressive slice of a key picture, when the number of nonzero coefficients in transform block in the base layer is larger than 0, but the corresponding transform coefficient level is equal to 0.

A variable MaxDiffRefScaleZeroBaseCoeff is derived as follows.

– If max_diff_ref_scale_for_zero_base_coeff is equal to 0, the variable MaxDiffRefScaleZeroBaseCoeff is set equal to 0.

– Otherwise (max_diff_ref_scale_for_zero_base_coeff is not equal to 0), the variable MaxDiffRefScaleZeroBaseCoeff is set equal to (max_diff_ref_scale_for_zero_base_coeff + 1).

motion_refinement_flag equal to 1 specifies that in the current progressive refinement slice for each inter-coded macroblock additional syntax elements are transmitted which allow to specify a new macroblock type and new motion vectors. motion_refinement_flag equal to 0 specifies that the base layer motion data are to be used for motion-compensated prediction. When motion_refinement_flag is equal to 1, constrained_intra_pred_flag shall to be equal to 1.

G.7.4.4.1 Semantics of decoded reference picture marking syntax for decoded base pictures

The syntax elements base_adaptive_ref_pic_marking_mode_flag, memory_management_control_operation, difference_of_pic_nums_minus1 and long_term_pic_num specify marking of reference pictures as “unused for reference” for the decoded base pictures.

base_adaptive_ref_pic_marking_mode_flag has the same semantics as the syntax element adaptive_ref_pic_marking_mode_flag in the dec_ref_pic_marking() syntax structure.

memory_management_control_operation specifies a control operation to be applied to affect the reference picture marking. The memory_management_control_operation syntax element is followed by data necessary for the operation specified by the value of memory_management_control_operation. The values and control operations associated with memory_management_control_operation are specified in Table G-4. The memory_management_control_operation syntax elements are processed by the decoding process in the order in which they appear, and the semantics constraints expressed for each memory_management_control_operation apply at the specific position in that order at which that individual memory_management_control_operation is processed.

memory_management_control_operation shall not be equal to 1 unless the specified reference picture is marked as "used for short-term reference" and as “base representation” when the memory_management_control_operation is processed by the decoding process. The specified reference picture may be absent.

memory_management_control_operation shall not be equal to 2 unless the specified long-term picture number refers to a reference picture that is marked as "used for long-term reference" and as “base representation” when the memory_management_control_operation is processed by the decoding process. The specified long-term reference picture may be absent.

Table G-4 – Memory management control operation (memory_management_control_operation) values

memory_management_control_operation	Memory Management Control Operation
0	End memory_management_control_operation syntax element loop
1	Mark a short-term decoded base picture as “unused for reference”
2	Mark a long-term decoded base picture as “unused for reference”

difference_of_pic_nums_minus1 is used (with memory_management_control_operation equal to 1) to mark a short-term reference picture as “unused for reference”. When the associated memory_management_control_operation is processed by the decoding process, the resulting picture number derived from difference_of_pic_nums_minus1 shall be a picture number assigned to one of the reference pictures marked as "used for reference" and as “base representation”. The specified reference picture may be absent.

The resulting picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs, or there is no reference frame or complementary reference field pair that has been assigned the resulting picture number.

– Otherwise (field_pic_flag is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields, or there is no reference pair that has has been assigned the resulting picture number.

long_term_pic_num is used (with memory_management_control_operation equal to 2) to mark a long-term reference picture as "unused for reference". When the associated memory_management_control_operation is processed by the decoding process, long_term_pic_num shall be equal to a long-term picture number assigned to one of the reference pictures that is currently marked as "used for long-term reference" and as “base representation”. The specified long-term reference picture may be absent.

The resulting long-term picture number is constrained as follows.

– If field_pic_flag is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs, or there is no reference frame or complementary reference field pair that has been assigned the resulting long-term picture number..

– Otherwise (field_pic_flag is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields, or there is no reference pair that has has been assigned the resulting long-term picture number.

G.7.4.5 Slice data in scalable extension semantics

The semantic specified in subclauses 7.4.4 shall apply. Additionally the sementics for mb_skip_run and mb_skip_flag are modified.

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P or EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as a B macroblock type. The value of mb_skip_run shall be in the range of 0 to PicSizeInMbs – CurrMbAddr, inclusive.

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or EP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding a B slice, mb_type shall be inferred to be B_Skip and the macroblock type is collectively referred to as B macroblock type. mb_skip_flag equal to 0 specifies that the current macroblock is not skipped.

G.7.4.6 Macroblock layer in scalable extension semantics

The semantic specified in subclauses 7.4.5 shall apply. Additionally, the following semantics are specified.

base_mode_flag equal to 1 specifies that mb_type for the current macroblock is inferred. The reference indices and motion vectors shall be inferred in dependence of the corresponding base macroblocks where applicable. base_mode_flag equal to 0 specifies that mb_type is not inferred.

When base_mode_flag is not present, base_mode_flag shall be inferred as follows.

– If in_crop_window(CurrMbAddr) is equal to FALSE, the value of base_mode_flag shall be inferred to be equal to 0.

– Otherwise (in_crop_window(CurrMbAddr) is equal to TRUE), the value of base_mode_flag shall be inferred to be equal to 1.

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for EI, EP, and EB slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function). When a value is not applicable it is designated by "na". In the text, the value of mb_type may be referred to as the macroblock type and a value X of MbPartPredMode( ) may be referred to in the text by "X macroblock (partition) prediction mode" or as "X prediction macroblocks".

Table G-5 shows the allowed collective macroblock types for each slice_type.

Table G-5 – Allowed collective macroblock types for slice_type.

slice_type	allowed collective macroblock types
EI (slice)	I (see Table 7-11) (macroblock types)
EP (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)
EB (slice)	B (see Table 7-14) and I (see Table 7-11) (macroblock types)

Macroblock types that may be collectively referred to as I macroblock types are specified in Table G-6 and Table 7-11. mb_type value 0 is specified in Table G-6, whereas the other mb_type are specified in Table 7-11. Table G-6 additionally specifies the inferred macroblock type I_BL.

The macroblock types for EI slices are all I macroblock types.

Table G-6 – Macroblock types for EI slices.

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-33	Equation 7-33
0	I_NxN	1	Intra_8x8	na	Equation 7-33	Equation 7-33
inferred	I_BL	na	Intra_Base	na	Equation 7-33	Equation 7-33

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for EP slices are specified in Table 7-13, Table 7-11 and Table G-6. mb_type values 0 to 4 are specified in Table 7-13, mb_type value 5 is specified in Table G-6 indexed by subtracting 5 from the value of mb_type and mb_type values 6 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type. Table G-6 additionally specifies the inferred macroblock type I_BL.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for EB slices are specified in Table 7-14, Table 7-11 and Table G-6. mb_type values 0 to 22 are specified in Table 7-14, mb_type values 23 is specified in Table G-6 indexed by subtracting 23 from the value of mb_type and mb_type 24 to 48 are specified in Table 7-14, indexed by subtracting 23 from the value of mb_type. Table G-6 additionally specifies the inferred macroblock type I_BL.

G.7.4.6.1 Macroblock prediction in scalable extension semantics

The semantic specified in subclauses 7.4.5.1 shall apply. Additionally, the following semantics are specified.

motion_prediction_flag_l0[mbPartIdx **]** equal to 1 specifies that the (possibly scaled) base motion vector(s) is/are used as motion vector predictor(s) for the list 0 motion vector(s) of the macroblock partition mbPartIdx. When motion_prediction_flag_l0[mbPartIdx] is equal to 1, the reference index ref_idx_l0[mbPartIdx] is inferred using the base reference index and the components of the motion vector(s) mvL0 are obtained by adding the motion vector differences mvd_l0[mbPartIdx][ ] to the (possibly scaled) base motion vectors.

When motion_prediction_flag_l0[mbPartIdx] is not present, motion_prediction_flag_l0[mbPartIdx] shall be inferred to be equal to 0.

motion_prediction_flag_l1[mbPartIdx **]** has the same semantics as motion_prediction_flag_l0[mbPartIdx], with l0 and list 0 replaced by l1 and list 1, respectively.

G.7.4.6.2 Sub-macroblock prediction in scalable extension semantics

– The semantic specified in subclauses 7.4.5.2 shall apply.

G.7.4.6.3 Residual in scalable extension semantics

The semantic specified in subclauses 7.4.5.3 shall apply. Additionally, the following semantics are specified

residual_prediction_flag equal to 1 specifies that the residual signal is predicted by the (possibly upsampled) reconstructed residual signal of the base macroblock or sub-macroblock. residual_prediction_flag equal to 0 specifies that the residual signal is not predicted.

When the syntax element residual_prediction_flag is not present, residual_prediction_flag is inferred as follows.

– If all of the following conditions are true, residual_prediction_flag is inferred to be equal to 1.
  – base_mode_flag is equal to 1 or mb_type does not specify an I macroblock type
  – layer_base_flag is equal to 0
– Otherwise, residual_prediction_flag is inferred to be equal to 0.

smoothed_reference_flag equal to 1 specifies that smoothed reference sample prediction is used. When the syntax element smoothed_reference_flag is not present, it is inferred to be equal to 0.

G.7.4.7 Progressive refinement slice data in scalable extension semantics

Within a progressive refinement slice, the syntax elements for the representation of the transform coefficient levels for a slice are not parsed in the default scanning order. Syntax elements of various macroblocks are interleaved. The current macroblock is specified by the variable CurrMbAddr as specified in the syntax diagram in subclause G.7.3.7.

Depending on luma or chroma, the parsing of transform coefficient levels proceeds as follows

– Depending on the value of value of transform_size_8x8_flag for the current macroblock CurrMbAddr, which is either decoded or inferred using the base representation data as specified in subclause G.7.3.7, the following applies.
  – If transform_size_8x8_flag is equal to 0, for each of the 16 4x4 luma blocks indexed by i = 0..15, the 16 transform coefficient levels of the i-th block are parsed into the i-th list LumaLevel[i].
  – Otherwise, the 64 transform coefficient levels of the i8x8-th block are parsed into the i8x8-th list LumaLevel8x8[i8x8]

– When chroma_format_idc is not equal to 0, for each chroma component, indexed by iCbCr = 0..1, the DC transform coefficient levels of the 4 * NumC8x8 4x4 chroma blocks are parsed into the iCbCr-th list ChromaDCLevel[iCbCr].

– When chroma_format_idc is not equal to 0, for each of the 4x4 chroma blocks, indexed by i4x4 = 0..3 and i8x8 = 0..NumC8x8 – 1, of each chroma component, indexed by iCbCr = 0..1, the 15 AC transform coefficient levels are parsed into the (i8x8*4 + i4x4)-th list of the iCbCr-th chroma component ChromaACLevel[iCbCr][i8x8*4 + i4x4].

After a progressive refinement slice is completely parsed, all syntax elements and derived variables, to which a value NotCoded was assigned during the parsing process, are set equal to 0.

num_end_vals indicates the number of scan positions, commencing from the first scan position, for which the end-of-block (EOB) symbol is the least probable VLC symbol. **eobShiftLuma[** num_end_vals **]** indicates the probability rank of the end-of-block (EOB) symbol for luma coefficients at scan position num_end_vals.

eob_run indicates, for a given scan position, the number of scan positions for which the end-of-block (EOB) symbol has a probability rank one less than the EOB probability rank at the current scan position.

eob_shift_chroma[num_end_vals **]** indicates the probability rank of the end-of-block (EOB) symbol for chroma AC coefficients at scan position num_end_vals.

num_sig_vlc_selectors indicates number of vlc selectors that need to be parsed from the bitstream.

sig_vlc_selector indicates the VLC codebook that will be used for parsing the zero run or end-of_block (EOB) at the scan position in zigzag scan order. sig_vlc_selector is coded as a truncated unary code.

Table G-7 – Codeword table for sig_vlc_selector

sig_vlc_selector	Truncated unary code
0	1
1	01
2	001
3	0001
4	0000

NOTE – There is one vlc selector for each scan position in zigzag scan order. The vlc selector which is not coded in the slice has the default value of 0.

init_luma_cbp_flag indicates initial value for parsing luma code block pattern.

init_chroma_cbp indicates initial value for parsing chroma coded block pattern.

cycle_start_symbol indicates whether decoding of chroma AC refinement values should be deferred beyond one cycle.

cycle_start_symbol_ext indicates the number of cycles by which chroma AC refinement values should be deferred.

end_of_slice_flag is used to derive the variable PRsliceCompletelyDecoded, which specifies whether the progressive refinement slice has been completely decoded. When end_of_slice_flag is not present, end_of_slice_flag shall be inferred to be equal to 0.

When PRsliceCompletelyDecoded is equal to 0, the access unit shall not contain any progressive refinement slices prSlice for which all of the following conditions are true

– The variable dependency_id is equal to the value of the variable dependency_id for the slice with PRsliceCompletelyDecoded equal to 0.

– The variable quality_level is greater than the value of the variable quality_level for the slice with PRsliceCompletelyDecoded equal to 0.

G.7.4.7.1 Motion data in progressive refinement slice data semantics

base_mode_flag equal to 1 specifies that mb_type for the current macroblock is inferred. The reference indices and motion vectors shall be inferred in dependence of the corresponding base macroblocks where applicable. base_mode_flag equal to 0 specifies that mb_type is not inferred.

When more_cabac_data_in_nal_unit( ) is equal to 0 after parsing the syntax structure motion_data_in_progressive_refinement_slices( ), base_mode_flag is set equal to 1.

NOTE – The syntax element base_mode_flag is set equal to 1, when the motion data for the current macroblock could not be read completely.

mb_type has the same semantics as mb_type in subclause G.7.3.6.

residual_prediction_flag equal to 1 specifies that the residual signal is predicted by the residual signal of the collocated macroblock with the next lower quality_level. When residual_prediction_flag is equal to 1, the transform size of the current macroblock and the collocated macroblock shall be identical. residual_prediction_flag equal to 0 specifies that no residual signal prediction is employed. When residual_prediction_flag is not present, it shall be inferred to be equal to 0. When residual_prediction_flag is equal to 0, all transform coefficient levels of the base quality layer are considered to be equal to 0 for the parsing of the current slice.

G.7.4.7.1.1 Macroblock prediction in progressive refinement slice data semantics

The semantic specified in subclauses G.7.4.6.1 shall apply.

G.7.4.7.1.2 Sub-macroblock prediction in progressive refinement slice data semantics

The semantic specified in subclauses G.7.4.6.2 shall apply.

G.7.4.7.2 Macroblock Luma Coded Block Pattern in progressive refinement slice data CAVLC semantics

luma_cbp_run_minus1 plus 1 specifies the number of the consecutive Luma Coded Block Pattern bits of the same value.

G.7.4.7.3 Macroblock Chroma Coded Block Pattern in progressive refinement slice data CAVLC semantics

chroma_cbp_transition_flag indicates how Macroblock Chroma Coded Block Pattern changes from one value to one of two other values.

NOTE – The value of Macroblock Chroma Coded Block Pattern can be either 0, or 1, or 2.

chroma_cbp_run_minus1 plus 1 specifies the number of the consecutive Chroma Coded Block Pattern of the same value.

G.7.4.7.4 Luma coefficient in progressive refinement slice data semantics

coded_block_pattern_bit[i] represents the i-th bit (starting from the LSB) of the syntax element coded_block_pattern as it is specified in subclause 7.4.5. If the syntax element is not present, it shall be inferred to be equal to 0.

The combined syntax element coded_block_pattern is derived as follows:

```
coded_block_pattern = 0
for( i = 0; i < 6; i++ )    coded_block_pattern += ( coded_block_pattern_bit[ i ] << i )
```

Based on the value of coded_block_pattern, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as follows.

$$\text{CodedBlockPatternLuma} = \text{coded_block_pattern} \% 16 \quad \text{(G-71)}$$
$$\text{CodedBlockPatternChroma} = \text{coded_block_pattern} / 16 \quad \text{(G-72)}$$

The meaning of CodedBlockPatternLuma and CodedBlockPatternChroma is specified in subclause 7.4.5 [Ed. (JR) check that ref].

mb_qp_delta can change the value of $QP_{Y,PRk}$ with k being the value of quality_level in the macroblock layer. The decoded value of mb_qp_delta shall be in the range of $-(26 + QpBdOffset_Y / 2)$ to $+(25 + QpBdOffset_Y / 2)$, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock.

For each macroblock mbAddr, the values of $QP_{Y,PRk}$, $QP'_{Y,PRk}$, $QP_{C,PRk}$, and $QP'_{C,PRk}$ are derived as follows

– If function base_mb_sig(mbAddr) as specified in subclause G.7.2 is equal to FALSE, the following applies

  – The value of $QP_{Y,PRk}$ is derived as

$$QP_{Y,PRk} = ((QP_{Y,PRk,PREV} + mb_qp_delta + 52 + 2 * QpBdOffset_Y) \% (52 + QpBdOffset_Y)) - QpBdOffset_Y \quad (G\text{-}73)$$

  where $QP_{Y,PRk,PREV}$ is the luma quantisation parameter, QP_Y, of the macroblock for which the previous syntax element mb_qp_delta was decoded (and not inferred). At the beginning of the parsing process for a progressive refinement slice, $QP_{Y,PRk,PREV}$ is initially set to $SliceQP_Y$.

  – The value of $QP'_{Y,PRk}$ is derived as

$$QP'_{Y,PRk} = QP_{Y,PRk} + QpBdOffset_Y \quad (G\text{-}74)$$

  – The values $QP_{C,PRk}$ and $QP'_{C,PRk}$ are derived is specified in subclause 8.5.7 when replacing QP_Y, QP'_Y, QP_C, and QP'_C with $QP_{Y,PRk}$, $QP'_{Y,PRk}$, $QP_{C,PRk}$, and $QP'_{C,PRk}$, respectively.

– Otherwise, the following applies.

  – The derivation process for base quality slices in subclause G.6.1 is invoked with baseQualityLevel = quality_level – 1 as input and the output is assigned to baseQualitySlice.

  – With $QP_{Y,Base}$ and $QP_{C,Base}$ being the value of QP_Y and QP_C (if quality_level is equal to 1) or $QP_{Y,PR(k-1)}$ and $QP_{C,PR(k-1)}$ (otherwise) for the macroblock mbAddr in the slice baseQualitySlice, the following applies.

$$QP_{Y,PRk} = \max(0, (QP_{Y,Base} - 6)) \quad (G\text{-}75)$$
$$QP_{C,PRk} = \max(0, (QP_{C,Base} - 6)) \quad (G\text{-}76)$$

  – The values of $QP'_{Y,PRk}$ and $QP'_{C,PRk}$ are derived as

$$QP'_{Y,PRk} = QP_{Y,PRk} + QpBdOffset_Y \quad (G\text{-}77)$$
$$QP'_{C,PRk} = QP_{C,PRk} + QpBdOffset_C \quad (G\text{-}78)$$

transform_size_8x8_flag has the same semantics as transform_size_8x8_flag in subclause 7.4.5.

coded_block_flag_luma[i **]** represents the syntax element coded_block_flag for the i-th luma 4x4 block in scanning order. It specifies whether the block contains non-zero transform coefficient levels:

– If coded_block_flag_luma[i] is equal to 0, all transform coefficient levels LumaLevel[i][j] with j = 0..15 are equal to 0.

– Otherwise, at least one transform coefficient levels LumaLevel[i][j] with j = 0..15 is not equal to 0

coeff_sig_vlc_symbol specifies the information related to the position of significance coefficient in a block. If coeff_sig_vlc_symbol is not equal to end-of-block (EOB), it specifies the run of zeros in front of the next significant coefficient in the zigzag scan order**.** Otherwise, it indicates that the position of all the significant coefficients in a block has been specified.

sig_mag_flag [Ed. (JR) missing]

coef_mag_ref_bit [Ed. (JR) missing]

coef_mag_ref_bit [Ed. (JR) missing]

G.7.4.7.5 Chroma DC coefficient in progressive slice data semantics

coded_block_flag_chromaDC[iCbCr **]** represents the syntax element coded_block_flag for the chroma DC block of component iCbCr. It specifies whether the block contains non-zero transform coefficient levels:

– If coded_block_flag_chromaDC[iCbCr] is equal to 0, all transform coefficient levels ChromaDCLevel[iCbCr][j] with j = 0..NumC8x8-1 are equal to 0.

– Otherwise, at least one transform coefficient levels ChromaDCLevel[iCbCr][j] with j = 0..NumC8x8-1 is not equal to 0

G.7.4.7.6 Chroma AC coefficient in progressive refinement slice data semantics

coded_block_flag_chromaAC[iCbCr **][** i **]** represents the syntax element coded_block_flag for the i-th chroma AC block of component iCbCr. It specifies whether the block contains non-zero transform coefficient levels:

– If coded_block_flag_chromaAC[iCbCr][i] is equal to 0, all transform coefficient levels ChromaACLevel[iCbCr][i][j] with j = 0..14 are equal to 0.

– Otherwise, at least one transform coefficient levels ChromaACLevel[iCbCr][i][j] with j = 0..14 is not equal to 0

G.7.4.7.7 Significant coefficient in progressive slice data CABAC semantics

The semantic specified in subclauses 7.4.5.3.2 shall apply

last_significant_coeff_flag [Ed. (JR) shall this be re-specified, as the main doc uses the same syntax element with []]

significant_coeff_flag [Ed. (JR) shall this be re-specified, as the main doc uses the same syntax element with []]

coeff_sign_flag [Ed. (JR) shall this be re-specified, as the main doc uses the same syntax element with []]

coeff_abs_level_minus1 [Ed. (JR) shall this be re-specified, as the main doc uses the same syntax element with []]

G.7.4.7.8 Significant coefficient in progressive slice data CAVLC semantics

The semantic specified in subclauses 7.4.5.3.2 shall apply

coeff_sign_flag [Ed. (JR) shall this be re-specified, as the main doc uses the same syntax element with [], moreover this is normally used only for CABAC]

G.7.4.7.9 Coefficient refinement in progressive refinement slice data semantics

coeff_refinement_flag equal to 0 specifies that the value of a transform coefficient refinement level is equal to 0. coeff_refinement_flag equal to 1 specifies that the value of a transform coefficient refinement level is either equal to –1 or equal to 1.

coeff_refinement_direction_flag specifies the sign of a transform coefficient refinement level. If coeff_refinement_direction_flag is equal to 1, the sign of the transform coefficient refinement level is equal to the sign of its base representation; otherwise, the refinement level has the opposite sign.

G.8 Decoding process

[Ed. (HS): We should think about a re-ordering / re-structuring / re-classifying of subclause in F.8. I would somehow prefer to group similar subclauses as it has been done for F.8.11 in order to increase the readability.]

[Ed. (JR) as we are specifying only a single loop decoder (at least so far), it will have to be stated somewhere that constrained_intra_pred_flag must be 1 for all SVC layers, except maybe the last level…][Ed. (HS): Shouldn't that go into semantics.]

This clause specifies how the decoded picture is derived from syntax elements and global variables that are derived from NAL units in an access unit. The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Let the variable DependencyIdmax be equal to a value of dependency_id in the access unit.

[Ed. (JR) Shouldn't this condition be moved to the semantincs of dependency_id? It is not very clear that this is a constraints]

Let dId be any value in the range of 0..DependencyIdmax – 1, inclusive. The bit-stream that would be obtained by discarding all NAL units with dependency_id greater than dId shall be conforming to this Recommendation | International Standard.

For all NAL units in the access unit, let a variable dOiDX be derived as

$$dOiDX = dependency_id << 2 + quality_level \quad \text{(G-79)}$$

Let dOiDXmax be the largest value of dOiDX inside the access unit within the range of (DependencyIdmax << 2) to ((DependencyIdmax << 2) + 3), inclusive.

NOTE - dOiDXmax may differ for the access units in the coded video sequence within the range (DependencyIdmax << 2) and ((DependencyIdmax << 2) + 3), inclusive.

DependencyIdmax of any access unit shall be equal to the value of DependencyIdmax of the previous access unit in decoding order unless in the access unit the picture with dependency_id equal to DependencyIdmax is an IDR picture.

A variable EnhIdr is derived as follows.

– If nal_unit_type of any NAL unit with dependency_id equal to DependencyIdmax is equal to 5 or 21, EnhIdr is set equal to 1.

– Otherwise, EnhIdr is set equal to 0.

This clause describes the decoding process for one NAL unit with dOiDXmax and all NAL units within the access unit that the NAL unit with dOiDXmax depends on through inter-layer or intra-layer prediction. All these NAL units are specified as inputs to this process as described below. The syntax elements and upper-case variables derived for each of these NAL units are referred to by their values of dOiDX and passed as input to the processes that are invoked within this subclause.

Inputs to this process are syntax elements and upper-case variables from clause 7 and subclause G.7 for the following NAL units

– One NAL unit with a value of dOiDX being equal to dOiDXmax.

NOTE - Some NAL units with dOiDX being less then dOiDXmax that are specified as input below may not have to be necessarily input to a conforming decoding process for the NAL unit with dOiDXmax as they may be references for inter-layer or intra-layer prediction for samples at locations within the decoded picture that are outside the area that corresponds to the coded representation of the NAL unit with dOiDXmax.

– When dOiDXmax > (DependencyIdmax << 2), all NAL units with a value of dOiDX that is in the range from (DependencyIdmax << 2) to (dOiDXmax – 1), inclusive.

– Let the variable bID be equal to the maximum value of base_id of all NAL unit with dependency_id being equal to DependencyIdmax and quality_level being equal to 0.

– Let the variable lBF be equal to 0, if the syntax element layer_base_flag of any NAL unit, with dependency_id being equal to DependencyIdmax and quality_level being equal to 0, is equal to 0. Otherwise lBF is set equal to 1.

– While lBF is equal to 0, the following NAL units are inputs to this process

  – Let the variables dependencyIdBase, qualityLevelBase, and fragmentOrderBase are derived by

$$dependencyIdBase = bID >> 4 \quad \text{(G-80)}$$
$$qualityLevelBase = (bID >> 2) \& 3 \quad \text{(G-81)}$$
$$fragmentOrderBase = bID \& 3 \quad \text{(G-82)}$$

  – The NAL units with dependency_id being equal to dependencyIdBase and quality_level being equal to qualityLevelBase and fragment_order being less than or equal to fragmentOrderBase and discardable_flag being equal to 0.

  – The NAL units with dependency_id being equal to dependencyIdBase and quality_level being less than qualityLevelBase and discardable_flag being equal to 0.

  – Set bID equal to the maximum value of base_id of all NAL unit with dependency_id being equal to dependencyIdBase and quality_level being equal to 0.

  – Set the variable lBF be equal to 0, if the syntax element layer_base_flag, of any NAL unit with dependency_id equal to dependencyIdBase and quality_level equal to 0, is equal to 0. Otherwise lBF is set equal to 1.

Outputs of this process are decoded samples of the current access unit.

Let dOiDXmin be the smallest value of dOiDX for the set of NAL units that are input to the process specified in this subclause.

All NAL units with the value of the syntax element first_mb_in_slice being equal and dOiDX being equal are concatenated. [Ed. (TW) be more specific about how the fragments are being concatenated]

The array assignment and initialization process in subclause G.8.1 is invoked with syntax elements and global variables derived for any NAL unit with dOiDX equal to dOiDXmin as input and currentVars as output.

Variables and functions relating to picture order count are derived by invoking the process specified in subclause 8.2.1 with the syntax elements of the slice header(s) with dOiDX equal to ((dOiDXmax >> 2) << 2). Identical variables and function relating to picture order count shall be obtained when the process in subclause 8.2.1 is invoked with syntax elements of slice header(s) with any value of dependency_id less than DependencyIdmax and quality_level equal to 0 for all access units in the coded video sequence.

The decoding process for gaps in frame_num as specified in subclause 8.2.5.2 is invoked with the syntax elements and derived upper-case variables of the slice header(s) with dependency_id equal to DependencyIdmax and quality_level equal to 0.

[Ed. (HS): we also need to invoke the processes 8.2.2 (one per dependency_id) and may be 8.2.3 somewhere]

Let dOiDXbase be equal to dOiDXmin. For all values of dOiDX with dOiDXmin <= dOiDX <= dOiDXmax in increasing order, the following applies

- Depending on dOiDX and on SpatialScalabilityType, the following applies.
  - If dOiDX is equal to dOiDXmin or if SpatialScalabilityType is equal to 0, the following applies.
    - For all NAL units with an equal value of dOiDX the following applies.
      - The base slice decoding process as specified in G.8.2 is invoked with syntax elements and global variables derived for the NAL unit with dOiDX and currentVars as input and the output is a modified version of currentVars.
  - Otherwise (dOiDX is greater than dOiDXmin and SpatialScalabilityType is not equal to 0), the following applies.
    - The variables that are assigned to the collective term currentVars are assigned to the collective term baseVars.
    - The array initialization process as specified in subclause G.8.1 is invoked with syntax elements and global variables derived for the NAL unit with dOiDX as input and currentVars being the output.
    - For all NAL units with an equal value of dOiDXbase the following applies
      - The base slice decoding process prior to resampling as specified in G.8.3.1 is invoked with syntax elements and global variables derived for the NAL unit with dOiDXbase, baseVars as input and the output is a modified version of baseVars
    - The deblocking filter process for Intra_Base prediction as specified in G.8.9.1 is invoked with syntax elements and global variables derived for the NAL unit with dOiDXbase, baseVars as input and the output is a modified version of baseVars
    - For all NAL units with an equal value of dOiDX the following applies
      - The base slice decoding process as specified in G.8.3 is invoked with syntax elements and global variables derived for the NAL unit with dOiDX, baseVars and currentVars as input and the output is a modified version of currentVars.
- A variable decodeBaseRepresentationFlag is derived as follows
  - If all of the following conditions are true, decodeBaseRepresentationFlag is set equal to 1
    - dOiDX is less than dOiDXmax
    - quality_level is equal to 0
    - dependency_id is equal to DependencyIdmax
    - store_base_rep_flag is equal to 1
  - Otherwise, decodeBaseRepresentationFlag is set equal to 0.
- When decodeBaseRepresentationFlag is equal to 1, the following applies.

– The variables that are assigned to the collective term currentVars are assigned to the collective term baseRepVars.

NOTE – Any modification of the variables baseRepVars does not influence the variables currentVars.

– The target slice decoding process as specified in subclause G.8.4 is invoked with syntax elements and global variables derived for the NAL unit with dOiDX and baseRepVars as the input and the output being constructed samples prior to the deblocking filter process that are stored in the arrays S'_L, and, when chroma_format_idc is not equal to 0, S'_{Cb} and S'_{Cr}.

– The constructed samples prior to the deblocking filter process that are stored in the arrays S'_L, and, when chroma_format_idc is not equal to 0, S'_{Cb} and S'_{Cr} are processed by a deblocking filter as specified in subclause G.8.14 with the output being the decoded samples stored in the arrays S_L, and, when chroma_format_idc is not equal to 0, S_{Cb} and S_{Cr}. The syntax elements and global variables that shall be considered current and used are those from the NAL unit with dOiDX.

– All sample values the arrays S_L are copied to the array B_L.

– When chroma_format_idc is not equal to 0, all sample values of the arrays S_{Cb} and S_{Cr} are copied to the arrays B_{Cb} and B_{Cr}, respectively.

NOTE – The sample arrays B_L, B_{Cb}, and B_{Cr} represent the base representation of a decoded key pictures. This picture is marked as “used for short-term reference” or “used for long-term reference” as specified in subclause G.8.13, but it does not represent an output of the decoding process.

– Set dOiDXbase equal to dOiDX

The target slice decoding process as specified in subclause G.8.4is invoked with syntax elements and global variables derived for the NAL unit with dOiDXmax and currentVars from the previous invocation of the base slice decoding process as the input and the output being constructed samples prior to the deblocking filter process.

The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter as specified in subclause G.8.14 with the output being the decoded samples. The syntax elements and global variables that shall be considered current and used are those from the NAL unit with dOiDXmax.

When the value of nal_ref_idc for the NAL unit(s) with dOiDXmax is larger than 0 and after all NAL units with dOiDXmax have been decoded, the decoded reference picture marking process as specified in subclause G.8.13 is invoked with dOiDXmax as input. [Ed. (HS): All values of FrameNum, FrameNumWrap, and PicNum (LongTermPicNum?) that are used in this process are values for DependencyIdmax.]

G.8.1 Array assignment initialization process

Input to this process is a collective term currentVars.

Output of this process is a set of arrays that are assigned to the collective term currentVars.

The following arrays are collectively referred to as currentVars.

– An array fieldMb with PicSizeInMbs elements specifying whether a macroblock is a field or frame macroblock. An element of this array for a macroblock with address mbAddr is referred to as fieldMb[mbAddr]. All elements of the array fieldMb are initially marked as “unspecified”.

– An array mbType with PicSizeInMbs elements specifying macroblock types. An element of this array for a macroblock with address mbAddr is referred to as mbType[mbAddr]. All elements of the array mbType are initially marked as “unspecified”.

– An array lumaTrafo with PicSizeInMbs elements specifying the transforms for the luma component. An element of this array for a macroblock with address mbAddr is referred to as lumaTrafo[mbAddr]. Any element of lumaTrafo can have one of the following values: T_4x4, T_8x8, T_16x16, T_IPCM. All elements of the array lumaTrafo are initially marked as “unspecified”.

– An array subMbType with (PicSizeInMbs)x4 elements specifying sub-macroblock types. An element of this array for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as subMbType[mbAddr][mbPartIdx]. All elements of the array subMbType are initially marked as “unspecified”.

– An array ipred4x4 with (PicSizeInMbs)x16 elements specifying Intra_4x4 prediction modes. An element of this array for a macroblock with address mbAddr and a 4x4 luma block index luma4x4BlkIdx is referred to as ipred4x4[mbAddr][luma4x4BlkIdx]. All elements of the array ipred4x4 are initially marked as “unspecified”.

– An array ipred8x8 with (PicSizeInMbs)x4 elements specifying Intra_8x8 prediction modes. An element of this array for a macroblock with address mbAddr and a 8x8 luma block index luma8x8BlkIdx is referred to as ipred8x8[mbAddr][luma8x8BlkIdx]. All elements of the array ipred8x8 are initially marked as "unspecified".

– Two arrays refIdxL0 and refIdxL1 with (PicSizeInMbs)x4 elements specifying reference indices for list 0 and list 1, respectively. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as refIdxLX[mbAddr][mbPartIdx], with X being replaced by 0 or 1. All elements of the arrays refIdxL0 and refIdxL1 are initially set equal to –1.

– Two arrays predFlagL0 and predFlagL1 with (PicSizeInMbs)x4 elements specifying prediction utilization flags for list 0 and list 1, respectively. An element of these arrays for a macroblock with address mbAddr and a macroblock partition index mbPartIdx is referred to as predFlagLX[mbAddr][mbPartIdx], with X being replaced by 0 or 1. All elements of the arrays predFlagL0 and predFlagL1 are initially set equal to 0.

– Two arrays mvL0 and mvL1 with (PicSizeInMbs)x4x4 elements specifying motion vectors for list 0 and list 1, respectively. An element of these arrays for a macroblock with address mbAddr, a macroblock partition index mbPartIdx, and a macroblock sub-partition index subMbPartIdx is referred to as mvLX[mbAddr][mbPartIdx][subMbPartIdx], with X being replaced by 0 or 1. The motion vectors of macroblock partitions can also be referred to as mvLX[mbAddr][mbPartIdx], which is identical to mvLX[mbAddr][mbPartIdx][0]. All elements of the arrays mvL0 and mvL1 are initially set equal to 0.

– An array sTCoeff with (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) elements specifying scaled transform coefficient values. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as sTCoeff[mbAddr][tCoeffIdx]. All elements of the array sTCoeff are initially set equal to 0.

– An array sigBCoeff with (PicSizeInMbs)x(256 + 2 * MbWidthC * MbHeightC) elements specifying significance indicators for transform coefficient levels in slices with quality_level equal to 0. An element of this array for a macroblock with address mbAddr and a transform coefficient index tCoeffIdx is referred to as sigBCoeff[mbAddr][tCoeffIdx]. All elements of the array sigBCoeff are initially set equal to 0.

– An array ctx4x4Id with (PicSizeInMbs)x(16) elements specifying context identifiers for 4x4 luma blocks. An element of this array for a macroblock with address mbAddr and a 4x4 luma block index luma4x4BlkIdx is referred to as ctx4x4Id[mbAddr][luma4x4BlkIdx]. All elements of the array ctx4x4Id are initially set equal to 0.

– Any array rS_L with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements specifying residual luma sample values. An element of this array with a luma location (x, y) relative to the upper-left luma sample of the macroblock with address 0 is referred to as rS_L[x, y]. All elements of the array rS_L are initially set equal to 0.

– When chroma_format_idc is not equal to 0, two array rS_{Cb} and rS_{Cr} with ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) elements specifying residual chroma sample values for the Cb and Cr component, respectively. An elements of these arrays with a chroma location (x, y) relative to the upper left chroma sample of the macroblock with address 0 is referred to as rS_C[x, y], with C being replaced by Cb or Cr. All elements of the arrays rS_{Cb} and rS_{Cr} are initially set equal to 0.

– Any array cS_L with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements specifying constructed luma sample values. An element of this array with a luma location (x, y) relative to the upper-left luma sample of the macroblock with address 0 is referred to as cS_L[x, y]. All elements of the array cS_L are initially set equal to 0.

– When chroma_format_idc is not equal to 0, two array cS_{Cb} and cS_{Cr} with ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) elements specifying residual chroma sample values for the Cb and Cr component, respectively. An elements of these arrays with a chroma location (x, y) relative to the upper left chroma sample of the macroblock with address 0 is referred to as cS_C[x, y], with C being replaced by Cb or Cr. All elements of the arrays cS_{Cb} and cS_{Cr} are initially set equal to 0.

G.8.2 Base slice decoding process without resolution change

Input to this process are

– syntax elements and global variables derived from a NAL unit

– a set of arrays collectively referred to as currentVars

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and global variables of the NAL unit and the values of the variables assigned to currentVars that are the input.

In the following of this subclause, the variables fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr} of the collective term currentVars are referred to as fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1, mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr}, respectively.

From here to the end of this subclause, it is assumed that the syntax elements and global variables from the NAL unit currently being decoded are the current syntax elements and global variables and are available. This also applies to all invoked processes and all recursive invoking.

When the current slice is a P, EP, B, or EB slice or when the current slice is a PR slice with motion_refinement_flag equal to 1, and BaseSliceType equal to P, EP, B, or EB, the decoding process for reference picture lists construction as specified in subclause G.8.12 is invoked with currDependencyId equal to dependency_id as the input and the outputs are a reference picture list refPicList0 and, when applicable, a reference picture list refPicList1.

The macroblocks of the NAL unit currently being decoded are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, CurrMbAddr is set equal to mbAddr, and depending on the value of slice_type the following applies.

- If slice_type is equal PR, depending on mbType[mbAddr] the following applies.
  - If mbType[mbAddr] specifies an I macroblock, the following applies.
    - The progressive refinement process for scaled transform coefficients as specified in subclause G.8.11.3 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo[mbAddr] and the list of scaled transform coefficient values sTCoeff[mbAddr] as the inputs and a modified version of the list of scaled transform coefficient values sTCoeff[mbAddr] as the output.
  - Otherwise (mbType[mbAddr] does not specify an I macroblock), the following applies.
    - When motion_refinement_flag is equal to 1, the following applies.
      - When base_mode_flag is equal to 0, mbType[mbAddr] is set equal to mbType, and for i = 0..3, subMbType[mbAddr][i] is set equal sub_mb_type[mbAddr][i].
      - The motion data derivation process as specified in subclause G.8.5 is invoked with mbAddr, mbType[mbAddr], subMbType[mbAddr], predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], mvL1[mbAddr] (if applicable), refPicList0, and refPicList1 (if applicable) as the inputs and the outputs are modified versions of predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], and mvL1[mbAddr] (if applicable).
    - Depending on transform_size_8x8_flag, the following applies.
      - If transform_size_8x8_flag is equal to 1, lumaTrafo[mbAddr] is set equal to T_8x8.
      - Otherwise (transform_size_8x8_flag is equal to 0), lumaTrafo[mbAddr] is equal to T_4x4.
    - Depending on motion_refinement_flag and residual_prediction_flag, the following applies.
      - If motion_refinement_flag is equal to 1 and residual_prediction_flag is equal to 0, the transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo[mbAddr] as the input and the outputs are assigned to the list of scaled transform coefficient values sTCoeff[mbAddr], the list of significance indicators sigBCoeff[mbAddr], and the list of context identifiers ctx4x4Id.
      - Otherwise (motion_refinement_flag is equal to 0 or residual_prediction_flag is equal to 1), the progressive refinement process for scaled transform coefficients as specified in subclause G.8.11.3 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo[mbAddr] and the list of scaled transform coefficient values sTCoeff[mbAddr] as the inputs and a modified version of the list of scaled transform coefficient values sTCoeff[mbAddr] as the output.
- Otherwise (slice_type is not equal to PR), the following applies.

- The initialization process for macroblock, sub-macroblock, and luma transform type as specified in subclause G.8.11.1 is invoked with mbAddr, mbType, subMbType, predFlagL0, predFlagL1, and the syntax elements of the macroblock mbAddr as input and the outputs are assigned to fieldMb[mbAddr], mbType[mbAddr], subMbType[mbAddr], and lumaTrafo[mbAddr].
- Depending on the macroblock mbAddr, the following applies.
  - If the macroblock mbAddr is coded using an I macroblock type, the following applies.
    - If mb_type is equal to I_BL, the following applies
      - The residual construction process as specified in subclause G.8.11.5 is invoked with mbAddr, lumaTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are modified versions of rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.
      - The sample construction process as specified in subclause G.8.11.7 is invoked with mbAddr, rS_L, cS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb}, rS_{Cr}, cS_{Cr}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when chroma_format_idc is not equal to 0, modified versions of cS_{Cr} and cS_{Cr}.
      - The transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo [mbAddr] as the input and the outputs are assigned to the list of scaled transform coefficient values sTCoeff[mbAddr], the list of significance indicators sigBCoeff[mbAddr], and the list of context identifiers ctx4x4Id.
      - The residual construction process as specified in subclause G.8.11.5 is invoked with mbAddr, lumaTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}as the inputs and the outputs are modified versions of rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.
      - The sample construction process as specified in subclause G.8.11.7 is invoked with mbAddr, rS_L, cS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb}, rS_{Cr}, cS_{Cr}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when chroma_format_idc is not equal to 0, modified versions of cS_{Cr} and cS_{Cr}.
      - All (256 + 2 * MbWidthC * MbHeightC) elements of the list of scaled transform coefficient values sTCoeff[mbAddr] are set equal to 0.
    - Otherwise (mb_type is not equal to I_BL), the following applies.
      - The intra macroblock decoding process as specified in subclause G.8.8 is invoked with fieldMb[mbAddr], mbType[mbAddr], subMbType[mbAddr], lumaTrafo[mbAddr], ipred4x4, ipred8x8, cS_L, and, when chroma_format_idc is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are sTCoeff[mbAddr] as well as modified versions of ipred4x4, ipred8x8, cS_L, and, when chroma_format_idc is not equal to 0, cS_{Cb} and cS_{Cr}.
  - Otherwise (the macroblock mbAddr is not coded using an I macroblock type), the following applies.
    - The motion data derivation process as specified in subclause G.8.5 is invoked with mbAddr, mbType[mbAddr], subMbType[mbAddr], predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], mvL1[mbAddr] (if applicable), refPicList0, and refPicList1 (if applicable) as the inputs and the outputs are modified versions of predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], and mvL1[mbAddr] (if applicable).
    - When the syntax element residual_prediction_flag of the macroblock mbAddr is equal to 1, the following applies.
      - The residual accumulation process as specified in subclause G.8.11.4 is invoked with mbAddr, lumaTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are modified versions of rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.

– The transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with the luma transform type fieldMb[mbAddr], lumaTrafo[mbAddr] as the input and the outputs are assigned to the list of scaled transform coefficient values sTCoeff[mbAddr], the list of significance indicators sigBCoeff[mbAddr], and the list of context identifiers ctx4x4Id.

G.8.3 Base slice decoding process with resolution change

Input to this process are

– syntax elements and global variables derived from a NAL unit

– a set of arrays collectively referred to as baseVars

– a set of arrays collectively referred to as currentVars

Output of this process is the modified set of arrays collectively referred to as currentVars.

This process modifies the variables assigned to currentVars using syntax elements and global variables of the NAL unit and the values of the variables assigned to baseVars that are the input.

In the following of this subclause, the variables fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, sTCoef, sigBCoeff, ctx4x4Id f, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr} of the collective term currentVars are referred to as fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1, mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr}, respectively.

The variables fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr} of the collective term baseVars are referred to as fieldMbBase, mbTypeBase, lumaTrafoBase, subMbTypeBase, ipred4x4Base, ipred8x8Base, refIdxL0Base, refIdxL1Base, predFlagL0Base, predFlagL1Base, mvL0Base, mvL1Base, sTCoeffBase, sigBCoeffBase, ctx4x4IdBase, rB_L, rB_{Cb}, rB_{Cr}, cB_L, cB_{Cb}, and cB_{Cr}, respectively.

From here to the end of this subclause, it is assumed that the syntax elements and global variables from the NAL unit currently being decoded are the current syntax elements and global variables and are available. This also applies to all invoked processes and all recursive invoking.

When the current slice is a P, EP, B, or EB slice, the decoding process for reference picture lists construction as specified in subclause G.8.12 is invoked with currDependencyId equal to dependency_id as the input and the outputs are a reference picture list refPicList0 and for B and EB slice a reference picture list refPicList1.

The macroblocks of the NAL unit currently being decoded are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, CurrMbAddr is set equal to mbAddr, and the following applies.

– The initialization process for macroblock, sub-macroblock, and luma transform type as specified in subclause G.8.11.1 is invoked with mbAddr, mbTypeBase, subMbTypeBase, predFlagL0Base, predFlagL1Base, and the syntax elements of the macroblock mbAddr as input and the outputs are assigned to fieldMb[mbAddr], mbType[mbAddr], subMbType[mbAddr], and lumaTrafo[mbAddr].

– Depending on the macroblock mbAddr, the following applies.

  – If the macroblock mbAddr is coded using an I macroblock type, the following applies.

    – If mb_type is equal to I_BL, the following applies.

      – The resampling process for intra samples as specified in subclause G.8.9 is invoked with mbAddr, lumaTrafoBase, mbTypeBase, cB_L, cS_L, and, when chroma_format_idc is not equal to 0, cB_{Cb}, cB_{Cr}, cS_{Cb}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L, and, when chroma_format_idc is not equal to 0, modified versions of cS_{Cb} and cS_{Cr}.

      – The transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo[mbAddr] as the input and the outputs are assigned to the list of scaled transform coefficient values sTCoeff[mbAddr], the list of significance indicators sigBCoeff[mbAddr], and the list of context identifiers ctx4x4Id.

      – The residual construction process as specified in subclause G.8.11.5 is invoked with mbAddr, lumaTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when chroma_format_idc is not equal to 0,

rS_{Cb} and rS_{Cr} as the inputs and the outputs are modified versions of rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.

– The sample construction process as specified in subclause G.8.11.7 is invoked with mbAddr, rS_L, cS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb}, rS_{Cr}, cS_{Cr}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when chroma_format_idc is not equal to 0, modified versions of cS_{Cr} and cS_{Cr}.

– All (256 + 2 * MbWidthC * MbHeightC) elements of the list of scaled transform coefficient values sTCoeff[mbAddr] are set equal to 0.

– Otherwise (mb_type is not equal to I_BL), the following applies.

– The intra macroblock decoding process as specified in subclause G.8.8 is invoked with fieldMb[mbAddr], mbType[mbAddr], subMbType[mbAddr], lumaTrafo[mbAddr], ipred4x4, ipred8x8, cS_L, and, when chroma_format_idc is not equal to 0, cS_{Cb} and cS_{Cr} as the inputs and the outputs are sTCoeff[mbAddr] as well as modified versions of ipred4x4, ipred8x8, cS_L, and, when chroma_format_idc is not equal to 0, cS_{Cb} and cS_{Cr}.

– Otherwise (the macroblock mbAddr is not coded using an I macroblock type), the following applies.

– When base_mode_flag is equal to 1, the resampling process for motion data as specified in subclause G.8.6 is invoked with mbAddr, mbType[mbAddr], subMbType[mbAddr], the arrays refIdxLX[mbAddr], mvLX[mbAddr], refPicListX[mbAddr], and the arrays mbTypeBase, subMbTypeBase, predFlagLXBase, refIdxLXBase, and mvLXBase as input and the arrays mbType[mbAddr], subMbType[mbAddr], refIdxLX[mbAddr], mvLX[mbAddr], and refPicListX[mbAddr] as output, with X being replaced by 0 or 1.

– The motion data derivation process as specified in subclause G.8.5 is invoked with mbAddr, mbType[mbAddr], subMbType[mbAddr], predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], mvL1[mbAddr] (if applicable), refPicList0, and refPicList1 (if applicable) as the inputs and the outputs are modified versions of predFlagL0[mbAddr], predFlagL1[mbAddr] (if applicable), refIdxL0[mbAddr], refIdxL1[mbAddr] (if applicable), mvL0[mbAddr], and mvL1[mbAddr] (if applicable).

– When the syntax element residual_prediction_flag of the macroblock mbAddr is equal to 1, or when the syntax element smoothed_reference_flag is equal to 1, the following applies.

– The resampling process for residual data as specified in subclause G.8.10 is invoked with mbAddr, lumaTrafoBase, rB_L, rS_L, and, when chroma_format_idc is not equal to 0, rB_{Cb}, rB_{Cr}, rS_{Cb}, and rS_{Cr} as the inputs and the outputs are a modified version of rS_L and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.

– The transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with fieldMb[mbAddr], the luma transform type lumaTrafo[mbAddr] as the input and the outputs are assigned to the list of scaled transform coefficient values sTCoeff[mbAddr], the list of significance indicators sigBCoeff[mbAddr], and the list of context identifiers ctx4x4Id.

G.8.3.1 Base slice decoding process prior to resampling

Input to this process are

– syntax elements and global variables derived from a NAL unit

– a set of arrays collectively referred to as baseVars

Output of this process is the modified set of arrays collectively referred to as baseVars.

This process modifies the variables assigned to baseVars using syntax elements and global variables of the NAL unit and the values of the variables assigned to baseVars that are the input.

From here to the end of this subclause, it is assumed that the syntax elements and global variables from the NAL unit currently being decoded are the current syntax elements and global variables and are available. This also applies to all invoked processes and all recursive invoking.

In the following of this subclause, the variables fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb},

and cS_{Cr} of the collective term baseVars are referred to as fieldMbBase, mbTypeBase, lumaTrafoBase, subMbTypeBase, ipred4x4Base, ipred8x8Base, refIdxL0Base, refIdxL1Base, predFlagL0Base, predFlagL1Base, mvL0Base, mvL1Base, sTCoeffBase, sigBCoeffBase, ctx4x4IdBase, rB_L, rB_{Cb}, rB_{Cr}, cB_L, cB_{Cb}, and cB_{Cr}, respectively.

The macroblocks of the NAL unit currently being decoded are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, CurrMbAddr is set equal to mbAddr, and the following applies.

- Depending on the macroblock mbAddr, the following applies.
  - If the macroblock mbAddr is coded using an I macroblock type, the following applies.
    - The residual construction process as specified in subclause G.8.11.5 is invoked with mbAddr, lumaTrafoBase[mbAddr], sTCoeffBase[mbAddr], rB_L, and, when chroma_format_idc is not equal to 0, rB_{Cb} and rB_{Cr} as the inputs and the outputs are modified versions of rB_L, and, when chroma_format_idc is not equal to 0, rB_{Cb} and rB_{Cr}.
    - The sample construction process as specified in subclause G.8.11.7 is invoked with mbAddr, rB_L, cB_L, and, when chroma_format_idc is not equal to 0, rB_{Cb}, rB_{Cr}, cB_{Cr}, and cB_{Cr} as the inputs and the outputs are a modified version of cB_L and, when chroma_format_idc is not equal to 0, modified versions of cB_{Cr} and cB_{Cr}.
  - Otherwise (the macroblock mbAddr is not coded using an I macroblock type), the following applies.
    - The residual accumulation process as specified in subclause G.8.11.4 is invoked with mbAddr, lumaTrafoBase[mbAddr], sTCoeffBase[mbAddr], rB_L, and, when chroma_format_idc is not equal to 0, rB_{Cb} and rB_{Cr} as the inputs and the outputs are modified versions of rB_L, and, when chroma_format_idc is not equal to 0, rB_{Cb} and rB_{Cr}.

G.8.4 Target slice decoding process

Input to this process are

- syntax elements and global variables derived from a NAL unit
- a set of arrays collectively referred to as currentVars

Output of this process are constructed samples prior to the deblocking filter process.

In the following of this subclause, the variables fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1. mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr} of the collective term currentVars are referred to as fieldMb, mbType, lumaTrafo, subMbType, ipred4x4, ipred8x8, refIdxL0, refIdxL1, predFlagL0, predFlagL1, mvL0, mvL1, sTCoeff, sigBCoeff, ctx4x4Id, rS_L, rS_{Cb}, rS_{Cr}, cS_L, cS_{Cb}, and cS_{Cr}, respectively.

From here to the end of this subclause, it is assumed that the syntax elements and global variables from the NAL unit currently being decoded are known and considered to be the current ones. This also applies to all invoked processes.

When the current slice is a P, EP, B, or EB slice or when the current slice is a PR slice with BaseSliceType equal to P, EP, B, or EB, the following applies.

- The decoding process for reference picture lists construction as specified in subclause G.8.12 is invoked with currDependencyId equal to dependency_id as the input and the outputs are a reference picture list refPicList0 and, when applicable, a reference picture list refPicList1.
- The decoding process for reference picture lists construction as specified in subclause G.8.12 is invoked with currDependencyId equal to dependency_id and useBaseRepFlag equal to 1 as the input and the outputs are a reference picture list refPicListBase0 and, when applicable, a reference picture list refPicListBase1.

The macroblocks of the NAL unit currently being decoded are processed in increasing order of their macroblock addresses. For each macroblock with macroblock address mbAddr, CurrMbAddr is set equal to mbAddr, and the following applies.

- The residual construction process as specified in subclause G.8.11.5 is invoked with mbAddr, lumaTrafo[mbAddr], sTCoeff[mbAddr], rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr} as the inputs and the outputs are modified versions of rS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb} and rS_{Cr}.

– When mbType[mbAddr] specifies an P or B macroblock type, the following applies.

  – The inter prediction process as specified in subclause G.8.7 is invoked with mbAddr, lumaTrafo[mbAddr], mbType[mbAddr], subMbType[mbAddr], the predFlagL0[mbAddr], predFlagL1[mbAddr], refIdxL0[mbAddr], refIdxL1[mbAddr], mvL0[mbAddr], mvL1[mbAddr], refPicList0, refPicList1, refPicListBase0, refPicListBase1, sigBCoeff[mbAddr], ctx4x4Id[mbAddr], rS_L, cS_L, and $logWD_L$, w_{0L}, w_{1L}, o_{1L}, and o_{0L}, which have been derived as specified in subclause G.8.7.4,and, when chroma_format_idc is not equal to 0, rS_{Cb}, rS_{Cr}, cS_{Cb}, cS_{Cr}, and $logWD_C$, w_{0C}, w_{1C}, o_{0C}, and o_{1C}, which have been derived as specified in subclause G.8.7.4, as the inputs and the outputs are a modified version of cS_L, and, when chroma_format_idc is not equal to 0, modified version of cS_{Cb} and cS_{Cr}.

– The sample construction process as specified in subclause G.8.11.7 is invoked with mbAddr, rS_L, cS_L, and, when chroma_format_idc is not equal to 0, rS_{Cb}, rS_{Cr}, cS_{Cr}, and cS_{Cr} as the inputs and the outputs are a modified version of cS_L and, when chroma_format_idc is not equal to 0, modified versions of cS_{Cr} and cS_{Cr}.

– The picture construction process prior to deblocking filter process as specified in subclause G.8.11.6 is invoked with mbAddr, cS_L and, when chroma_format_idc is not equal to 0, cS_{Cr} and cS_{Cr} as the inputs.

G.8.5 Motion data derivation process

Inputs to this process are

– a macroblock address mbAddr,

– a variable mbType specifying the macroblock type for the macroblock mbAddr,

– a list subMbType of 4 elements specifying the sub-macroblock types for the macroblock mbAddr (if applicable),

– two arrays predFlagL0 and predFlagL1 of prediction utilization flags,

– two arrays of reference indices refIdxL0 and refIdxL1,

– two arrays of luma motion vectors mvL0 and mvL1,

– reference picture lists refPicList0 and refPicList1.

Output of this process are

– modified versions of the two arrays predFlagL0 and predFlagL1 of prediction utilization flags,

– modified versions of the two arrays of reference indices refIdxL0 and refIdxL1,

– modified versions of the two arrays of luma motion vectors mvL0 and mvL1.

Let the syntax elements and upper case variables for the macroblock with the macroblock address mbAddr be available.

Let the variable MvCnt be initially set equal to 0.

The variable numMbPart is derived as follows.

– If mbType is equal to B_Skip or B_Direct_16x16, numMbPart is set equal to 4.

– Otherwise, numMbPart is set equal to NumMbPart(mbType).

The macroblock partition index mbPartIdx proceeds over values 0..numMbPart – 1, and for each value of mbPartIdx the following applies.

– The variable numSubMbPart is derived as follows.

  – If mbType is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, numSubMbPart is set equal to 1.

  – Otherwise, if mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is not equal to B_Direct_8x8, numSubMbPart is set equal to NumSubMbPart(subMbType[mbPartIdx]).

  – Otherwise (mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8, numSubMbPart is set equal to 4.

- The sub-macroblock partition index proceeds over values 0..numSubMbPart – 1, and for each value of subMbPartIdx the following applies.
  - If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The sub-partition motion vector count variable subMvCnt is set equal to 1.
  - Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or subMbTypeCurr is equal to B_Direct_8x8, subclause G.8.5.1 is invoked with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output.
  - Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX the following applies.
    - If MbPartPredMode(mbTypeCurr, mbPartIdx) or SubMbPredMode(subMbTypeCurr) is equal to Pred_LX or to BiPred, the following applies.
      - If base_mode_flag is equal to 1,

        predFlagLX = 1 (G-83)
        mvLX[0] = mvLX[mbPartIdx][subMbPartIdx][0] (G-84)
        mvLX[1] = mvLX[mbPartIdx][subMbPartIdx][1] (G-85)
        refIdxLX = refIdxLX[mbPartIdx] (G-86)

      - Otherwise, the following applies.
        - The reference index refIdxLX and a luma motion vector prediction mvpLX are derived as follows.
        - If motion_prediction_flag_LX[mbPartIdx] is equal to 1, the following applies.

          mvpLX[0] = mvLX[mbPartIdx][subMbPartIdx][0] (G-87)
          mvpLX[1] = mvLX[mbPartIdx][subMbPartIdx][1] (G-88)
          refIdxLX = refIdxLX[mbPartIdx] (G-89)

        - Otherwise, the following applies.
          - The reference index refIdxLX is set equal to ref_idx_lX[mbPartIdx]
          - The derivation process for luma motion vector prediction in subclause 8.4.1.3 is invoked with mbPartIdx subMbPartIdx, refIdxLX, and list suffix LX as the input and the output is assigned to mvpLX.
        - The variable subMvCnt for sub-partition motion vector count is set equal to predFlagL0 + predFlagL1.
        - The prediction utilization flag predFlagLX and the luma motion vector mvLX are derived by

          predFlagLX = 1 (G-90)
          mvLX[0] = mvpLX[0] + mvd_lX[mbPartIdx][subMbPartIdx][0] (G-91)
          mvLX[1] = mvpLX[1] + mvd_lX[mbPartIdx][subMbPartIdx][1] (G-92)

    - Otherwise, the following applies.

      refIdxLX = –1 (G-93)
      predFlagLX = 0 (G-94)

    - If predFlagLX != 0, the derived variables mbLX, refIdxLX, and predFlagLX are assigned as follows

mvLX[mbPartIdx][subMbPartIdx][0]	= mvLX[0]	(G-95)
mvLX[mbPartIdx][subMbPartIdx][1]	= mvLX[1]	(G-96)
refIdxLX[mbPartIdx]	= refIdxLX	(G-97)
predFlagLX[mbPartIdx][subMbPartIdx]	= predFlagLX	(G-98)

– The variable MvCnt is incremented by subMvCnt.

G.8.5.1 Derivation process for luma vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8

This process is invoked when mb_type is equal to B_Skip or B_Direct_16x16, or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1.

Outputs of this process are the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1.

From here to the end of this subclause, the entries in the reference picture lists refPicList0 and refPicList1 specify the reference pictures and motion data arrays with refDepencencyId equal to dependency_id. This also applies to all invoked processes.

The derivation process depends on the values of direct_spatial_mv_pred_flag and slice_type, which are present in the bitstream in the slice header syntax as specified in subclause G.7.3.3, and is specified as follows.

– If direct_spatial_mv_pred_flag is equal to 1 and slice_type is not equal to EB (B slice in scalable extension), the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode.

– Otherwise, if direct_spatial_mv_pred_flag is equal to 1 and slice_type is equal to EB (B slice in scalable extension), the mode in which the outputs of this process are derived is referred to as spatial direct prediction mode in scalable extension.

– Otherwise (direct_spatial_mv_pred_flag is equal to 0), the mode in which the outputs of this process are derived is referred to as temporal direct prediction mode.

The spatial direct prediction mode and the temporal direct prediction mode use the co-located motion vectors and reference indices as specified in subclause 8.4.1.2.1 with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the picture colPic, the co-located macroblock mbAddrCol, the motion vector mvCol, the reference index refIdxCol, and the variable vertMvScale as output.

The motion vectors and reference indices are derived as follows.

– If direct_spatial_mv_pred_flag is equal to 1 and slice_type is not equal to EB (B slice in scalable extension), the spatial direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.2 is used, with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output. [Ed.(MW): is proposed change in G.8.4.1.2.2 of JVT-Q202-AnnexG needed?]

– Otherwise, if direct_spatial_mv_pred_flag is equal to 1 and slice_type is equal to EB (B slice in scalable extension), the spatial direct motion vector and reference index prediction mode in scalable extension specified in subclause G.8.5.1.1 is used, with mbPartIdx, subMbPartIdx as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, the sub-partition motion vector count subMvCnt, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output.

– Otherwise (direct_spatial_mv_pred_flag is equal to 0), the direct motion vector and reference index prediction mode specified in subclause 8.4.1.2.3 is used with mbPartIdx, subMbPartIdx, and the reference picture lists refPicList0 and refPicList1 as input and the reference indices refIdxL0, refIdxL1, the motion vectors mvL0 and mvL1, and the prediction list utilization flags, predFlagL0 and predFlagL1 as output and the variable subMvCnt is derived as follows.

If subMbPartIdx is equal to 0, subMvCnt is set equal to 2.

Otherwise (subMbPartIdx is not equal to 0), subMvCnt is set equal to 0.

G.8.5.1.1 Derivation process for spatial direct luma motion vector and reference index prediction mode in scalable extension

This process is invoked when direct_spatial_mv_pred_flag is equal to 1, the syntax element slice_type is equal to EB (B slice in scalable extension), and any of the following conditions is true.

– mb_type is equal to B_Skip

– mb_type is equal to B_Direct_16x16

– sub_mb_type[mbPartIdx] is equal to B_Direct_8x8.

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx.

Outputs of this process are

– the reference indices refIdxL0 and refIdxL1,

– the motion vectors mvL0 and mvL1,

– the sub-partition motion vector count subMvCnt,

– the prediction utilization flags, predFlagL0 and predFlagL1.

The reference indices refIdxL0 and refIdxL1 are derived as follows. [Ed. Note(HS): It may be beneficial to replace the derivation process for the reference indices with a process similar to that of sec. S.8.4.1.2.1 (normal spatial direct mode).]

$$refIdxL0 = 0, \tag{G-99}$$
$$refIdxL1 = 0. \tag{G-100}$$

The motion vectors mvLX (with X being 0 or 1) are derived as follows.

– If sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx, subMbPartIdx = 0, refIdxLX, and list suffix LX as inputs and the output is assigned to mvLX.

– Otherwise, the process specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxLX, and list suffix LX as inputs and the output is assigned to mvLX.

NOTE – In the first case, the returned motion vector mvLX is identical for all four 4x4 sub-macroblock partitions of a sub-macroblock, while in the second case, the returned motion vector mvLX is identical for all sixteen 4x4 sub-macroblock partitions of a macroblock.

The prediction utilization flags predFlagLX (with X being 0 or 1) are derived as follows.

– If refIdxLX is less than 0, predFlagLX is set equal to 0.

– Otherwise (refIdxLX is equal to or greater than 0), predFlagLX is set equal to 1.

The variable subMvCnt is derived as follows.

– If subMbPartIdx is equal to 0 or direct_8x8_inference_flag is equal to 0, subMvCnt is set equal to predFlagL0 + predFlagL1.

– Otherwise (subMbPartIdx is not equal to 0 and direct_8x8_inference_flag is equal to 1), subMvCnt is set equal to 0.

G.8.6 Resampling process for motion data

Inputs to this process are

– a macroblock address mbAddr,

– a variable mbType specifying the macroblock type for the macroblock mbAddr,

– a list subMbType of 4 elements specifying the sub-macroblock types for the macroblock mbAddr (if applicable),

- two arrays of reference indices refIdxL0 and refIdxL1,
- two arrays of luma motion vectors mvL0 and mvL1,
- reference picture lists refPicList0 and refPicList1,
- an array mbTypeBase specifying the macroblock types for the base macroblocks,
- an array subMbTypeBase containing lists of 4 elements specifying the sub-macroblock types for the base macroblocks,
- two arrays predFlagL0Base and predFlagL1Base containing arrays of prediction utilization flags for each base macroblock,
- two arrays refIdxL0Base and refIdxL1Base containing arrays of reference indices for each base macroblock,
- two arrays mvL0Base and mvL1Base containing arrays of luma motion vectors for each base macroblock,

Output of this process are

- a modified version of mbType and subMbType,
- modified versions of the two arrays of reference indices refIdxL0 and refIdxL1,
- modified versions of the two arrays of luma motion vectors mvL0 and mvL1.

The derivation process for macroblock type and sub-macroblock type in inter-layer prediction as specified in subclause G.6.4 is invoked with mbAddr, mbTypeBase, subMbTypeBase, predFlagL0Base, predFlagL1Base as inputs and the outputs are assigned to mbType, subMbType, and a 4x4 array partInfo.

The variable mbType shall not be marked as not available after invoking the process in subclause G.6.4.

Let baseHeight and scaledHeight be temporary variables derived as follows.

- If SpatialScalabilityType is lower than or equal to 3, baseHeight is set equal to BasePicHeight and scaledHeight is set equal to ScaledBaseHeight.
- Otherwise (SpatialScalabilityType is greater than 3), the following applies.
  - baseHeight is derived as follows.
    - Let baseProgFrameFlag be a flag derived as follows.
      - If one of the following conditions is true, baseProgFrameFlag is set equal to 1,
        - BaseFrameMbsOnlyFlag is equal to 1,
        - BaseMbaffFrameFlag is equal to 0 and BaseFieldPicFlag is equal to 0.
      - Otherwise, baseProgFrameFlag is set equal to 0.
    - The variable baseHeight is derived as follows.
      - If one of the following conditions is true, baseHeight is set equal to BasePicHeight / 2.
        - macroblock mbAddr is a field macroblock and BaseMbaffFrameFlag is equal to 1,
        - macroblock mbAddr is a field macroblock, baseProgFrameFlag is equal to 1, andScaledBaseHeight is lower than (2 * BasePicHeight),
      - Otherwise, baseHeight is set equal to BasePicHeight.
  - scaledHeight is derived as follows.
    - If MbaffFrameFlag is equal to 1 and macroblock mbAddr is a field macroblock, scaledHeight is set equal to ScaledBaseHeight / 2.
    - Otherwise (MbaffFrameFlag is equal to 0 or macroblock mbAddr is a frame macroblock), scaledHeight is set equal to ScaledBaseHeight.

The macroblock partition index mbPartIdx proceeds over values 0..NumMbPart(mbType) – 1, and for each value of mbPartIdx, the following applies.

- Let numSubMbPart be a variable, which is set equal to NumSubMbPart(subMbType[mbPartIdx]).

- Let tmpRefIdxL0 and tmpRefIdxL1 be two arrays of numSubMbPart elements.
- The sub-macroblock partition index proceeds over values 0..numSubMbPart – 1, and for each value of subMbPartIdx the following applies.
  - The inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with subMbPartIdx and the output is assigned to (xS, yS).
  - mbAddrBase, mbPartIdxBase and subMbPartIdxBase are derived as follows.

```
mbAddrBase        =  partInfo[ xS / 4, yS / 4 ] / 16                    (G-101)
mbPartIdxBase     = ( partInfo[ xS / 4, yS / 4 ] % 16 ) / 4             (G-102)
subMbPartIdxBase =  partInfo[ xS / 4, yS / 4 ] % 4                      (G-103)
```

  - For X being replaced by either 0 or 1, the variables tmpRefIdxLX[subMbPartIdx] and mvLX[mbPartIdx][subMbPartIdx] are derived as follows.
    - If predFlagLXBase[mbAddrBase][mbPartIdxBase] is equal 0,

```
tmpRefIdxLX[ subMbPartIdx ]                  = –1                       (G-104)
mvLX       [ mbPartIdx ][ subMbPartIdx ][ 0 ] = 0                       (G-105)
mvLX       [ mbPartIdx ][ subMbPartIdx ][ 1 ] = 0                       (G-106)
```

    - Otherwise (predFlagLXBase[mbAddrBase][mbPartIdxBase] is equal to 1), the following applies.
      - The variable tmpRefIdxLX[subMbPartIdx] is derived as follows
        - If mbAddr is a frame macroblock and mbAddrBase is a field macroblock,

```
tmpRefIdxLX[ subMbPartIdx ] = refIdxLXBase[ mbAddrBase ][ mbPartIdxBase ] / 2      (G-107)
```

        - Otherwise, if mbAddr is a field macroblock and mbAddrBase is a frame macroblock,

```
tmpRefIdxLX[ subMbPartIdx ] = refIdxLXBase[ mbAddrBase ][ mbPartIdxBase ] * 2      (G-108)
```

        - Otherwise,

```
tmpRefIdxLX[ subMbPartIdx ] = refIdxLXBase[ mbAddrBase ][ mbPartIdxBase ]          (G-109)
```

      - Let mv be a temporary array with 2 elements, which is set equal to mvLXBase[mbAddrBase][mbPartIdxBase][subMbPartIdxBase].
      - mv[1] is modified as follows.
        - If all the following conditions are true, mv[1] is multiplied by 2.
          - BaseMbaffFrameFlag is equal to 1,
          - base macroblock mbAddrBase is a field macroblock,
          - mbAddr is a frame macroblock
        - Otherwise, if one of the following conditions is true, mv[1] is divided by 2.
          - BaseFrameMbsOnlyFlag is equal to 0 and macroblock mbAddrBase is a frame macroblock and macroblock mbAddr is a field macroblock,
          - BaseFrameMbsOnlyFlag is equal to 1 and macroblock mbAddr is a field macroblock and FrameHeightInMbs is lower than (2 * BasePicWidthInMbs).
        - Otherwise, mv[1] is not modified.
      - The variable mvLX[mbPartIdx][subMbPartIdx] is derived by

mvLX[mbPartIdx][subMbPartIdx][0] =
(mv[0] * ScaledBaseWidth + sign(mv[0]) * BasePicWidth / 2) / BasePicWidth (G-110)
mvLX[mbPartIdx][subMbPartIdx][1] =
(mv[1] * scaledHeight + sign(mv[1]) * baseHeight / 2) / baseHeight (G-111)

– When SpatialScalabilityType is greater than 2, the following applies.
  – Let refScaledBaseLeftOffset and refScaledBaseTopOffset be the values of the variables ScaledBaseLeftOffset and ScaledBaseTopOffset that have been derived for the picture, which is referenced by the reference index refIdxLX[mbPartIdx].
  – The motion vector mvLX[mbPartIdx][subMbPartIdx] is modified by

mvLX[mbPartIdx][subMbPartIdx][0] +=
4 * (refScaledBaseLeftOffset - ScaledBaseLeftOffset) (G-112)
mvLX[mbPartIdx][subMbPartIdx][1] +=
4 * (refScaledBaseTopOffset - ScaledBaseTopOffset) (G-113)

– For X being replaced by either 0 or 1, refIdxLX[mbPartIdx] is derived as follows.
– If SpatialScalabilityType is less than 3 or numSubMbPart is equal to 1, refIdxLX[mbPartIdx] is derived by

refIdxLX[mbPartIdx] = tmpRefIdxLX[subMbPartIdx] (G-114)

– Otherwise (SpatialScalabilityType is greater than 2 and numSubMbPart is greater than 1), the following applies to ensure that each motion vector of the partition uses the same reference index.
  – Let minRefIdx be a variable, which is set equal to 100.
  – For subMbPartIdx = 0..numSubMbPart – 1, the following applies.
    – When tmpRefIdxLX[subMbPartIdx] is greater than –1 and smaller than minRefIdx,

minRefIdx = tmpRefIdxLX[subMbPartIdx] (G-115)

  – refIdxLX[mbPartIdx] is derived by

refIdxLX[mbPartIdx] = (minRefIdx = = 100 ? 0 : minRefIdx) (G-116)

  – subMbPartIdx proceeds over the values 0..numSubMbPart – 1, and for each subMbPartIdx, for which tmpRefIdxLX[subMbPartIdx] is not equal to refIdxLX[mbPartIdx], the following applies.
    – Let predSubMbIdx be a variable, which is set equal to subMbPartIdx.
    – The variable predSubMbIdx is modified as follows.
      – If tmpRefIdxLX[2 * (subMbPartIdx / 2) + 1 – (subMbPartIdx % 2)] is equal to minRefIdx,

predSubMbIdx = 2 * (subMbPartIdx / 2) + 1 – (subMbPartIdx % 2) (G-117)

      – Otherwise, if numSubMbPart is equal to 4 and tmpRefIdxLX[2 * (1 – subMbPartIdx / 2) + (subMbPartIdx % 2)] is equal to refIdxLX[mbPartIdx] ,

predSubMbIdx = 2 * (1 – subMbPartIdx / 2) + (subMbPartIdx % 2) (G-118)

      – Otherwise, if numSubMbPart is equal to 4 and tmpRefIdxLX[2 * (1 – subMbPartIdx / 2) + 1 - (subMbPartIdx % 2)] is equal to refIdxLX[mbPartIdx] ,

predSubMbIdx = 2 * (1 – subMbPartIdx / 2) + 1 - (subMbPartIdx % 2) (G-119)

  - Otherwise, predSubMbIdx is not modified.
- When predSubMbPartIdx is not equal to subMbPartIdx, the following applies.

mvLX[mbPartIdx][subMbPartIdx][0] = mvLX[mbPartIdx][predSubMbIdx][0] (G-120)
mvLX[mbPartIdx][subMbPartIdx][1] = mvLX[mbPartIdx][predSubMbIdx][1] (G-121)

G.8.7 Inter prediction process

Inputs to this process are

- a macroblock address mbAddr
- a variable lumaTrafo specifying the transformation for the luma component
- a variable mbType specifying the macroblock type for the macroblock mbAddr
- a list subMbType of 4 elements specifying the sub-macroblock types for the macroblock mbAddr (if applicable)
- two arrays predFlagL0 and predFlagL1 of prediction utilization flags
- two arrays of reference indices refIdxL0 and refIdxL1
- two arrays of luma motion vectors mvL0 and mvL1
- reference picture lists refPicList0 and refPicList1
- reference picture lists refPicListBase0 and refPicListBase1 relating to base representations
- a list sigBCoeff of significance indicators for transform coefficient levels of a base slice
- a list of context identifiers ctx4x4Id with 16 elements
- an array of luma sample values $picSample_L$ with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements
- when chroma_format_idc is not equal to 0, two arrays of chroma sample values $picSample_{Cb}$ and $picSample_{Cr}$ with ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) elements
- an array of residual luma sample values $picRes_L$ with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements
- when chroma_format_idc is not equal to 0, two arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$ with ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) elements
- the variables for weighted prediction of luma samples $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L},
- when chroma_format_idc is not equal to 0 (monochrome), the variables for weighted prediction of chroma samples $logWD_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C}.

Outputs of this process are

- a modified version of the array of luma sample values $picSample_L$ relating to luma prediction samples for the current macroblock
- when chroma_format_idc is not equal to 0, modified versions of the two arrays of chroma sample values $picSample_{Cb}$ and $picSample_{Cr}$ relating to chroma prediction samples for the current macroblock

Let $predMb_L$ be a 16x16 array of luma prediction samples for the macroblock mbAddr.

When chroma_format_idc is not equal to 0, let $predMb_{Cb}$ and $predMb_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays of chroma prediction samples for the macroblock mbAddr.

When adaptive_ref_fgs_flag is equal to 1, the following applies.

- Let $diffPredMb_L$ be a 16x16 array of differential luma prediction samples for the macroblock mbAddr.
- When chroma_format_idc is not equal to 0, let $diffPredMb_{Cb}$ and $diffPredMb_{Cr}$ be two (MbWidthC)x(MbHeightC) arrays of differential chroma prediction samples for the macroblock mbAddr.

The variable numMbPart is derived as follows.

- If mbType is equal to B_Skip or B_Direct_16x16, numMbPart is set equal to 4.

– Otherwise, numMbPart is set equal to NumMbPart(mbType).

The macroblock partition index mbPartIdx proceeds over values 0..numMbPart – 1, and for each value of mbPartIdx the following applies.

– The variable numSubMbPart is derived as follows.

  – If mbType is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, numSubMbPart is set equal to 1.

  – Otherwise, if mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is not equal to B_Direct_8x8, numSubMbPart is set equal to NumSubMbPart(subMbType[mbPartIdx]).

  – Otherwise (mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8, numSubMbPart is set equal to 4.

– The sub-macroblock partition index proceeds over values 0..numSubMbPart – 1, and for each value of subMbPartIdx the following applies.

  – The variables partWidth and partHeight are derived as follows

    – If mbType is not equal to P_8x8, P_8x8ref0, B_Skip, B_Direct_16x16, or B_8x8, partWidth and partHeight are derived as

      partWidth = MbPartWidth(mbType) (G-122)
      partHeight = MbPartHeight(mbType) (G-123)

    – Otherwise, if mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is not equal to B_Direct_8x8, partWidth and partHeight are derived as

      partWidth = SubMbPartWidth(subMbType[mbPartIdx]) (G-124)
      partHeight = SubMbPartHeight(subMbType[mbPartIdx]) (G-125)

    – Otherwise (mbType is equal to P_8x8 or P_8x8ref0, or mbType is equal to B_8x8 and subMbType[mbPartIdx] is equal to B_Direct_8x8, partWidth and partHeight are derived as

      partWidth = 4 (G-126)
      partHeight = 4 (G-127)

  – When chroma_format_idc is not equal to 0, the following applies.

    – The variables partWidthC and partHeightC are derived as

      partWidthC = partWidth / SubWidthC (G-128)
      partHeightC = partHeight / SubWidthC (G-129)

    – When predFlagLX (with X being replaced by 0 or 1) is equal to 1, the derivation process for chroma motion vectors as specified in subclause 8.4.1.4 is invoked with mvLX[mbPartIdx][subMbPartIdx] and refIdxLX[mbPartIdx] as the inputs and the mvCLX as the output.

  – Depending on use_base_representation_flag, the following applies.

    – If use_base_representation_flag is equal to 1, the temporary variable refPicListTemp0 and refPicListTemp1 are set equal to refPicListBase0 and refPicListBase1 respectively.

    – Otherwise (use_base_representation_flag is equal to 0), the temporary variable refPicListTemp0 and refPicListTemp1 are set equal to refPicList0 and refPicList1 respectively

  – The decoding process for Inter prediction samples as specified in subclause 8.4.2 is invoked with mbPartIdx, subMbPartIdx, partWidth and partHeight, partWidthC and partHeightC (if available), luma motion vectors mvL0[mbPartIdx][subMbPartIdx] and mvL1[mbPartIdx][subMbPartIdx], chroma motion vectors mvCL0 and mvCL1 (if available), reference indices refIdxL0[mbPartIdx] and refIdxL1[mbPartIdx], prediction utilization flags predFlagL0[mbPartIdx] and predFlagL1[mbPartIdx], reference picture lists refPicListTemp0 and refPicListTemp1, as well as luma prediction weights $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L} and chroma prediction weights $logWD_C$, w_{0C}, w_{1C}, o_{1C}, and o_{0C} as inputs and the outputs are a

(partWidth)x(partHeight) array $predPart_L$ of luma prediction samples and, when chroma_format_idc is not equal to 0, two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$ and $predPart_{Cr}$ of chroma prediction samples.

– When adaptive_ref_fgs_flag is equal to 1, the decoding process for differential Inter prediction samples as specified in subclause G.8.7.1 is invoked with mbPartIdx, subMbPartIdx, partWidth and partHeight, partWidthC and partHeightC (if available), luma motion vectors mvL0[mbPartIdx][subMbPartIdx] and mvL1[mbPartIdx][subMbPartIdx], chroma motion vectors mvCL0 and mvCL1 (if available) reference indices refIdxL0[mbPartIdx] and refIdxL1[mbPartIdx], prediction utilization flags predFlagL0[mbPartIdx] and predFlagL1[mbPartIdx] as well as reference picture lists refPicList0, refPicList1, refPicListBase0, and refPicListBase1 as the inputs and the outputs are a (partWidth)x(partHeight) array $diffPredPart_L$ of differential luma prediction samples and, when chroma_format_idc is not equal to 0, two (partWidthC)x(partHeightC) arrays $diffPredPart_{Cb}$ and $diffPredPart_{Cr}$ of differential chroma prediction samples.

– The inverse macroblock partition scanning process as specified in subclause 6.4.2.1 is invoked with mbPartIdx as the input and (xP, yP) as the output.

– A location (xS, yS) is derived as follows.

  – If mbType is equal to P_8x8, P_8x8ref0, or B_8x8, the inverse sub-macroblock partition scanning process as specified in subclause 6.4.2.2 is invoked with mbPartIdx and subMbPartIdx as the inputs and the outputs is assigned to (xS, yS).

  – Otherwise, (xS, yS) is set equal to (0, 0).

– For x = 0..partWidth – 1 and y = 0..partHeight – 1, the 16x16 array $predMb_L$ is modified by

$$predMb_L[xP + xS + x, yP + yS + y] = predPart_L[x, y] \quad \text{(G-130)}$$

– When adaptive_ref_fgs_flag is equal to 1, for x = 0..partWidth – 1 and y = 0..partHeight – 1, the 16x16 array $diffPredMb_L$ is modified by

$$diffPredMb_L[xP + xS + x, yP + yS + y] = diffPredPart_L[x, y] \quad \text{(G-131)}$$

– When chroma_format_idc is not equal to 0, the following applies.

  – For x = 0..partWidthC – 1 and y = 0..partHeightC – 1, the (MbWidthC)x(MbHeightC) arrays $predMb_{Cb}$ and $predMb_{Cr}$ are modified by

$$predMb_{Cb}[(xP + xS) / SubWidthC + x, (yP + yS) / SubHeightC + y] = predPart_{Cb}[x, y] \quad \text{(G-132)}$$
$$predMb_{Cr}[(xP + xS) / SubWidthC + x, (yP + yS) / SubHeightC + y] = predPart_{Cr}[x, y] \quad \text{(G-133)}$$

  – When adaptive_ref_fgs_flag is equal to 1, for x = 0..partWidthC – 1 and y = 0..partHeightC – 1, the (MbWidthC)x(MbHeightC) arrays $diffPredMb_{Cb}$ and $diffPredMb_{Cr}$ are modified by

$$diffPredMb_{Cb}[(xP + xS) / SubWidthC + x, (yP + yS) / SubHeightC + y] = diffPredPart_{Cb}[x, y] \quad \text{(G-134)}$$
$$diffPredMb_{Cr}[(xP + xS) / SubWidthC + x, (yP + yS) / SubHeightC + y] = diffPredPart_{Cr}[x, y] \quad \text{(G-135)}$$

When adaptive_ref_fgs_flag is equal to 1, the following applies.

– The scaling process for differential Inter prediction samples as specified in subclause G.8.7.2 is invoked with lumaTrafo, sigBCoeff, ctx4x4Id, $diffPredMb_L$, and, when chroma_format_idc is not equal to 0, $diffPredMb_{Cb}$ and $diffPredMb_{Cr}$ as the inputs and the outputs are a modified array $diffPredMb_L$, and, when chroma_format_idc is not equal to 0, modified arrays $diffPredMb_{Cb}$ and $diffPredMb_{Cr}$.

– The array $predMb_L$ is modified by

$$predMb_L[x, y] = Clip1_Y(predMb_L[x, y] + diffPredMb_L[x, y]) \quad \text{with } x, y = 0..15 \quad \text{(G-136)}$$

– When chroma_format_idc is not equal to 0, the arrays $predMb_C$ with C being replaced by Cb or Cr are modified by

$$predMb_C[x, y] = Clip1_C(predMb_C[x, y] + diffPredMb_C[x, y]) \quad \text{with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \quad \text{(G-137)}$$

When smoothed_reference_flag is equal to 1, the smoothed reference sample prediction process in subclause G.8.7.3 is invoked with $predMb_L$, and, when chroma_format_idc is not equal to 0, $predMb_{Cb}$, $predMb_{Cr}$, and $picRes_L$, and, when

chroma_format_idc is not equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as inputs, and $predMb_L$, and, when chroma_format_idc is not equal to 0, $predMb_{Cb}$, and $predMb_{Cr}$ as output.

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $predMb_L$, $picSample_L$ and, when chroma_format_idc is not equal to 0, $predMb_{Cb}$, $predMB_{Cr}$, $picSample_{Cb}$, and $picSample_{Cr}$ as inputs and the outputs are a modified version of $picSample_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picSample_{Cb}$, and $picSample_{Cr}$.

G.8.7.1 Decoding process for differential Inter prediction samples

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available)

– luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags, predFlagL0 and predFlagL1

– reference picture lists refPicList0 and refPicList1 (if available)

– reference picture lists refPicListBase0 and refPicListBase1 (if available) relating to base representations

Outputs of this process are

– the differential Inter prediction samples diffPredPart, which are a (partWidth)x(partHeight) array $diffPredPart_L$ of differential luma prediction samples, and when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays $diffPredPart_{Cb}$, $diffPredPart_{Cr}$ of differential chroma prediction samples, one for each of the chroma components Cb and Cr.

Let $diffPredPartL0_L$ and $diffPredPartL1_L$ be (partWidth)x(partHeight) arrays of predicted luma sample values and when chroma_format_idc is not equal to 0 (monochrome) $diffPredPartL0_{Cb}$, $diffPredPartL1_{Cb}$, $diffPredPartL0_{Cr}$, and $difPredPartL1_{Cr}$ be (partWidthC)x(partHeightC) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, refPicListX, refPicListBaseX, refIdxLX, refPicLX, diffPredPartLX, the following is specified.

When predFlagLX is equal to 1, the following applies.

– The differential reference picture consisting of an ordered two-dimensional array $diffRefPicLX_L$ of luma samples and when chroma_format_idc is not equal to 0 (monochrome) two ordered two-dimensional arrays $diffRefPicLX_{Cb}$ and $diffRefPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause G.8.7.1.1 with refIdxLX, refPicListX, and refPicListBaseX given as input.

– The array $diffPredPartLX_L$ and when chroma_format_idc is not equal to 0 (monochrome) the arrays $diffPredPartLX_{Cb}$ and $diffPredPartLX_{Cr}$ are derived by invoking the process specified in subclause G.8.7.1.2 with the current partition specified by mbPartIdx\subMbPartIdx, the variables partWidth, partHeight, partWidthC (if available) and partHeightC (if available) specifying the width and height of the partition, the motion vectors mvLX, mvCLX (if available), and the reference arrays with $diffRefPicLX_L$, $diffRefPicLX_{Cb}$ (if available), and $diffRefPicLX_{Cr}$ (if available) given as input.

For C being replaced by L, Cb (if available), or Cr (if available), the array $diffPredPart_C$ of the prediction samples of component C is derived by invoking the process specified in subclause 8.4.2.3 with the current partition specified by mbPartIdx and subMbPartIdx, the arrays $diffPredPartL0_C$ and $diffPredPartL1_C$, the reference picture lists refPicList0 and refPicList1 as well as the prediction utilization flags predFlagL0 and predFlagL1 given as input.

G.8.7.1.1 Differential reference picture selection process

Inputs to this process are

– a reference index refIdxLX

– a reference picture list refPicListX

– a second reference picture list refPicListBaseX relating to a reference picture list of decoded base pictures

Output of this process is a reference picture consisting of

– an array $diffRefPicLX_L$ of differential luma sample values

– when chroma_format_idc is not equal to 0, two arrays $diffRefPicLX_{Cb}$ and $diffRefPicLX_{Cr}$ of differential chroma sample values

The reference picture selection process as specified in subclause 8.4.2.1 is invoked with refIdxLX and refPicListX as the inputs and the outputs are an array $refPicEL_L$ of luma samples and, when chroma_format_idc is not equal to 0, two arrays $refPicEL_{Cb}$ and $refPicEL_{Cr}$ of chroma sample values.

The reference picture selection process as specified in subclause 8.4.2.1 is invoked with refIdxLX and refPicListBaseX as the inputs and the outputs are an array $refPicBL_L$ of luma samples and, when chroma_format_idc is not equal to 0, two arrays $refPicBL_{Cb}$ and $refPicBL_{Cr}$ of chroma sample values.

The variables $refPicHeightInSamples_L$ and, when chroma_format_idc is not equal to 0, $refPicHeightInSamples_C$ are derived as follows.

– If the field_pic_flag is equal to 0 and the current macroblock is a field macroblock,

$$refPicHeightInSamples_L = PicHeightInSamples_L / 2 \quad \text{(G-138)}$$
$$refPicHeightInSamples_C = PicHeightInSamples_C / 2 \quad \text{(G-139)}$$

– Otherwise (field_pic_flag is equal to 1 or the current macroblock is a frame macroblock)

$$refPicHeightInSamples_L = PicHeightInSamples_L \quad \text{(G-140)}$$
$$refPicHeightInSamples_C = PicHeightInSamples_C \quad \text{(G-141)}$$

The $(PicWidthInSamples_L)x(refPicHeightInSamples_L)$ array $refPicLX_L$, which represents the constructed luma samples of the reference picture that is output of this process, is derived as follows.

– For $x = 0...PicWidthInSamples_L - 1$ and $y = 0..refPicHeightInSamples_L - 1$,

$$diffRefPicLX_L[\ x, y\ ] = refPicEL_L[\ x, y\ ] - refPicBL_L[\ x, y\ ] \quad \text{(G-142)}$$

When chroma_format_idc not equal to 0, the $(PicWidthInSamples_C)x(refPicHeightInSamples_C)$ arrays $refPicLX_{cb}$ and $refPicLX_{cr}$, which represent the constructed chroma samples of the reference picture that is output of this process, are derived as follows.

– For $x = 0...PicWidthInSamples_C - 1$ and $y = 0..refPicHeightInSamples_C - 1$,

$$diffRefPicLX_{cb}[\ x, y\ ] = refPicEL_{Cb}[\ x, y\ ] - refPicBL_{Cb}[\ x, y\ ] \quad \text{(G-143)}$$
$$diffRefPicLX_{cr}[\ x, y\ ] = refPicEL_{Cr}[\ x, y\ ] - refPicBL_{Cr}[\ x, y\ ] \quad \text{(G-144)}$$

G.8.7.1.2 Fractional differential sample interpolation process

[Ed. (HS): This subclause is mostly repeating the content of 8.4.2.2, but different variables names are used and different processes are invoked. Check whether and how we can save redundant text here.]

Inputs to this process are

– the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available)

– a luma motion vector mvLX given in one-quarter luma sample units,

– when chroma_format_idc is not equal to 0, a chroma motion vector mvCLX given in one-eighth chroma sample units, and

– differential reference picture sample arrays $diffRefPicLX_L$ and, when chroma_format_idc is not equal to 0, $diffRefPicLX_{Cb}$, and $diffRefPicLX_{Cb}$

Outputs of this process are

– a (partWidth)x(partHeight) array $diffPredPartLX_L$ of prediction luma sample values and

– two (partWidthC)x(partHeightC) arrays $diffPredPartLX_{Cb}$, and $diffPredPartLX_{Cr}$ of prediction chroma sample values.

Let (xA_L, yA_L) be the location given in full-sample units of the upper-left luma sample of the current partition given by mbPartIdx\subMbPartIdx relative to the upper-left luma sample location of the given two-dimensional array of luma samples.

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in quarter-sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the differential reference sample arrays $diffRefPicLX_L$, $diffRefPicLX_{Cb}$ (if available), and $diffRefPicLX_{Cr}$ (if available).

For each luma sample location (0 <= x_L < partWidth, 0 <= y_L < partHeight) inside the prediction luma sample array $diffPredPartLX_L$, the corresponding predicted luma sample value $diffPredPartLX_L[\ x_L, y_L\ ]$ is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$ and $yFrac_L$ are derived by

$$xInt_L = xA_L + (mvLX[\ 0\ ] >> 2) + x_L \quad \text{(G-145)}$$
$$yInt_L = yA_L + (mvLX[\ 1\ ] >> 2) + y_L \quad \text{(G-146)}$$

$$xFrac_L = mvLX[\ 0\ ]\ \&\ 3 \quad \text{(G-147)}$$
$$yFrac_L = mvLX[\ 1\ ]\ \&\ 3 \quad \text{(G-148)}$$

– The prediction sample value $diffPredPartLX_L[\ x_L, y_L\ ]$ is derived by invoking the process specified in subclause G.8.7.1.3 with ($xInt_L$, $yInt_L$), ($xFrac_L$, $yFrac_L$) and $diffRefPicLX_L$ given as input.

When chroma_format_idc is not equal to 0 (monochrome), the following applies.

– Let ($xInt_C$, $yInt_C$) be a chroma location given in full-sample units and ($xFrac_C$, $yFrac_C$) be an offset given in one-eighth sample units. These variables are used only inside this subclause for specifying general fractional-sample locations inside the reference sample arrays $diffRefPicLX_{Cb}$, and $diffRefPicLX_{Cr}$.

– For each chroma sample location (0 <= x_C < partWidthC, 0 <= y_C < partHeightC) inside the prediction chroma sample arrays $diffPredPartLX_{Cb}$ and $diffPredPartLX_{Cr}$, the corresponding prediction chroma sample values $diffPredPartLX_{Cb}[\ x_C, y_C\ ]$ and $diffPredPartLX_{Cr}[\ x_C, y_C\ ]$ are derived as follows:

  – Depending on chroma_format_idc, the variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived as follows.

    – If chroma_format_idc is equal to 1,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[\ 0\ ] >> 3) + x_C \quad \text{(G-149)}$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[\ 1\ ] >> 3) + y_C \quad \text{(G-150)}$$

$$xFrac_C = mvCLX[\ 0\ ]\ \&\ 7 \quad \text{(G-151)}$$
$$yFrac_C = mvCLX[\ 1\ ]\ \&\ 7 \quad \text{(G-152)}$$

    – Otherwise, if chroma_format_idc is equal to 2,

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[\ 0\ ] >> 3) + x_C \quad \text{(G-153)}$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[\ 1\ ] >> 2) + y_C \quad \text{(G-154)}$$

$$xFrac_C = mvCLX[\ 0\ ]\ \&\ 7 \quad \text{(G-155)}$$
$$yFrac_C = (mvCLX[\ 1\ ]\ \&\ 3) << 1 \quad \text{(G-156)}$$

    – Otherwise (chroma_format_idc is equal to 3),

$$xInt_C = (xA_L / SubWidthC) + (mvCLX[0] >> 2) + x_C \quad \text{(G-157)}$$
$$yInt_C = (yA_L / SubHeightC) + (mvCLX[1] >> 2) + y_C \quad \text{(G-158)}$$

$$xFrac_C = (mvCLX[0] \& 3) << 1 \quad \text{(G-159)}$$
$$yFrac_C = (mvCLX[1] \& 3) << 1 \quad \text{(G-160)}$$

– The prediction sample value $diffPredPartLX_{Cb}[x_C, y_C]$ is derived by invoking the process specified in subclause G.8.7.1.4 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $diffRefPicLX_{Cb}$ given as input.

– The prediction sample value $diffPredPartLX_{Cr}[x_C, y_C]$ is derived by invoking the process specified in subclause G.8.7.1.4 with ($xInt_C$, $yInt_C$), ($xFrac_C$, $yFrac_C$) and $diffRefPicLX_{Cr}$ given as input.

G.8.7.1.3 Fractional differential luma sample interpolation process

Inputs to this process are

– a luma location in full-sample units ($xInt_L$, $yInt_L$),

– a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and

– the luma sample array of the selected differential reference picture $diffRefPicLX_L$

Output of this process is a predicted luma sample value $diffPredPartLX_L[x_L, y_L]$.

The variable $refPicHeightEffective_L$, which is the height of the effective reference picture luma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_L$ is set equal to $PicHeightInSamples_L$ / 2.

To derive the predicted luma sample value $diffPredPartLX_L[x_L, y_L]$, four luma samples located at full-sample locations A_L, B_L, C_L and D_L inside the given two-dimensional array $diffRefPicLX_L$ are utilized. The locations of A_L, B_L, C_L and D_L are specified as follows:

$$xA_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L) \quad \text{(G-161)}$$
$$xB_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L + 1) \quad \text{(G-162)}$$
$$xC_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L) \quad \text{(G-163)}$$
$$xD_L = Clip3(0, PicWidthInSamples_L - 1, xInt_L + 1) \quad \text{(G-164)}$$

$$yA_L = Clip3(0, refPicHeightEffective_L - 1, yInt_L) \quad \text{(G-165)}$$
$$yB_L = Clip3(0, refPicHeightEffective_L - 1, yInt_L) \quad \text{(G-166)}$$
$$yC_L = Clip3(0, refPicHeightEffective_L - 1, yInt_L + 1) \quad \text{(G-167)}$$
$$yD_L = Clip3(0, refPicHeightEffective_L - 1, yInt_L + 1) \quad \text{(G-168)}$$

Given the luma samples A_L, B_L, C_L, and D_L at full-sample locations, the predicted luma sample value $diffPredPartLX_L[x_L, y_L]$ is derived by

$$diffPredPartLX_L[x_L, y_L] = ((4 - xFrac_L) * (4 - yFrac_L) * A_L + xFrac_L * (4 - yFrac_L) * B_L + (4 - xFrac_L) * yFrac_L * C_L + xFrac_L * yFrac_L * D_L + 8) >> 4 \quad \text{(G-169)}$$

G.8.7.1.4 Fractional differential chroma sample interpolation process

Inputs to this process are

– a chroma location in full-sample units ($xInt_C$, $yInt_C$),

– a chroma location offset in fractional-sample units ($xFrac_C$, $yFrac_C$), and

– chroma component samples from the selected differential reference picture $diffRefPicLX_C$.

Output of this process is a predicted chroma sample value $diffPredPartLX_C[\ x_C, y_C\ ]$.

The variable $refPicHeightEffective_C$, which is the height of the effective reference picture chroma array, is derived as follows.

– If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$.

– Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), $refPicHeightEffective_C$ is set equal to $PicHeightInSamples_C$ / 2.

To derive the predicted chroma sample value $diffPredPartLX_C[\ x_C, y_C\ ]$, four chroma samples located at full-sample locations A_C, B_C, C_C and D_C inside the given two-dimensional array $diffRefPicLX_C$ are utilized. The locations of A_C, B_C, C_C and D_C are specified as follows:

$$xA_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C\ ) \quad \text{(G-170)}$$
$$xB_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C + 1\ ) \quad \text{(G-171)}$$
$$xC_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C\ ) \quad \text{(G-172)}$$
$$xD_C = Clip3(\ 0, PicWidthInSamples_C - 1, xInt_C + 1\ ) \quad \text{(G-173)}$$

$$yA_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C\ ) \quad \text{(G-174)}$$
$$yB_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C\ ) \quad \text{(G-175)}$$
$$yC_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C + 1\ ) \quad \text{(G-176)}$$
$$yD_C = Clip3(\ 0, refPicHeightEffective_C - 1, yInt_C + 1\ ) \quad \text{(G-177)}$$

Given the chroma samples A_C, B_C, C_C, and D_C at full-sample locations, the predicted chroma sample value $diffPredPartLX_C[\ x_C, y_C\ ]$ is derived by

$$diffPredPartLX_C[\ x_C, y_C\ ] = (\ (\ 8 - xFrac_C\ ) * (\ 8 - yFrac_C\ ) * A_C + xFrac_C * (\ 8 - yFrac_C\ ) * B_C + (\ 8 - xFrac_C\ ) * yFrac_C * C_C + xFrac_C * yFrac_C * D_C + 32\ ) >> 6 \quad \text{(G-178)}$$

G.8.7.2 Scaling process for differential Inter prediction samples

Inputs to this process are

– a variable lumaTrafo specifying the transformation for the luma component

– a list sigBCoeff of significance indicators for transform coefficient levels of a base slice

– a list of context identifiers ctx4x4Id with 16 elements

– a 16x16 array $diffPred_L$ of differential luma sample values

– when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $diffPred_{Cb}$ and $diffPred_{Cr}$ of differential chroma sample values

Outputs of the this process are

– a modified version of the array $diffPred_L$

– when chroma_format_idc is not equal to 0, modified versions of the arrays $diffPred_{Cb}$ and $diffPred_{Cr}$

Depending on lumaTrafo the following applies.

– If lumaTrafo is equal to T_4x4, for each 4x4 luma block index by luma4x4BlkIdx = 0..15, the following applies.

  – The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and the output is assigned to (xB, yB).

  – A 4x4 array diffPred4x4 is derived by

$$diffPred4x4[\ x, y\ ] = diffPred_L[\ xB + x, yB + y\ ] \quad \text{with } x, y = 0...3 \quad \text{(G-179)}$$

- The scaling process for differential Inter prediction samples of 4x4 luma blocks as specified in subclause G.8.7.2.1 is invoked with luma4x4BlkIdx, sigBCoeff, ctx4x4Id[luma4x4BlkIdx], and diffPred4x4 as the inputs and the output is a modified version of the array diffPred4x4.
- The array $diffPred_L$ is modified by

$$diffPred_L[xB + x, yB + y] = diffPred4x4[x, y] \quad \text{with } x, y = 0...3 \qquad \text{(G-180)}$$

- Otherwise (lumaTrafo is equal to T_8x8), for each 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following applies.
  - The inverse 8x8 luma block scanning process as specified in subclause 6.4.4 is invoked with luma8x8BlkIdx as the input and the output is assigned to (xB, yB).
  - An 8x8 array diffPred8x8 is derived by

$$diffPred8x8[x, y] = diffPred_L[xB + x, yB + y] \quad \text{with } x, y = 0...7 \qquad \text{(G-181)}$$

  - The scaling process for differential Inter prediction samples of 8x8 luma blocks as specified in subclause G.8.7.2.2 is invoked with luma4x4BlkIdx, sigBCoeff, and diffPred8x8 as the inputs and the output is a modified version of the array diffPred8x8.
  - The array $diffPred_L$ is modified by

$$diffPred_L[xB + x, yB + y] = diffPred8x8[x, y] \quad \text{with } x, y = 0...7 \qquad \text{(G-182)}$$

When chroma_format_idc is not equal to 0, the following applies.

- The scaling process for differential Inter prediction samples of chroma blocks as specified in subclause G.8.7.2.3 is invoked with iCbCr equal to 0, sigBCoeff, and $diffPred_{Cb}$ as the inputs and the output is a modified version of $diffPred_{Cb}$.
- The scaling process for differential Inter prediction samples of chroma blocks as specified in subclause G.8.7.2.3 is invoked with iCbCr equal to 1, sigBCoeff, and $diffPred_{Cr}$ as the inputs and the output is a modified version of $diffPred_{Cr}$.

G.8.7.2.1 Scaling process for differential Inter prediction samples of 4x4 luma blocks

Inputs to this process are

- a 4x4 luma block index luma4x4BlkIdx
- a list sigBCoeff of significance indicators for transform coefficient levels of a base slice
- a context identifier ctx4x4Id
- a 4x4 array diffPred4x4 of differential luma samples

Output of this process is a modified version of the 4x4 array diffPred4x4 of differential luma samples.

Let sBC be a 4x4 array of significance indicators for luma transform coefficient levels, which is derived by

$$sBC[x, y] = sigBCoeff[16 * luma4x4BlkIdx + 4 * y + x] \quad \text{with } x, y = 0...3 \qquad \text{(G-183)}$$

Let numBaseSig be the number of values equal to 1 inside the 4x4 array sBC.

Depending on numBaseSig the following applies.

- If numBaseSig is equal to 0, the following applies.
  - A scaling factor sF is derived as follows.
    - If ctx4x4Id is equal to 0, sF is set equal to MaxDiffRefScaleZeroBaseBlock.
    - Otherwise (ctx4x4Id is not equal to 0), sF is set equal to max(0, MaxDiffRefScaleZeroBaseBlock – 4).
  - The 4x4 array diffPred4x4 of differential luma prediction samples is modified by

$$diffPred4x4[x, y] = (sF * diffPred4x4[x, y] + 16) >> 5 \quad \text{with } x, y = 0...3 \qquad \text{(G-184)}$$

– Otherwise (numBaseSig is not equal to 0), the following applies.

  – A scaling factor sF is derived as follows.

    – If numBaseSig is equal to 0, sF is set equal to MaxDiffRefScaleForBaseCoeff.

    – Otherwise, if numBaseSig is larger than 4, sF is set equal to 0.

    – Otherwise (numBaseSig is in the range of 1 to 4, inclusive), sF is derived by

$$tmp = (MaxDiffRefScaleZeroBaseBlock * (numBaseSig - 1) + 2) >> 2$$
$$sF = (tmp == 1 ? 0 : tmp) \quad \text{(G-185)}$$

  – The forward 4x4 transform process as specified in subclause G.8.7.2.1.2 is invoked with the 4x4 array diffPred4x4 as the input and the output is a 4x4 array diffCoeff4x4 of transform coefficient values.

  – Each element diffCoeff4x4[x, y] with x, y = 0..3 of the array of transform coefficient values diffCoeff4x4 is modified by

$$diffCoeff4x4[x, y] = (1 - sBC[x, y]) * ((sF * diffCoeff4x4[x, y] + 16) >> 5) \quad \text{(G-186)}$$

  – The normalization process for 4x4 block of transform coefficient values as specified in subclause G.8.7.2.1.3 is invoked with diffCoeff4x4 and specialDCNormalization equal to 0 as the inputs and the output is a modified version of diffCoeff4x4.

  – The transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with diffCoeff4x4 as the input and the output is assigned to diffPred4x4.

G.8.7.2.1.1 Forward 4-point one dimensional transform process

The input to this process is a one-dimensional array d with elements d_i, i, = 0..3.

Output of this process are transform coefficients as a one-dimensional array f with elements f_i, i, = 0..3.

A set of intermediate values is computed as follows.

$$e_0 = d_0 + d_3 \quad \text{(G-187)}$$
$$e_1 = d_1 + d_2 \quad \text{(G-188)}$$
$$e_2 = d_0 - d_3 \quad \text{(G-189)}$$
$$e_3 = d_1 - d_2 \quad \text{(G-190)}$$

Then, the transformed result is computed from these intermediate values as follows.

$$f_0 = e_0 + e_1 \quad \text{(G-191)}$$
$$f_1 = e_0 - e_1 \quad \text{(G-192)}$$
$$f_2 = (e_2 << 1) + e_3 \quad \text{(G-193)}$$
$$f_3 = e_2 - (e_3 << 1) \quad \text{(G-194)}$$

G.8.7.2.1.2 Forward 4x4 transform process on differential Inter prediction samples

Input to this process is a 4x4 array s with elements s_{ij}, i, j = 0..3.

Outputs of this process are transform coefficients as 4x4 array c with elements c_{ij}, i, j = 0..3.

First, each (horizontal) row of 4x4 array, s_{ij}, with j = 0..3, is transformed using a one-dimensional transform as specified in G.8.7.2.1.1, with output assigned to a row in an intermediate 4x4 array, t_{ij}, with j = 0..3.

Then, each (vertical) column of the resulting matrix, t_{ij}, with i = 0..3, is transformed using the same one-dimensional transform as specified in G.8.7.2.1.1, with output assigned to a column in an 4x4 array, c_{ij}, with i = 0..3.

G.8.7.2.1.3 Normalization process for 4x4 blocks of transform coefficient values

Inputs to this process are

– a 4x4 array c with elements c_{ij}, i, j = 0..3

– a variable specialDCNormalization

Output of this process is a modified version of the 4x4 array c.

A 4x4 matrix norm4x4 is defined by

$$\text{norm4x4} = \begin{bmatrix} 512 & 410 & 512 & 410 \\ 410 & 328 & 410 & 328 \\ 512 & 410 & 512 & 410 \\ 410 & 328 & 410 & 328 \end{bmatrix} \quad \text{(G-195)}$$

Depending on specialDCNormalization the following applies.

– If specialDCNormalization is equal to 0, the elements c_{ij} with i, j = 0..3 are modified by

$$c_{ij} = (c_{ij} * \text{norm4x4}_{ij} + 64) >> 7 \quad \text{(G-196)}$$

– Otherwise (specialDCNormalization is equal to 1), the following applies.

  – The transform coefficient value c_{00} is modified by

$$c_{00} = (c_{00} * \text{norm4x4}_{00} + 128) >> 8 \quad \text{(G-197)}$$

  – The transform coefficient values c_{ij} with i, j = 0..3 and i != j are modified by

$$c_{ij} = (c_{ij} * \text{norm4x4}_{ij} + 64) >> 7 \quad \text{(G-198)}$$

G.8.7.2.2 Scaling process for differential Inter prediction samples of 8x8 luma blocks

Inputs to this process are

– a 8x8 luma block index luma8x8BlkIdx

– a list sigBCoeff of significance indicators for transform coefficient levels of a base slice

– an 8x8 array diffPred8x8 of differential luma samples

Output of this process is a modified version of the 8x8 array diffPred8x8 of differential luma samples.

Let sBC be a 8x8 array of significance indicators for luma transform coefficient levels, which is derived by

sBC[x, y] = sigBCoeff[64 * luma8x8BlkIdx + 8 * y + x] with x, y = 0...7 (G-199)

Let numBaseSig be the number of values equal to 1 inside the 8x8 array sBC.

Depending on numBaseSig the following applies.

– If numBaseSig is equal to 0, the following applies.

  – A scaling factor sF is set equal to MaxDiffRefScaleZeroBaseBlock.

  – The 8x8 array diffPred8x8 of differential luma prediction samples is modified by

diffPred8x8[x, y] = (sF * diffPred8x8[x, y] + 16) >> 5 with x, y = 0...7 (G-200)

– Otherwise (numBaseSig is not equal to 0), the following applies.

  – A scaling factor sF is set equal to MaxDiffRefScaleZeroBaseCoeff.

  – The forward 8x8 transform process as specified in subclause G.8.7.2.2.2 is invoked with the 8x8 array diffPred8x8 as the input and the output is a 8x8 array diffCoeff8x8 of transform coefficient values.

  – Each element diffCoeff8x8[x, y] with x, y = 0..8 of the array of transform coefficient values diffCoeff8x8 is modified by

diffCoeff8x8[x, y] = (1 – sBC[x, y]) * ((sF * diffCoeff8x8[x, y] + 16) >> 5) (G-201)

– The normalization process for 8x8 blocks of transform coefficient values as specified in subclause G.8.7.2.2.3 is invoked with diffCoeff8x8 as the input and the output is a modified version of diffCoeff4x4.

– The transformation process for residual 8x8 blocks as specified in subclause 8.5.11.2 is invoked with diffCoeff8x8 as the input and the output is assigned to diffPred8x8.

G.8.7.2.2.1 Forward 8-point one dimensional transform process

The input to this process is a one-dimensional array d with elements d_i, i, = 0..7.

Output of this process are transform coefficients as a one-dimensional array f with elements g_i, i, = 0..7.

A set of intermediate values is computed as follows.

$$e_0 = d_0 + d_7 \quad \text{(G-202)}$$
$$e_1 = d_1 + d_6 \quad \text{(G-203)}$$
$$e_2 = d_2 + d_5 \quad \text{(G-204)}$$
$$e_3 = d_3 + d_4 \quad \text{(G-205)}$$
$$e_4 = d_0 - d_7 \quad \text{(G-206)}$$
$$e_5 = d_1 - d_6 \quad \text{(G-207)}$$
$$e_6 = d_2 - d_5 \quad \text{(G-208)}$$
$$e_7 = d_3 - d_4 \quad \text{(G-209)}$$

A second set of intermediate results f_i is computed from the intermediate values e_i as

$$f_0 = e_0 + e_3 \quad \text{(G-210)}$$
$$f_1 = e_1 + e_2 \quad \text{(G-211)}$$
$$f_2 = e_0 - e_3 \quad \text{(G-212)}$$
$$f_3 = e_1 - e_2 \quad \text{(G-213)}$$
$$f_4 = e_5 + e_6 + ((e_4 >> 1) + e_4) \quad \text{(G-214)}$$
$$f_5 = e_4 - e_7 - ((e_6 >> 1) + e_6) \quad \text{(G-215)}$$
$$f_6 = e_4 + e_7 - ((e_5 >> 1) + e_5) \quad \text{(G-216)}$$
$$f_7 = e_5 - e_6 + ((e_7 >> 1) + e_7) \quad \text{(G-217)}$$

Then, the transformed result g_i is computed from these intermediate values f_i as

$$g_0 = f_0 + f_1 \quad \text{(G-218)}$$
$$g_1 = f_4 + (f_7 >> 2) \quad \text{(G-219)}$$
$$g_2 = f_2 + (f_3 >> 1) \quad \text{(G-220)}$$
$$g_3 = f_5 + (f_6 >> 2) \quad \text{(G-221)}$$
$$g_4 = f_0 - f_1 \quad \text{(G-222)}$$
$$g_5 = f_6 - (f_5 >> 2) \quad \text{(G-223)}$$
$$g_6 = (f_2 >> 1) - f_3 \quad \text{(G-224)}$$
$$g_7 = (f_4 >> 2) - f_7 \quad \text{(G-225)}$$

G.8.7.2.2.2 Forward 8x8 transform process on differential Inter luma prediction samples

Input to this process is a 8x8 array s with elements s_{ij}, i, j = 0..7.

Outputs of this process are transform coefficients as 4x4 array c with elements c_{ij}, i, j = 0..7.

First, each (horizontal) row of 8x8 array, s_{ij}, with j = 0..7, is transformed using a one-dimensional transform as specified in G.8.7.2.2.1, with output assigned to a row in an intermediate 8x8 array, t_{ij}, with j = 0..7.

Then, each (vertical) column of the resulting matrix, t_{ij}, with i = 0..7, is transformed using the same one-dimensional transform as specified in G.8.7.2.2.1, with output assigned to a column in an 8x8 array, c_{ij}, with i = 0..7.

G.8.7.2.2.3 Normalization process for 8x8 blocks of transform coefficient values

Input to this process is an 8x8 array c with elements c_{ij}, i, j = 0..7.

Output of this process is a modified version of the 4x4 array c.

An 8x8 matrix norm8x8 is defined by

$$\text{norm8x8} = \begin{bmatrix} 512 & 454 & 819 & 454 & 512 & 454 & 819 & 454 \\ 454 & 402 & 726 & 402 & 454 & 402 & 726 & 402 \\ 819 & 726 & 1311 & 726 & 819 & 726 & 1311 & 726 \\ 454 & 402 & 726 & 402 & 454 & 402 & 726 & 402 \\ 512 & 454 & 819 & 454 & 512 & 454 & 819 & 454 \\ 454 & 402 & 726 & 402 & 454 & 402 & 726 & 402 \\ 819 & 726 & 1311 & 726 & 819 & 726 & 1311 & 726 \\ 454 & 402 & 726 & 402 & 454 & 402 & 726 & 402 \end{bmatrix} \quad \text{(G-226)}$$

The elements c_{ij} with i, j = 0..7 are modified by

$$c_{ij} = (c_{ij} * \text{norm8x8}_{ij} + 256) >> 9 \quad \text{(G-227)}$$

G.8.7.2.3 Scaling process for differential Inter prediction samples of chroma blocks

Inputs to this process are

– a chroma component index iCbCr

– a list sigBCoeff of significance indicators for transform coefficient levels of a base slice

– an (MbWidthC)x(MbHeightC) array diffPred of differential chroma sample values

Output of this process is a modified version of the (MbWidthC)x(MbHeightC) array diffPred of differential chroma sample values.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

Let the variable cO be set equal to 256 + iCbCr * MbWidthC * MbHeightC.

Let sBC be a list of numChroma4x4Blks 4x4 arrays of significance indicators and let diffPred4x4 be a list of numChroma4x4Blks 4x4 arrays of differential chroma samples values. For each chroma 4x4 block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the variables sBC and diffPred4x4 are derived as follows

– Depending on chroma_format_idc, a location offset (xB, yB) is derived as follows

  – If chroma_format_idc is equal to 1 or 2,

    xB = 4 * (chroma4x4BlkIdx % 2) (G-228)
    yB = 4 * (chroma4x4BlkIdx / 2) (G-229)

  – Otherwise (chroma_format_idc is equal to 3), the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with chroma4x4BlkIdx and the output is assigned to (xB, yB).

– For x, y = 0..3, the 4x4 arrays sBC[chroma4x4BlkIdx] and diffPredC4x4[chroma4x4BlkIdx] are derived by

    sBC[chroma4x4BlkIdx][x, y] = sigBCoeff[cO + 16 * chroma4x4BlkIdx + 4 * y + x] (G-230)

    diffPred4x4[chroma4x4BlkIdx][x, y] = diffPred[xB + x, yB + y] (G-231)

Let numBaseSigDC be the number of values diffPred4x4[chroma4x4BlkIdx][0, 0] that are equal to 1 for chroma4x4BlkIdx = 0..numChroma4x4Blks – 1.

Depending on numBaseSigDC the following applies.

– If numBaseSigDC is equal to 0, for each 4x4 chroma block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1 the following applies.

  – Let numBaseSigAC be the number of values equal to 1 inside the 4x4 array sBC[chroma4x4BlkIdx].

  – Depending on numBaseSigAC the following applies.

    – If numBaseSigAC is equal to 0, the following applies.

- A scaling factor sF is set equal to MaxDiffRefScaleZeroBaseBlock.
- The 4x4 array diffPred4x4[chroma4x4BlkIdx] is modified by

  diffPred4x4[chroma4x4BlkIdx][x, y] =
  (sF * diffPred4x4[chroma4x4BlkIdx][x, y] + 16) >> 5 with x, y = 0...3 (G-232)

- Otherwise (numBaseSigAC is not equal to 0), the following applies.
  - A scaling factor sF is derived as follows.
    - If numBaseSigAC is larger than 4, sF is set equal to 0.
    - Otherwise (numBaseSigAC is in the range of 1 to 4, inclusive), sF is derived by

      tmp = (MaxDiffRefScaleZeroBaseBlock * (numBaseSigAC – 1) + 2) >> 2
      sF = (tmp = = 1 ? 0 : tmp) (G-233)

  - The forward 4x4 transform process as specified in subclause G.8.7.2.1.2 is invoked with the 4x4 array diffPred4x4[chroma4x4BlkIdx] as the input and the output is a 4x4 array diffCoeff4x4 of transform coefficient values.
  - Each element diffCoeff4x4[x, y] with x, y = 0..3 of the array of transform coefficient values diffCoeff4x4 is modified by

    diffCoeff4x4[x, y] = (1 – sBC[chroma4x4BlkIdx][x, y]) *
    ((sF * diffCoeff4x4[x, y] + 16) >> 5) (G-234)

  - The normalization process for 4x4 block of transform coefficient values as specified in subclause G.8.7.2.1.3 is invoked with diffCoeff4x4 and specialDCNormalization equal to 0 as the inputs and the output is a modified version of diffCoeff4x4.
  - The transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with diffCoeff4x4 as the input and the output is assigned to diffPred4x4[chroma4x4BlkIdx].

- Otherwise (numBaseSigDC is greater than 0), the following applies.
  - Let diffCoeff4x4 be a list of numChroma4x4Blks 4x4 arrays of transform coefficient values.
  - For each 4x4 chroma block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the forward 4x4 transform process as specified in subclause G.8.7.2.1.2 is invoked with the 4x4 array diffPred4x4[chroma4x4BlkIdx] as the input and the output is assigned to the 4x4 array diffCoeff4x4[chroma4x4BlkIdx].
  - The process specified in subclause G.8.7.2.3.1 is invoked with diffCoeff4x4 as the input and the output is an (MbWidthC / 4)x(MbHeightC / 4) array dcCoeff of chroma DC transform coefficient values.
  - The transformation process for chroma DC coefficients as specified in subclause 8.5.9.1 is invoked with dcCoeff as the input and the output is again assigned to dcCoeff.
  - The process specified in subclause G.8.7.2.3.2 is invoked with dcCoeff and diffCoeff4x4 as the inputs and the output is a modified version of diffCoeff4x4.
  - For each 4x4 chroma block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the following applies.
    - A scaling factor sF is set equal to MaxDiffRefScaleZeroBaseCoeff.
    - Each element diffCoeff4x4[chroma4x4BlkIdx][x, y] with x, y = 0..3 of the 4x4 array of transform coefficient values diffCoeff4x4[chroma4x4BlkIdx] is modified by

      diffCoeff4x4[chroma4x4BlkIdx][x, y] = (1 – sBC[chroma4x4BlkIdx][x, y]) *
      ((sF * diffCoeff4x4[chroma4x4BlkIdx][x, y] + 16) >> 5) (G-235)

    - The normalization process for 4x4 block of transform coefficient values as specified in subclause G.8.7.2.1.3 is invoked with diffCoeff4x4[chroma4x4BlkIdx] and specialDCNormalization equal to 1 as the inputs and the output is a modified version of diffCoeff4x4[chroma4x4BlkIdx].

- The process specified in subclause G.8.7.2.3.1 is invoked with diffCoeff4x4 as the input and the output is an (MbWidthC / 4)x(MbHeightC / 4) array dcCoeff of chroma DC transform coefficient values.
- The transformation process for chroma DC coefficients as specified in subclause 8.5.9.1 is invoked with dcCoeff as the input and the output is again assigned to dcCoeff.
- Let lShift be a variable that is set equal to chroma_format_idc.
- The elements dcCoeff[x, y] with x = 0..(MbWidthC / 4) – 1 and y = 0..(MbHeightC / 4) – 1 of the array dcCoeff are modified by

  dcCoeff[x, y] = (dcCoeff[x, y] + ((1 << lShift) >> 1)) >> lShift (G-236)

- The process specified in subclause G.8.7.2.3.2 is invoked with dcCoeff and diffCoeff4x4 as the inputs and the output is a modified version of diffCoeff4x4.
- For each 4x4 chroma block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with diffCoeff4x4[chroma4x4BlkIdx] as the input and the output is assigned to diffPred4x4[chroma4x4BlkIdx].

For each 4x4 chroma block with chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the arrays diffPred is modified as follows.

- Depending on chroma_format_idc, a location offset (xB, yB) is derived as follows
  - If chroma_format_idc is equal to 1 or 2,

    xB = 4 * (chroma4x4BlkIdx % 2) (G-237)
    yB = 4 * (chroma4x4BlkIdx / 2) (G-238)

  - Otherwise (chroma_format_idc is equal to 3), the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with chroma4x4BlkIdx and the output is assigned to (xB, yB).
- For x, y = 0..3, the arrays diffPred is modified by

  diffPred[xB + x, yB + y] = diffPred4x4[chroma4x4BlkIdx][x, y] (G-239)

G.8.7.2.3.1 Assignment process for a matrix of chroma DC transform coefficient values

Input to this process is a list diffCoeff4x4 of ((MbWidthC / 4) * (MbHeightC / 4)) 4x4 arrays of chroma transform coefficient values.

Output of this process is an (MbWidthC / 4)x(MbHeightC / 4) array of chroma DC transform coefficient values.

Depending on chroma_format_idc, the array dcCoeff is derived as follows.

- If chroma_format_idc is equal to 1 or 2,

  dcCoeff[x, y] = diffCoeff4x4[2 * y + x][0, 0] with x = 0..1, y = 0..(MbHeightC / 4) – 1 (G-240)

- Otherwise (chroma_format_idc is equal to 3),

  dcCoeff[x, y] = diffCoeff4x4[8 * (y / 2) + 4 * (x/ 2) + 2 * (y % 2) + (x % 2)][0, 0]

  with x = 0..(MbWidthC / 4) – 1, y = 0..(MbHeightC / 4) – 1 (G-241)

G.8.7.2.3.2 Construction process for chroma DC coefficient values

Inputs to this process are

- an (MbWidthC / 4)x(MbHeightC / 4) array of chroma DC transform coefficient values.
- a list diffCoeff4x4 of ((MbWidthC / 4) * (MbHeightC / 4)) 4x4 arrays of chroma transform coefficient values.

Output of this process is a modified version of diffCoeff4x4.

Depending on chroma_format_idc, the array diffCoeff4x4 is modified as follows

– If chroma_format_idc is equal to 1 or 2,

$$\text{diffCoeff4x4}[\ 2 * y + x\ ][\ 0, 0\ ] = \text{dcCoeff}[\ x, y\ ] \quad \text{with } x = 0..1,\ y = 0..(\ \text{MbHeightC} / 4\ ) - 1 \qquad \text{(G-242)}$$

– Otherwise (chroma_format_idc is equal to 3),

$$\text{diffCoeff4x4}[\ 8 * (\ y / 2\ ) + 4 * (\ x/ 2\ ) + 2 * (\ y \% 2\ ) + (\ x \% 2\ )\ ][\ 0, 0\ ] = \text{dcCoeff}[\ x, y\ ]$$

$$\text{with } x = 0..(\ \text{MbWidthC} / 4\ ) - 1,\ y = 0..(\ \text{MbHeightC} / 4\ ) - 1 \qquad \text{(G-243)}$$

G.8.7.3 Decoding process for smoothed reference inter prediction samples

This process is invoked when smoothed_reference_flag is equal to 1.

Input to this process is the luma prediction samples, $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, $\text{predMb}_{Cr}[\ x, y\ ]$, picRes_L, and, if chroma_format_idc is not equal to 0, picRes_{Cb} and picRes_{Cr}.

Outputs of this process are the modified luma prediction samples, $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, and $\text{predMb}_{Cr}[\ x, y\ ]$.

Smoothed reference sample prediction process is applied as follows.

– The macroblock sample extraction process as specified in subclause G.8.11.7.2 is invoked with mbAddr, picRes_L, and picRes_{Cb} and picRes_{Cr} (if available) as the inputs and mbResBase_L, and mbResBase_{Cb} and mbResBase_{Cr} (if available) as output.

– Each luma prediction sample $\text{pred}_L[\ x, y\ ]$ with x = 0..15 and y = 0..15 is derived by

$$\text{predMb}_L[\ x, y\ ] = \text{predMb}_L[\ x, y\ ] + \text{mbResBase}_L[\ x, y\ ] \qquad \text{(G-244)}$$

– When chroma_format_idc is not equal to 0, each inter prediction chroma sample $\text{pred}_{Cb}[\ x, y\ ]$ and $\text{pred}_{Cr}[\ x, y\ ]$ with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$$\text{predMb}_{Cb}[\ x, y\ ] = \text{predMb}_{Cb}[\ x, y\ ] + \text{mbResBase}_{Cb}[\ x, y\ ] \qquad \text{(G-245)}$$
$$\text{predMb}_{Cr}[\ x, y\ ] = \text{predMb}_{Cr}[\ x, y\ ] + \text{mbResBase}_{Cr}[\ x, y\ ] \qquad \text{(G-246)}$$

– The smoothing process in subclause G.8.7.3.1 is invoked with $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, $\text{predMb}_{Cr}[\ x, y\ ]$, picRes_L, picRes_{Cb}, and picRes_{Cr} as input and $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, $\text{predMb}_{Cr}[\ x, y\ ]$, as output.

– Each luma prediction sample $\text{predMb}_L[\ x, y\ ]$ with x = 0..15 and y = 0..15 is derived by

$$\text{predMb}_L[\ x, y\ ] = \text{predMb}_L[\ x, y\ ] - \text{mbResBase}_L[\ x, y\ ]$$

– When chroma_format_idc is not equal to 0, each inter prediction chroma sample $\text{predMb}_{Cb}[\ x, y\ ]$ and $\text{predMb}_{Cr}[\ x, y\ ]$ with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$$\text{predMb}_{Cb}[\ x, y\ ] = \text{predMb}_{Cb}[\ x, y\ ] - \text{mbResBase}_{Cb}[\ x, y\ ] \qquad \text{(G-247)}$$
$$\text{predMb}_{Cr}[\ x, y\ ] = \text{predMb}_{Cr}[\ x, y\ ] - \text{mbResBase}_{Cr}[\ x, y\ ] \qquad \text{(G-248)}$$

G.8.7.3.1 Smoothing process for smoothed reference inter prediction samples

This process can only be invoked when the value of the syntax element smoothed_reference_flag is equal to 1.

Inputs to this process are the inter prediction sample arrays, $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, $\text{predMb}_{Cr}[\ x, y\ ]$, picRes_L, picRes_{Cb} and picRes_{Cr}.

Outputs of this process are the modified inter prediction sample arrays, $\text{predMb}_L[\ x, y\ ]$, $\text{predMb}_{Cb}[\ x, y\ ]$, and $\text{predMb}_{Cr}[\ x, y\ ]$.

The horizontal smoothing process is applied to all inside samples as follows.

– For x = 1..14 and y = 0..15

$$\text{predMb}_L[\ x, y\ ] = (\ \text{predMb}_L[\ x - 1, y] + 2 * \text{predMb}_L[\ x, y\ ] + \text{predMb}_L[\ x + 1, y\ ] + 2\ ) >> 2 \qquad \text{(G-249)}$$

– When chroma_format_idc is not equal to 0, for x = 1..(MbWidthC – 2) and y = 0..(MbWidthC – 1)

$$predMb_{Cb}[x, y] = (predMb_{Cb}[x – 1, y] + 2 * predMb_{Cb}[x, y] + predMb_{Cb}[x + 1, y] + 2) >>2 \quad \text{(G-250)}$$
$$predMb_{Cr}[x, y] = (predMb_{Cr}[x – 1, y] + 2 * predMb_{Cr}[x, y] + predMb_{Cr}[x + 1, y] + 2) >>2 \quad \text{(G-251)}$$

The vertical smoothing process is applied to all inside samples as follows.

– For x = 0..15 and y = 1..14

$$predMb_L[x, y] = (predMb_L[x, y – 1] + 2 * predMb_L[x, y] + predMb_L[x, y + 1] + 2) >> 2 \quad \text{(G-252)}$$

– When chroma_format_idc is not equal to 0, for x = 0..(MbWidthC – 1) and y = 1..(MbWidthC – 2)

$$predMb_{Cb}[x, y] = (predMb_{Cb}[x, y – 1] + 2 * predMb_{Cb}[x, y] + predMb_{Cb}[x, y + 1] + 2) >> 2 \quad \text{(G-253)}$$
$$predMb_{Cr}[x, y] = (predMb_{Cr}[x, y – 1] + 2 * predMb_{Cr}[x, y] + predMb_{Cr}[x, y + 1] + 2) >> 2 \quad \text{(G-254)}$$

The position of the upper-left luma sample of the current macroblock currMbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with currMbAddr as input and the output being assigned to (xP, yP). When chroma_format_idc is not equal to 0, the position (xC, yC) of the upper-left chroma samples of the current macroblock are derived by

$$xC = xP / SubWidthC \quad \text{(G-255)}$$
$$yC = yP / SubHeightC \quad \text{(G-256)}$$

The horizontal smoothing process for the left edge of the current macroblock is applied as follows.

– For x = 0 and y = 0..15

$$predMb_L[x, y] = (picRes_L[xP + x – 1, yP + y] + 2 * predMb_L[x, y] + predMb_L[x + 1, y] + 2) >> 2 \quad \text{(G-257)}$$

– When chroma_format_idc is not equal to 0, for x = 0 and y = 0..(MbWidthC – 1)

$$predMb_{Cb}[x, y] = (picRes_{Cb}[xC + x – 1, yC + y] + 2 * predMb_{Cb}[x, y] + predMb_{Cb}[x + 1, y] + 2) >> 2 \quad \text{(G-258)}$$
$$predMb_{Cr}[x, y] = (picRes_{Cr}[xC + x – 1, yC + y] + 2 * predMb_{Cr}[x, y] + predMb_{Cr}[x + 1, y] + 2) >> 2 \quad \text{(G-259)}$$

The vertical smoothing process for the top edge of the current macroblock is applied as follows.

– For x = 0..15 and y = 0

$$predMb_L[x, y] = (picRes_L[xP + x, yP + y – 1] + 2 * predMb_L[x, y] + predMb_L[x, y + 1] + 2) >> 2 \quad \text{(G-260)}$$

– When chroma_format_idc is not equal to 0, for x = 0 and y = 0..(MbWidthC – 1)

$$predMb_{Cb}[x, y] = (picRes_{Cb}[xC + x, yC + y – 1] + 2 * predMb_{Cb}[x, y] + predMb_{Cb}[x, y + 1] + 2) >> 2 \quad \text{(G-261)}$$
$$predMb_{Cr}[x, y] = (picRes_{Cr}[xC + x, yC + y – 1] + 2 * predMb_{Cr}[x, y] + predMb_{Cr}[x, y + 1] + 2) >> 2 \quad \text{(G-262)}$$

G.8.7.4 Derivation process for prediction weights

Inputs to this process are

– the partition index mbPartIdx

- the sub-macroblock partition index subMbPartIdx
- reference picture lists refPicList0 and refPicList1
- prediction list utilization flags, predFlagL0 and predFlagL1

Outputs of this process are

- variables for weighted prediction of luma samples $logWD_L$, w_{0L}, w_{1L}, o_{1L}, o_{0L}
- when chroma_format_idc is not equal to 0 (monochrome), variables for weighted prediction of chroma samples $logWD_C$, w_{0C}, w_{1C}, o_{1C}, o_{0C}.

The output variables of this process are derived as follows.

- If base_pred_weight_table_flag is equal to 0, the process defined in subclause 8.4.2.3 is invoked
- otherwise if base_pred_weight_table_flag is equal to 1 the following process is applied.

For X being replaced by either 0 or 1 the following applies:

- if the variable setOfBaseSlices is available,
  - Let baseLumaLogWD, aBaseLumaWeightLX[], aBaseLumaOffsetLX[], baseChromaLogWD, aBaseChromaWeightLX[][], aBaseChormaOffsetLX[][], be defined as the values of the syntax elements of luma_log2_weight_denom, luma_weight_lX[], luma_offset_lX[],chroma_log2_weight_denom, chroma_weight_lX[] and chroma_offset_lX[] respectively of the covered base slices.
  - For the reference index refIdxLXWP in the current slice reference index list refPicListX, its associated weighting parameters are derived as follows
    - Let refPic be the picture that is referenced by refIdxLXWP
    - Let refPicBase, the reference picture of the corresponding base layer, be considered to exist if there is a picture for which all of the following conditions are true
      - The syntax element dependency_id for the picture refPicBase is equal to the variable DependencyIdBase of the picture refPic.
      - The syntax element quality_level for the picture refPicBase is equal to the variable QualityLevelBase of the the picture refPic.
      - The syntax element fragment_order for the picture refPicBase is equal to the variable FragmentOrderBase of the the picture refPic.
      - The value of PicOrderCnt(refPic) is equal to the value of PicOrderCnt(refPicBase).
      - There is an index baseRefIdxLX equal to the lowest valued reference index in the corresponding base layer reference index list BaseRefPicListX that references refPicBase.
    - If a refPicBase was found to exist the following applies:
      - If C in set equal to L for luma samples

$$logWD_C = baseLumaLogWD \quad \text{(G-263)}$$

$$w_{XC} = aBaseLumaWeightLX[baseRefIdxLX] \quad \text{(G-264)}$$

$$o_{XC} = aBaseLumaOffsetLX[baseRefIdxLX] * (1 << (BitDepth_Y - 8)) \quad \text{(G-265)}$$

      - Otherwise (C in set equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

$$logWD_C = baseChromaLogWD \quad \text{(G-266)}$$

w_{XC} = aBaseChromaWeightLX[baseRefIdxLX][iCbCr] (G-267)

o_{XC} = aBaseChromaOffsetLX[baseRefIdxLX][iCbCr] (G-268)

– Otherwise,
  – If C equal to L for luma samples

    $logWD_C$ = baseLumaLogWD (G-269)

    w_{XC} = 1 << baseLumaLogWD (G-270)

    o_{XC} = 0 (G-271)

  – Otherwise (C set equal to Cb or Cr for chroma samples, with iCbCr = 0 for Cb, iCbCr = 1 for Cr),

    $logWD_C$ = baseChromaLogWD (G-272)

    w_{XC} = 1 << baseChromaLogWD (G-273)

    o_{XC} = 0 (G-274)

– Otherwise,
  – Set baseLumaLogWD equal to 5.
  – Set baseChromaLogWD equal to 5.
  – Apply equations $logWD_C$ = baseLumaLogWD (G-269) to o_{XC} = 0 (G-274).

G.8.8 Intra macroblock decoding process

Inputs to this process are

– variables fieldMb, mbType, subMbType, and lumaTrafo
– syntax elements for a macroblock
– an array intra4x4PredMode specifying Intra_4x4 prediction modes for macroblocks coded in Intra_4x4 macroblock prediction mode
– an array intra8x8PredMode specifying Intra_8x8 prediction modes for macroblocks coded in Intra_8x8 macroblock prediction mode
– an array of luma sample values $picSample_L$ with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements relating to constructed luma samples prior to deblocking filter process for neighbouring macroblocks
– when chroma_format_idc is not equal to 0, two arrays of chroma sample values $picSample_{Cb}$ and $picSample_{Cr}$ with ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) elements relating to a constructed chroma samples prior to deblocking filter process for neighbouring macroblocks

Outputs of this process are

– a modified version of the array intra4x4PredMode
– a modified version of the array intra8x8PredMode
– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements
– a modified version of the array of luma sample values $picSample_L$ relating to constructed luma samples prior to deblocking filter process for the current macroblock

– when chroma_format_idc is not equal to 0, modified versions of the two arrays of chroma sample values $picSample_{Cb}$ and $picSample_{Cr}$ relating to constructed chroma samples prior to deblocking filter process for the current macroblock

The transform coefficient scaling process as specified in subclause G.8.11.2 is invoked with lumaTrafo as input and the output is assigned to sTCoeff.

The macroblock residual construction process as specified in subclause G.8.11.5.1 is invoked with lumaTrafo and sTCoeff as the inputs and the outputs are a 16x16 array of luma residual samples $mbRes_L$ and, when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays of chroma residual samples $mbRes_{Cb}$ and $mbRes_{Cr}$.

Depending on mbType the following applies.

– If mbType is equal to I_PCM, the sample construction process for I_PCM macroblocks as specified in subclause 8.3.5 is invoked with $picSample_L$ and, when chroma_format_idc is not equal to 0, $picSample_{Cb}$ and $picSample_{Cr}$ as inputs and the outputs are a possibly modified version of $picSample_L$ and, when chroma_format_idc is not equal to 0, possibly modified versions of $picSample_{Cb}$ and $picSample_{Cr}$.

– Otherwise (mbType is not equal to I_PCM), the following applies.

  – The constructed luma samples are derived as follows.

    – If MbPartPredMode(mbType, 0) is equal to Intra_4x4, the Intra_4x4 prediction and construction process for luma samples as specified in subclause G.8.8.1 is invoked with $mbRes_L$, $picSamples_L$, intra4x4PredMode, and intra8x8PredMode as the inputs and the outputs are modified versions of intra4x4PredMode and $picSamples_L$.

    – Otherwise, if MbPartPredMode(mbType, 0) is equal to Intra_8x8, the Intra_8x8 prediction and construction process for luma samples as specified in subclause G.8.8.2 is invoked with $mbRes_L$, $picSamples_L$, intra4x4PredMode, and intra8x8PredMode as the inputs and the outputs are modified versions of intra8x8PredMode and $picSamples_L$.

    – Otherwise (MbPartPredMode(mbType, 0) is equal to Intra_16x16), the Intra_16x16 prediction and construction process as specified in subclause G.8.8.3 is invoked with $mbRes_L$ and $picSamples_L$ as the inputs and the output is a modified version of $picSamples_L$.

  – When chroma_format_idc is not equal to 0, the intra prediction and construction process for chroma samples as specified in subclause G.8.8.4 is invoked with $mbRes_{Cb}$, $mbRes_{Cr}$, $picSample_{Cb}$, and $picSample_{Cr}$ as the inputs and the outputs are modified versions of $picSample_{Cb}$ and $picSample_{Cr}$.

All (256 + 2 * MbWidthC * MbHeightC) elements of the list of scaled transform coefficient values sTCoeff are set equal to 0.

G.8.8.1 Intra_4x4 prediction and construction process for luma samples

Inputs to this process are

– an 16x16 array $mbRes_L$ of residual luma samples for the current macroblock

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma sample values relating to constructed luma samples prior to the deblocking filter process of neighbouring macroblocks

– an array intra4x4PredMode specifying Intra_4x4 prediction modes for macroblocks coded in Intra_4x4 macroblock prediction mode

– an array intra8x8PredMode specifying Intra_8x8 prediction modes for macroblocks coded in Intra_8x8 macroblock prediction mode

Outputs of this process are

– a modified version of the array intra4x4PredMode

– a modified version of the array $picSamples_L$

Let $mbPred_L$ be a 16x16 array of intra prediction samples for the current macroblock, and let $mbSamples_L$ be a 16x16 array of constructed intra samples for the current macroblock.

The 4x4 luma blocks index by luma4x4BlkIdx = 0..15 are processed in increasing order of luma4x4BlkIdx. For each 4x4 luma block the following applies.

– The derivation process for neighbouring 4x4 luma blocks as specified in subclause 6.4.8.3 is invoked with luma4x4BlkIdx as input and the output is assigned to mbAddrA, luma4x4BlkIdxA, mbAddrB, and luma4x4BlkIdxB.

– The derivation process for Intra_4x4 prediction modes as specified in subclause 8.3.1.1.1 is invoked with mbAddrA, mbAddrB, luma4x4BlkIdxA, luma4x4BlkIdxB, intra4x4PredMode[mbAddrA], intra4x4PredMode[mbAddrB], intra8x8PredMode[mbAddrA], and intra8x8PredMode[mbAddrB] as inputs and the output is assigned to intra4x4PredMode[CurrMbAddr][luma4x4BlkIdx].

– The Intra_4x4 sample prediction process as specified in subclause 8.3.1.2 is invoked with luma4x4BlkIdx, intra4x4PredMode[CurrMbAddr], and the array $picSamples_L$ as inputs and the output is a 4x4 array of luma prediction samples $pred4x4_L$.

– The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (xP, yP) as the output.

– The 16x16 array $mbPred_L$ is modified by

$$mbPred_L[\ xP + x, yP + y\ ] = pred4x4_L[\ x, y\ ] \quad \text{with } x, y = 0..3 \qquad \text{(G-275)}$$

– The values of the 16x16 array $mbSamples_L$ are derived by

$$mbSamples_L[\ x,\ y\ ] = Clip1_Y(\ mbPred_L[\ x, y\ ] + mbRes_L[\ x, y\ ]\ ) \quad \text{with } x, y = 0..15 \qquad \text{(G-276)}$$

– The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with CurrMbAddr, mbW equal to 16, mbH equal to 16, $mbSamples_L$, and $picSamples_L$ as the inputs and the output is a modified version of the array $picSamples_L$.

NOTE – The array $mbSamples_L$ contains constructed luma samples only for 4x4 luma blocks with luma4x4BlkIdx less than the current value of luma4x4BlkIdx. The sample values for 4x4 blocks with a greater value of luma4x4BlkIdx do not influence the decoding process for intra 4x4 blocks. After the last 4x4 block with luma4x4BlkIdx equal to 15 has been processed, all entries of $mbSamples_L$ represent constructed luma samples prior to the deblocking filter process, and these samples for the current macroblock are placed into the array $picSamples_L$.

G.8.8.2 Intra_8x8 prediction and construction process for luma samples

Inputs to this process are

– an 16x16 array $mbRes_L$ of residual luma samples for the current macroblock

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma sample values relating to constructed luma samples prior to the deblocking filter process of neighbouring macroblocks

– an array intra4x4PredMode specifying Intra_4x4 prediction modes for macroblocks coded in Intra_4x4 macroblock prediction mode

– an array intra8x8PredMode specifying Intra_8x8 prediction modes for macroblocks coded in Intra_8x8 macroblock prediction mode

Outputs of this process are

– a modified version of the array intra8x8PredMode

– a modified version of the array $picSamples_L$

Let $mbPred_L$ be a 16x16 array of intra prediction samples for the current macroblock, and let $mbSamples_L$ be a 16x16 array of constructed intra samples for the current macroblock.

The 8x8 luma blocks index by luma8x8BlkIdx = 0..3 are processed in increasing order of luma8x8BlkIdx. For each 8x8 luma block the following applies.

– The derivation process for neighbouring 8x8 luma blocks as specified in subclause 6.4.8.2 is invoked with luma8x8BlkIdx as input and the output is assigned to mbAddrA, luma8x8BlkIdxA, mbAddrB, and luma8x8BlkIdxB.

– The derivation process for Intra_8x8 prediction modes as specified in subclause 8.3.2.1.1 is invoked with mbAddrA, mbAddrB, luma8x8BlkIdxA, luma8x8BlkIdxB, intra4x4PredMode[mbAddrA], intra4x4PredMode[mbAddrB], intra8x8PredMode[mbAddrA], and intra8x8PredMode[mbAddrB] as inputs and the output is assigned to intra8x8PredMode[CurrMbAddr][luma8x8BlkIdx].

– The Intra_8x8 sample prediction process as specified in subclause 8.3.2.2 is invoked with luma8x8BlkIdx, intra8x8PredMode[CurrMbAddr], and the array $picSamples_L$ as inputs and the output is an 8x8 array of luma prediction samples $pred8x8_L$.

– The inverse 4x4 luma block scanning process as specified in subclause 6.4.4 is invoked with luma8x8BlkIdx as the input and (xP, yP) as the output.

– The 16x16 array $mbPred_L$ is modified by

$$mbPred_L[\ xP + x, yP + y\ ] = pred8x8_L[\ x, y\ ] \quad \text{with } x, y = 0..7 \qquad \text{(G-277)}$$

– The values of the 16x16 array $mbSamples_L$ are derived by

$$mbSamples_L[\ x,\ y\ ] = Clip1_Y(\ mbPred_L[\ x, y\ ] + mbRes_L[\ x, y\ ]\ ) \quad \text{with } x, y = 0..15 \qquad \text{(G-278)}$$

– The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with CurrMbAddr, mbW equal to 16, mbH equal to 16, $mbSamples_L$, and $picSamples_L$ as the inputs and the output is a modified version of the array $picSamples_L$.

NOTE – The array $mbSamples_L$ contains constructed luma samples only for 8x8 luma blocks with luma8x8BlkIdx less than the current value of luma8x8BlkIdx. The sample values for 8x8 blocks with a greater value of luma8x8BlkIdx do not influence the decoding process for intra 8x8 blocks. After the last 8x8 block with luma8x8BlkIdx equal to 3 has been processed, all entries of $mbSamples_L$ represent constructed luma samples prior to the deblocking filter process, and these samples for the current macroblock are placed into the array $picSamples_L$.

G.8.8.3 Intra_16x16 prediction and construction process for luma samples

Inputs to this process are

– an 16x16 array $mbRes_L$ of residual luma samples for the current macroblock

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma sample values relating to constructed luma samples prior to the deblocking filter process of neighbouring macroblocks

Outputs of this process is a modified version of the array $picSamples_L$.

The Intra_16x16 prediction process for luma samples as specified in subclause 8.3.3 is invoked with $picSamples_L$ as the input and the output is a 16x16 array $mbPred_L$ of luma prediction samples.

Let $mbSamples_L$ be a 16x16 array of constructed intra samples for the current macroblock, the values of the 16x16 array $mbSamples_L$ are derived by

$$mbSamples_L[\ x,\ y\ ] = Clip1_Y(\ mbPred_L[\ x, y\ ] + mbRes_L[\ x, y\ ]\ ) \quad \text{with } x, y = 0..15 \qquad \text{(G-279)}$$

The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with CurrMbAddr, mbW equal to 16, mbH equal to 16, $mbSamples_L$, and $picSamples_L$ as the inputs and the output is a modified version of the array $picSamples_L$.

G.8.8.4 Intra prediction and construction process for chroma samples

Inputs to this process are

– two (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ of residual chroma samples for the current macroblock

– two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma sample values relating to constructed chroma samples prior to the deblocking filter process of neighbouring macroblocks

Outputs of this process are modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$.

The Intra prediction process for chroma samples as specified in subclause 8.3.4 is invoked with $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are two (MbWidthC)x(MbHeightC) arrays $mbPred_{Cb}$ and $mbPred_{Cr}$ of chroma prediction samples.

For each chroma component C, with C being replaced by Cb or Cr, the following applies.

– Let $mbSamples_C$ be a (MbWidthC)x(MbHeightC) array of constructed intra chroma samples for the current macroblock, the values of the array $mbSamples_C$ are derived by

$$mbSamples_C[\ x,\ y\ ] = Clip1_C(\ mbPred_C[\ x, y\ ] + mbRes_C[\ x, y\ ]\ )$$

$$\text{with } x = 0..MbWidthC - 1 \text{ and } y = 0..MbHeightC - 1 \qquad \text{(G-280)}$$

– The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with CurrMbAddr, mbW equal to MbWidthC, mbH equal to MbHeightC, $mbSamples_C$, and $picSamples_C$ as the inputs and the output is a modified version of the array $picSamples_C$.

G.8.9 Resampling process for intra samples

Inputs to this process are

– a macroblock address mbAddr

– an array lumaTrafoBase specifying the transforms for the luma component of the base slices

– an array mbTypeBase specifying the macroblock types for the base slices

– an array $picSamplesBase_L$ of luma samples for the base slices

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma samples

– when chroma_format_idc is not equal to 0, two arrays $picSamplesBase_{Cb}$ and $picSamplesBase_{Cr}$ of chroma samples for the base slices

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples

Output of this process are

– a modified version of the array $picSamples_L$ of luma samples

– when chroma_format_idc is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples

Let $pred_L[\ x, y\ ]$, with x = 0..15 and y = 0..15, be a 16x16 array of luma samples

When chroma_format_idc is not equal to 0, let $pred_{Cr}[\ x, y\ ]$ and $pred_{Cb}[\ x, y\ ]$, with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 be two (MbWidthC)x(MbHeightC) arrays of chroma samples.

The following variables are derived based on syntax elements chroma_phase_x_plus1, chroma_phase_y_plus1, base_chroma_phase_x_plus1, base_chroma_phase_y_plus1.

$$scaledBaseChromaPhaseX = chroma_phase_x_plus1 - 1 \qquad \text{(G-281)}$$
$$scaledBaseChromaPhaseY = chroma_phase_y_plus1 - 1 \qquad \text{(G-282)}$$
$$baseChromaPhaseX = base_chroma_phase_x_plus1 - 1 \qquad \text{(G-283)}$$
$$baseChromaPhaseY = base_chroma_phase_y_plus1 - 1 \qquad \text{(G-284)}$$

The position (xP, yP) of the upper-left luma sample of the macroblock mbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr as input and the output is assigned to (xP, yP).

When chroma_format_idc is not equal to 0, the position (xC, yC) of the upper-left chroma samples of the current macroblock are derived by

$$xC = xP / SubWidthC \qquad \text{(G-285)}$$
$$yC = yP / SubHeightC \qquad \text{(G-286)}$$

Positions in base layer sample arrays are computed as follows.

– If SpatialScalabilityType is equal to 0, the following applies.

xB = xP – ScaledBaseLeftOffset (G-287)
yB = yP – ScaledBaseTopOffset (G-288)

– When chroma_format_idc is not equal to 0, let (xCB, yCB) be computed as

xCB = xC – ScaledBaseLeftOffsetC (G-289)
yCB = yC – ScaledBaseTopOffsetC (G-290)

– Otherwise (SpatialScalabilityType is less than or equal to 3), the following applies.

– The variables deltaX and deltaY are derived by

deltaX = ((BasePicWidth << 15) + (ScaledBaseWidth >> 1)) / ScaledBaseWidth (G-291)
deltaY = ((BasePicHeight << 15) + (ScaledBaseHeight >> 1)) / ScaledBaseHeight (G-292)

– The corresponding fractional-sample position (xf, yf) in base layer of (xP, yP) is derived as follows.

xf = ((2 * (xP – ScaledBaseLeftOffset) + 1) * deltaX – 30720) >> 12 (G-293)

yf = ((2 * (yP – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 (G-294)

– The full-sample position (xB, yB) is derived as follows.

xB = xf >> 4 (G-295)
yB = yf >> 4 (G-296)

– The corresponding fractional-sample position (xf1, yf1) in base layer of (xP + 15, yP + 15) is derived as follows.

xf1 = ((2 * (xP + 15 – ScaledBaseLeftOffset) + 1) * deltaX – 30720) >> 12 (G-297)

yf1 = ((2 * (yP + 15 – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 (G-298)

– The full-sample position (xB1, yB1) is derived as follows.

xB1 = (xf1 + 15) >> 4 (G-299)
yB1 = (yf1 + 15) >> 4 (G-300)

– When chroma_format_idc is not equal to 0, the following applies.

– The variables deltaXC and deltaYC are derived by

deltaXC = ((BasePicWidthC << 14) + (ScaledBaseWidthC >> 1)) / ScaledBaseWidthC (G-301)
deltaYC = ((BasePicHeightC << 14) + (ScaledBaseHeightC >> 1)) / ScaledBaseHeightC (G-302)

– The location (xfC, yfC) is derived as follows

xfC = ((((4 * (xC – ScaledBaseLeftOffsetC) + (2 + scaledBaseChromaPhaseX)) * deltaXC)
+ 2048) >> 12) – 4 * (2 + baseChromaPhaseX) (G-303)

yfC = ((((4 * (yC – ScaledBaseTopOffsetC) + (2 + scaledBaseChromaPhaseY)) * deltaYC)
+ 2048) >> 12) – 4 * (2 + baseChromaPhaseY) (G-304)

– The full-sample position (xCB, yCB) is derived by

xCB = xfC >> 4 (G-305)
yCB = yfC >> 4 (G-306)

– The location (xfC1, yfC1) is derived as follows

xfC1 = ((((4 * (xC + MbWidthC – 1 – ScaledBaseLeftOffsetC) +
(2 + scaledBaseChromaPhaseX)) * deltaXC)
+ 2048) >> 12) – 4 * (2 + baseChromaPhaseX) (G-307)

yfC1 = = ((((4 * (yC + MbHeightC – 1 – ScaledBaseTopOffsetC) +
(2 + scaledBaseChromaPhaseY)) * deltaYC)
+ 2048) >> 12) – 4 * (2 + baseChromaPhaseY) (G-308)

– The full-sample position (xCB1, yCB1) is derived as follows.

xCB1 = (xfC1 + 15) >> 4 (G-309)
yCB1 = (yfC1 + 15) >> 4 (G-310)

– Otherwise (SpatialScalabilityType is greater than 3), the following applies.

– Let tempYP be defined as follows.

tempYP = yP * (1 + field_pic_flag) (G-311)

– Let tempBasePicHeight be updated as follows.

tempBasePicHeight = BasePicHeight * (1 + BaseFieldPicFlag) (G-312)

– Let tempScaledBaseHeight be updated as follows.

tempScaledBaseHeight = ScaledBaseHeight * (1 + field_pic_flag) (G-313)

– Let deltaX as specified in Equation G-291 and deltaY be defined as follows.

deltaY = ((tempBasePicHeight << 15) + (tempScaledBaseHeight >> 1)) / tempScaledBaseHeight (G-314)

– The corresponding fractional-sample position (xf, yf) in base layer of (xP, tempYP) are derived as specified in Equation G-293 and as follows .

yf = ((2 * (tempYP – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 (G-315)

– The full-sample position (xB, yB) is derived as specified in Equations G-295 and G-296

– The corresponding fractional-sample position (xf1, yf1) in base layer of (xP + 15 , tempYP + 15) is derived as specified in Equations G-297 as follows,

yf1 = ((2 * (tempYP + 15 – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 (G-316)

– The full-sample position (xB1, yB1) is derived as specified in Equations G-299 and G-300

– When field_pic_flag or mb_field_decoding_flag is equal to 1, yB1 is further updated as follows.

yB1 = yB1 + (yB1 – yB + 1) (G-317)

– When BaseFieldPicFlag is equal to 1, yB and yB1 are further updated as follows.

$$yB = yB >> 1 \quad \text{(G-318)}$$
$$yB1 = (yB1 + 1) >> 1 \quad \text{(G-319)}$$

– When chroma_format_idc is not equal to 0, the following applies.

  – Let tempYC be defined as follows.

$$tempYC = yC * (1 + field_pic_flag) \quad \text{(G-320)}$$

  – Let tempBasePicHeightC be updated as follows.

$$tempBasePicHeightC = BasePicHeightC * (1 + BaseFieldPicFlag) \quad \text{(G-321)}$$

  – Let tempScaledBaseHeightC be updated as follows.

$$tempScaledBaseHeightC = ScaledBaseHeightC * (1 + field_pic_flag) \quad \text{(G-322)}$$

  – Let deltaXC be derived as specified in Equation G-301 and deltaYC be defined as follows

$$deltaYC = ((tempBasePicHeightC << 14) + (tempScaledBaseHeightC >> 1)) / tempScaledBaseHeightC \quad \text{(G-323)}$$

  – Let (xfC, yfC) be defined as specified in Equation G-303 and as follows

$$yfC = ((((4 * (tempYC – ScaledBaseTopOffsetC) + (2 + scaledBaseChromaPhaseY)) * deltaYC) + 2048) >> 12) – 4 * (2 + baseChromaPhaseY) \quad \text{(G-324)}$$

  – The full-sample position (xCB, yCB) is derived as follows.

$$xCB = xfC >> 4 \quad \text{(G-325)}$$
$$yCB = yfC >> 4 \quad \text{(G-326)}$$

  – Let (xfC1, yfC1) be defined as specified in Equation G-307 and as follows

$$yfC1 = ((((4 * (tempYC + MbHeightC – 1 – ScaledBaseTopOffsetC) + (2 + scaledBaseChromaPhaseY)) * deltaYC) + 2048) >> 12) – 4 * (2 + baseChromaPhaseY) \quad \text{(G-327)}$$

  – The full-sample position (xCB1, yCB1) is derived as follows.

$$xCB1 = (xfC1 + 15) >> 4 \quad \text{(G-328)}$$
$$yCB1 = (yfC1 + 15) >> 4 \quad \text{(G-329)}$$

  – When field_pic_flag or mb_field_decoding_flag is equal to 1, yCB1 is further updated as follows.

$$yCB1 = yCB1 + (yCB1 – yCB + 1) \quad \text{(G-330)}$$

  – When BaseFieldPicFlag is equal to 1, yCB and yCB1 are further updated as follows.

$$yCB = yCB >> 1 \quad \text{(G-331)}$$
$$yCB1 = (yCB1 + 1) >> 1 \quad \text{(G-332)}$$

The Intra prediction samples for the macroblock mbAddr are derived as follows.

– If SpatialScalabilityType is equal to 0, the following applies. [Ed. (JR) should use F.8.7.3 for the MBAFF cases]

  – Each Intra prediction luma sample $pred_L[x, y]$ with x = 0..15 and y = 0..15 is derived by

$$pred_L[x, y] = picSamplesBase_L[xB + x, yB + y] \quad \text{(G-333)}$$

  – When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$ with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$$pred_{Cb}[x, y] = picSamplesBase_{Cb}[xCB + x, yCB + y] \quad \text{(G-334)}$$
$$pred_{Cr}[x, y] = picSamplesBase_{Cr}[xCB + x, yCB + y] \quad \text{(G-335)}$$

– Otherwise (SpatialScalabilityType is not equal to 0), the following applies.

  – Let $base_L[x, y]$ with x = –4..(xB1 –xB + 4), y = –8..(yB1 – yB + 7) be a luma sample array

  – When chroma_format_idc is not equal to 0, let $base_{Cb}[x, y]$ and $base_{Cr}[x, y]$ with x = –1..(xCB1 – xCB + 1) and y = –2..(yCB1 – yCB + 2) be two chroma samples arrays.

  – Each luma sample $base_L[x, y]$ with x = –4..(xB1 –xB + 4), y = –8..(yB1 – yB + 7) of the luma sample array is derived as follows.

    – Let the temporary position xT and yT be derived as

$$xT = Clip3(0, BasePicWidth – 1 , xB + x) \quad \text{(G-336)}$$
$$yT = Clip3(0, BasePicHeight – 1 , yB + y) \quad \text{(G-337)}$$

    – Depending on the sample $picSamplesBase_L[xT, yT]$ the following applies.

      – If the sample $picSamplesBase_L[xT, yT]$ does not represent a sample of an I macroblock, the sample $base_L[x, y]$ is marked as “not available for Intra_Base prediction”.

      – Otherwise (the sample $picSamplesBase_L[xT, yT]$ represents a sample of an I macroblock), the sample $base_L[x, y]$ is marked as “available for Intra_Base prediction” and derived by

$$base_L[x, y] = picSamplesBase_L[xT, yT] \quad \text{(G-338)}$$

  – When chroma_format_idc is not equal to 0, each chroma sample $base_{Cb}[x, y]$ and $base_{Cr}[x, y]$ with x = –1..(xCB1 – xCB + 1) and y = –2..(yCB1 – yCB + 2) of the chroma sample arrays is derived as follows.

    – Let the temporary position xCT and yCT be derived as

$$xCT = Clip3(0, BasePicWidthC – 1 , xCB + x) \quad \text{(G-339)}$$
$$yCT = Clip3(0, BasePicHeightC – 1 , yCB + y) \quad \text{(G-340)}$$

    – Depending on the sample $picSamplesBase_{Cr}[xCT, yCT]$ the following applies.

      – If $picSamplesBase_{Cr}[xCT, yCT]$ do not represent samples of an I macroblock, the samples $base_{Cb}[x, y]$ and $base_{Cr}[x, y]$ are marked as “not available for Intra_Base prediction”.

      – Otherwise ($picSamplesBase_{Cr}[xCT, yCT]$ represents a samples of an I macroblock), the samples $base_{Cb}[x, y]$ and $base_{Cr}[x, y]$ are marked as “available for Intra_Base prediction” and derived by

$$base_{Cb}[x, y] = picSamplesBase_{Cb}[xCT, yCT] \quad \text{(G-341)}$$
$$base_{Cr}[x, y] = picSamplesBase_{Cr}[xCT, yCT] \quad \text{(G-342)}$$

  – The construction process for not available or not existing samples in subclause G.8.9.2 is invoked with the sample array $base_L[x, y]$, the sample array size parameters (xB1 – xB, yB1 – yB), the luma sample position offset (xB, yB) and, when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}[x, y]$, $base_{Cr}[x, y]$, the sample array size parameters (xCB1 – xCB, yCB1 – yCB) and the chroma sample position offset (xCB, yCB) as inputs and the outputs are the sample array $base_L[x, y]$ with modified sample values,

and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y] with modified sample values.

– Depending on SpatialScalabilityType the following applies.

  – If SpatialScalabilityType is less than or equal to 3, the upsampling process for Intra_Base prediction in subclause G.8.9.3 is invoked with the macroblock address mbAddr, the sample array $base_L$[x, y], integer luma sample positions in basePic (xB, yB) and (xB1, yB1) of the basePic, and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y], and integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1) of the basePic, as inputs and the outputs are $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

  – Otherwise, if SpatialScalabilityType is equal to 4, the following applies

    – The control parameter botField is derived as follows

      botField = base_frame_and_bottom_field_coincided_flag + bottom_field_flag (G-343)

    – The upsampling process for Intra_Base prediction in subclause G.8.9.4 is invoked with the control parameter botField, the macroblock address mbAddr, the sample array $base_L$[x, y], integer luma sample positions in basePic (xB, yB) and (xB1, yB1), and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y], and integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1), as inputs and the outputs are $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y].

    – Depending on field_pic_flag and mb_field_decoding_flag the following applies.

      – If field_pic_flag or mb_field_decoding_flag is equal to 1, the values of array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], are assigned to $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y], respectively.

        – Each Intra prediction luma sample $pred_L$[x, y] with x = 0..15 and y = 0..15 is derived by

          $pred_L$[x, y] = $fieldPred_L$[x, y] (G-344)

        – When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

          $pred_{Cb}$[x, y] = $fieldPred_{Cb}$[x, y] (G-345)
          $pred_{Cr}$[x, y] = $fieldPred_{Cr}$[x, y] (G-346)

      – Otherwise, the vertical upsampling process for Intra_Base prediction in subclause G.8.9.5 is invoked with the control parameter botField, the sample array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, the sample arrays $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], as inputs and the outputs are $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

– Otherwise, if SpatialScalabilityType is equal to 5, the following applies

  – The control parameter botField is derived as follows

    – If BaseFieldPicFlag is equal to 0

      botField = base_bottom_field_coincided_flag (G-347)

    – Otherwise,

      botField = BaseBottomFieldFlag (G-348)

- The upsampling process for Intra_Base prediction in subclause G.8.9.4 is invoked with the control parameter botField, the sample array $base_L$[x, y], integer luma sample positions in basePic (xB, yB) and (xB1, yB1), and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y], and integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1), as inputs and the outputs are $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y].
- The vertical upsampling process for Intra_Base prediction in subclause G.8.9.5 is invoked with the control parameter botField, the sample array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, the sample arrays $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], as inputs and the outputs are $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

- Otherwise (SpatialScalabilityType is equal to 6), the following applies
  - The control parameter botField is derived as follows
    - If field_pic_flag is equal to 1 [SSUN Note: this implies the base_bottom_field be available.]

      botField = bottom_field_flag (G-349)

    - Otherwise, if BaseFieldPicFlag is equal to 1

      botField = BaseBottomFieldFlag (G-350)

    - Otherwise,

      botField = mb_field_decoding_flag & (CurrMdAddr % 2) (G-351)

  - The upsampling process for Intra_Base prediction in subclause G.8.9.4 is invoked with the control parameter botField, the macroblock address mbAddr, the sample array $base_L$[x, y], integer luma sample positions in basePic (xB, yB) and (xB1, yB1), and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y], and integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1), as inputs and the outputs are $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y].
  - Depending on BaseFieldPicFlag, field_pic_flag, and mb_field_decoding_flag the following applies.
    - If BaseFieldPicFlag, field_pic_flag, and mb_field_decoding_flag are all equal to 0, the following applies
      - The control parameter botField is updated as follows

        botField = 1 (G-352)

      - The upsampling process for Intra_Base prediction in subclause G.8.9.4 is invoked with the control parameter botField, the macroblock address mbAddr, the sample array $base_L$[x, y], integer luma sample positions in basePic (xB, yB) and (xB1, yB1), and when chroma_format_idc is not equal to 0, the sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y], and integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1), as inputs and the outputs are $field2Pred_L$[x, y], and when chroma_format_idc is not equal to 0, $field2Pred_{Cb}$[x, y] and $field2Pred_{Cr}$[x, y].
      - The values of array $fieldPred_L$[x, y] and $field2Pred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y], $fieldPred_{Cr}$[x, y], $field2Pred_{Cb}$[x, y], and $field2Pred_{Cr}$[x, y], are asigned to $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y], respectively.
        - Each Intra prediction luma sample $pred_L$[x, y] with x = 0..15 and y = 0..15 is derived by

          $pred_L$[x, y] = (y%2==0? $fieldPred_L$[x, y >> 1]: $field2Pred_L$[x, y >> 1]) (G-353)

– When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$pred_{Cb}$[x, y] = (y%2==0? $fieldPred_{Cb}$[x, y >> 1]: $field2Pred_{Cb}$[x, y >> 1]) (G-354)
$pred_{Cr}$[x, y] = (y%2==0? $fieldPred_{Cr}$[x, y >> 1]: $field2Pred_{Cr}$[x, y >> 1]) (G-355)

– Otherwise, if field_pic_flag or mb_field_decoding_flag is equal to 1, the values of array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], are asigned to $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y], respectively.

– Each Intra prediction luma sample $pred_L$[x, y] with x = 0..15 and y = 0..15 is derived by

$pred_L$[x, y] = $fieldPred_L$[x, y] (G-356)

– When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$pred_{Cb}$[x, y] = $fieldPred_{Cb}$[x, y] (G-357)
$pred_{Cr}$[x, y] = $fieldPred_{Cr}$[x, y] (G-358)

– Otherwise, the vertical upsampling process for Intra_Base prediction in subclause G.8.9.5 is invoked with the control parameter botField, the sample array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, the sample arrays $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], as inputs and the outputs are $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y].

Table G-8 – 16-phase interpolation filter for luma up-sampling in Intra_Base prediction

phase	interpolation filter coefficients					
	e[-2]	e[-1]	e[0]	e[1]	e[2]	e[3]
0	0	0	32	0	0	0
1	0	-2	32	2	0	0
2	1	-3	31	4	-1	0
3	1	-4	30	6	-1	0
4	1	-4	28	9	-2	0
5	1	-4	27	11	-3	0
6	1	-5	25	14	-3	0
7	1	-5	22	17	-4	1
8	1	-5	20	20	-5	1
9	1	-4	17	22	-5	1
10	0	-3	14	25	-5	1
11	0	-3	11	27	-4	1
12	0	-2	9	28	-4	1
13	0	-1	6	30	-4	1
14	0	-1	4	31	-3	1
15	0	0	2	32	-2	0

[Ed. Note(MW): Table is used in F.8.9.3. Should move there]

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $pred_L$, $picSamples_L$ and, when chroma_format_idc is not equal to 0, $pred_{Cb}$, $pred_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as inputs and the outputs are a modified version of $picSamples_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picSamples_{Cb}$, and $picSamples_{Cr}$

G.8.9.1 Deblocking filter process for Intra_Base prediction

Input to this process are

– syntax elements and global variables derived from a NAL unit

– a set of arrays collectively referred to as baseVars

Output of this process is the modified set of arrays collectively referred to as baseVars.

This process modifies the variables assigned to baseVars using syntax elements and global variables of the NAL unit and the values of the variables assigned to baseVars that are the input.

From here to the end of this subclause, it is assumed that the syntax elements and global variables from the NAL unit currently being decoded are the current syntax elements and global variables and are available. This also applies to all invoked processes and all recursive invoking.

In the following of this subclause, the variables lumaTrafo, cS_L, cS_{Cb}, and cS_{Cr} of the collective term baseVars are referred to as lumaTrafo, $picSamples_L$, $picSamples_{Cb}$ and $picSamples_{Cr}$, respectively.

From here to the end of this subclause and for all invoked processes, the variables FilterOffsetA and FilterOffsetB are replaced by InterlayerFilterOffsetA and InterlayerFilterOffsetA respectively.

[Ed. Note(HS/JR): Instead of repeating subclause 8.7, the subclauses.8.7 and 8.7.1 should be slightly modified and used for this process as well as the "normal" deblocking filter process, the point 5 should be put in a new clause that could be overwritten in SVC. The main problem concerns the loop over the MB and the copy at the end of (.7. A solution would be to create a new subclause witch receive some input picture instead of using global variables.]

[Ed. (JR) this subclause uses the global variables: PicSizeInMbs , PicWidthInMbs, MbaffFrameFlag which are the ones of the slice which is being filtered and not the one that is being decoded…]

For the current macroblock address CurrMbAddr proceeding over values 0..PicSizeInMbs – 1 of the slice represented by the array $picSamples_L$[x, y], the following applies.

1. The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.
2. The variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag are derived as follows.
   - The variable fieldModeMbFlag is derived as follows.
     - If any of the following conditions is true, fieldModeMbFlag is set equal to 1.
       - field_pic_flag is equal to 1
       - MbaffFrameFlag is equal to 1 and the macroblock CurrMbAddr is a field macroblock
     - Otherwise, fieldModeMbFlag is set equal to 0.
   - The variable filterInternalEdgesFlag is derived as follows.
     - If any of the following conditions is true, the variable filterInternalEdgesFlag is set equal to 0.
       - disable_interlayer_deblocking_filter_idc for the slice that contains the macroblock mbAddr is equal to 1
       - the macroblock mbAddr is coded using an Inter macroblock prediction mode.
     - Otherwise the variable filterInternalEdgesFlag is set equal to 1.
   - The variable filterLeftMbEdgeFlag is derived as follows.
     - If any of the following conditions is true, the variable filterLeftMbEdgeFlag is set equal to 0.
       - MbaffFrameFlag is equal to 0 and CurrMbAddr % PicWidthInMbs is equal to 0.
       - MbaffFrameFlag is equal to 1 and (CurrMbAddr >> 1) % PicWidthInMbs is equal to 0
       - disable_ interlayer_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1

- disable_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrA is not available
- the macroblock CurrMbAddr is coded using an Inter macroblock prediction mode
- the macroblock mbAddrA is coded using an Inter macroblock prediction mode

- Otherwise, the variable filterLeftMbEdgeFlag is set equal to 1.

- The variable filterTopMbEdgeFlag is derived as follows.
  - If any of the following conditions is true, the variable filterTopMbEdgeFlag is set equal to 0.
    - MbaffFrameFlag is equal to 0 and CurrMbAddr is less than PicWidthInMbs.
    - MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, and the macroblock CurrMbAddr is a field macroblock.
    - MbaffFrameFlag is equal to 1, (CurrMbAddr >> 1) is less than PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and CurrMbAddr % 2 is equal to 0.
    - disable_ interlayer_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 1
    - disable_ interlayer_deblocking_filter_idc for the slice that contains the macroblock CurrMbAddr is equal to 2 and the macroblock mbAddrB is not available
    - the macroblock CurrMbAddr is coded using an Inter macroblock prediction mode
    - the macroblock mbAddrB is coded using an Inter macroblock prediction mode
  - Otherwise, the variable filterTopMbEdgeFlag is set equal to 1.

3. Given the variables fieldModeMbFlag, filterInternalEdgesFlag, filterLeftMbEdgeFlag and filterTopMbEdgeFlag the deblocking filtering is controlled as follows.
   - When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical luma edge is specified as follows.
     - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (0, k)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
   - When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical luma edges is specified as follows.
     - When lumaTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (4, k)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
     - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (8, k)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
     - When lumaTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (12, k)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
   - When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal luma edge is specified as follows.
     - If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr - 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.
       - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, $(xE_k, yE_k) = (k, 0)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
       - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, $(xE_k, yE_k) = (k, 1)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.

- Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 0)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.

- When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal luma edges is specified as follows.
  - When lumaTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 4)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
  - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 8)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.
  - When lumaTrafo[CurrMbAddr] is not equal to T_8x8, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 0, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 12)$ with k = 0..15, and $picSamples_L$ as input and $picSamples_L$ as output.

- For the filtering of both chroma components with iCbCr = 0 for Cb and iCbCr = 1 for Cr, the following applies.
  - Let $picSamples_C$ be equal to $picSamples_{Cb}$ when iCbCr = 0 and $picSamples_{Cb}$ when iCbCr = 1
  - When filterLeftMbEdgeFlag is equal to 1, the filtering of the left vertical chroma edge is specified as follows.
    - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (0, k)$ with k = 0..MbHeightC 1, and $picSamples_C$ as input and $picSamples_C$ as output.
  - When filterInternalEdgesFlag is equal to 1, the filtering of the internal vertical chroma edge is specified as follows.
    - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (4, k)$ with k = 0..MbHeightC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
    - When chroma_format_idc is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (8, k)$ with k = 0..MbHeightC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
    - When chroma_format_idc is equal to 3, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 1, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (12, k)$ with k = 0..MbHeightC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
  - When filterTopMbEdgeFlag is equal to 1, the filtering of the top horizontal chroma edge is specified as follows.
    - If MbaffFrameFlag is equal to 1, (CurrMbAddr % 2) is equal to 0, CurrMbAddr is greater than or equal to 2 * PicWidthInMbs, the macroblock CurrMbAddr is a frame macroblock, and the macroblock (CurrMbAddr – 2 * PicWidthInMbs + 1) is a field macroblock, the following applies.
      - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
      - The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = 1, $(xE_k, yE_k) = (k, 1)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
    - Otherwise, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 0)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.
  - When filterInternalEdgesFlag is equal to 1, the filtering of the internal horizontal chroma edge is specified as follows.

– The process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 4)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.

– When chroma_format_idc is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 8)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.

– When chroma_format_idc is not equal to 1, the process specified in subclause 8.7.1 is invoked with chromaEdgeFlag = 1, verticalEdgeFlag = 0, fieldModeFilteringFlag = fieldModeMbFlag, $(xE_k, yE_k) = (k, 12)$ with k = 0..MbWidthC - 1, and $picSamples_C$ as input and $picSamples_C$ as output.

NOTE 2 – When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied across the top horizontal edges of a frame macroblock, this vertical filtering across the top or bottom macroblock boundary may involve some samples that extend across an internal block edge that is also filtered internally in frame mode.

NOTE 3 – For example, in 4:2:0 chroma format when transform_size_8x8_flag is equal to 0, the following applies. 3 horizontal luma edges, 1 horizontal chroma edge for Cb, and 1 horizontal chroma edge for Cr are filtered that are internal to a macroblock. When field mode filtering (fieldModeFilteringFlag is equal to 1) is applied to the top edges of a frame macroblock, 2 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr between the frame macroblock and the above macroblock pair are filtered using field mode filtering, for a total of up to 5 horizontal luma edges, 3 horizontal chroma edges for Cb, and 3 horizontal chroma edges for Cr filtered that are considered to be controlled by the frame macroblock. In all other cases, at most 4 horizontal luma, 2 horizontal chroma edges for Cb, and 2 horizontal chroma edges for Cr are filtered that are considered to be controlled by a particular macroblock.

G.8.9.2 Construction process for not available samples

Inputs to this process are

– luma sample array size parameters (wB, hB)

– luma sample position offset (xB, yB)

– a luma sample array $base_L$[x, y] with x = –4..wB+4 and y =–4..hB+4

– when chroma_format_idc is not equal to 0,

  – a set of array size parameters (wCB, hCB)

  – the chroma sample position offset (xCB, yCB)

  – two chroma sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y] with x = –4..wCB+4 and y = –4..hCB+4

Outputs of this process are

– the luma sample array baseL[x, y] with modified sample values

– when chroma_format_idc is not equal to 0, the two chroma sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y] with modified sample values

For each component Z (with Z being L, when chroma_format_idc is equal to 0, or Z being L, Cb, or Cr, otherwise), the following applies.

– The width wZ and height hZ of the block that is upsampled are derived as follows

$$wZ = (Z == L \ ? \ wB : wCB) \quad \text{(G-359)}$$
$$hZ = (Z == L \ ? \ hB : hCB) \quad \text{(G-360)}$$

– The position offset of the block xZ and yZ are derived as follows

$$xZ = (Z == L \ ? \ xB : xCB) \quad \text{(G-361)}$$
$$yZ = (Z == L \ ? \ yB : yCB) \quad \text{(G-362)}$$

– For each sample $base_Z$[x, y] with x = -4..wZ + 4 and y= -4 .. hZ + 4 the following applies.

  – If the sample $base_Z$[x, y] is marked as “available for Intra_Base prediction”, the values of the sample array $base_Z$[x, y] are not modified.

- Otherwise (the sample $base_Z$[x, y] is marked as "not available for Intra_Base prediction"), the following applies.
  - A location difference (xD, yD) is derived by [Ed. (JR): check this equation again… make sure it corresponds to the soft]

$$xD = (((x + xZ) \% 16) \geq 8 \ ? \ (x + xZ) \% 16 - 16 \ : \ (x + xZ) \% 16) \quad \text{(G-363)}$$
$$yD = (((y + yZ) \% 16) \geq 8 \ ? \ (y + yZ) \% 16 - 16 \ : \ (y + yZ) \% 16) \quad \text{(G-364)}$$

  - The sample array $base_Z$[x, y] is modified as follows.
    - If both the sample $base_Z$[x – xD, y] and the sample $base_Z$[x, y – yD] are marked as "available for Intra_Base prediction", the following applies.
      - A variable cornerSampleAvailable is derived as follows.
        - If the sample $base_Z$[x – xD, y – yD] is marked as "available for Intra_Base prediction", the variable cornerSampleAvailable is set equal to 1.
        - Otherwise, the variable cornerSampleAvailable is set equal to 0.
      - The diagonal border extension process in subclause G.8.9.2.2 is invoked with the variables wZ and hZ, the sample array $base_Z$[x, y], the location difference (xD, yD), the location (x, y), and the variable cornerSampleAvailable as inputs and the output is the sample array $base_Z$[x, y] with modified sample values.
    - Otherwise, the following applies.
      - The location difference (xD, yD) is modified as follows.
        - If the sample $base_Z$[x – xD, y] is marked as "available for Intra_Base prediction", yD is set equal to 0.
        - Otherwise, if the sample $base_Z$[x, y – yD] is marked as "available for Intra_Base prediction", xD is set equal to 0.
        - Otherwise, the location difference (xD, yD) is not modified.
        - The constant border extension process in subclause G.8.9.2.1 is invoked with variables wZ and hZ, the sample array $base_Z$[x, y], the location difference (xD, yD), and the location (x, y) as inputs and the output is the sample array $base_Z$[x, y] with modified sample values.

G.8.9.2.1 Constant border extension process

Inputs to this process are

- variables wZ and hZ
- a sample array p[x, y] with x = –4 .. wZ + 4 and y= –4 .. hZ + 4
- a location difference (xD, yD)
- a location (xO, yO)

Output of this process is the sample array p[x, y] with modified sample values.

The value of the sample p[xO, yO] is modified as follows.

$$p[xO, yO] = p[xO - xD, yO - yD] \quad \text{(G-365)}$$

G.8.9.2.2 Diagonal border extension process

Inputs to this process are

- variables wZ and hZ
- a sample array p[x, y] with x = –4 .. wZ + 4 and y= –4 .. hZ + 4

– a location difference (xD, yD)

– a location (xO, yO)

– a variable cornerSampleAvailable

Output of this process is the sample array p[x, y] with modified sample values.

The value of the sample p[xO, yO] is modified as follows.

Let d be a variable that is derived by

$$d = Abs(yD) - Abs(xD) \quad \text{(G-366)}$$

The value of the modified sample p[xO, yO] is derived as follows.

– If d is greater then 0 the following applies.

  – If xD is less than 0 the following applies

$$p[xO, yO] = (p[xO - (xD - 2), yO - yD] + 2 * p[xO - (xD - 1), yO - yD] + p[xO - xD, yO - yD] + 2) >> 2 \quad \text{(G-367)}$$

  – Otherwise (xD is greater or equal to 0) the following applies

$$p[xO, yO] = (p[xO - (xD + 2), yO - yD] + 2 * p[xO - (xD + 1), yO - yD] + p[xO - xD, yO - yD] + 2) >> 2 \quad \text{(G-368)}$$

– Otherwise if d is less than 0 the following applies

  – If yD is less than 0 the following applies

$$p[xO, yO] = (p[xO - xD, yO - (yD - 2)] + 2 * p[xO -xD, yO - (yD - 1)] + p[xO - xD, yO - yD] + 2) >> 2 \quad \text{(G-369)}$$

  – Otherwise (yD is greater or equal to 0) the following applies

$$p[xO, yO] = (p[xO - xD, yO - (yD + 2)] + 2 * p[xO - xD, yO - (yD + 1)] + p[xO - xD, yO - yD] + 2) >> 2 \quad \text{(G-370)}$$

– Otherwise (d is equal to 0) the following applies:

  – If cornerSampleAvailable is equal to 1,

$$p[xO, yO] = (p[xO, yO - yD] + 2 * p[xO - xD, yO - yD] + p[xO - xD, yO] + 2) >> 2 \quad \text{(G-371)}$$

  – Otherwise (cornerSampleAvailable is equal to 0),

$$p[xO, yO] = (p[xO, yO - yD] + p[xO - xD, yO] + 1) >> 1 \quad \text{(G-372)}$$

G.8.9.3 Upsampling process for Intra_Base prediction

Inputs to this process are

– a macroblock address mbAddr

– integer luma sample positions in the base picture (xB, yB) and (xB1, yB1)

– a luma sample array $base_L$[x, y] with x = –2..(xB1 – xB + 3) and y = –2..(yB1 – yB + 3)

– when chroma_format_idc is not equal to 0,

- integer chroma sample positions in the base picture (xCB, yCB) and (xCB1, yCB1)
- two chroma sample arrays $base_{Cb}$[x, y] and $base_{Cr}$[x, y] with x = –1..(xCB1 – xCB + 1) and y = –1..(yCB1 – yCB + 1)

Outputs of this process are

- a luma sample array $pred_L$[x, y] with x = 0..15 and y = 0..15
- when chroma_format_idc is not equal to 0, two chroma sample arrays $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1

The variables mbIdxX and mbIdxY are derived as follows.

- mbIdxX is set equal to mbAddr % PicWidthInMbs
- mbIdxY is set equal to mbAddr / PicWidthInMbs

The luma samples $pred_L$[x, y] with x = 0..15 and y = 0..15 are derived as follows.

- Let $temp_L$[x, y] with x = –2..(xB1 – xB + 3) and y =0..15 be a temporary luma sample array.
- Each $temp_L$[x, y] with x = –2..(xB1 – xB + 3) and y = 0..15 is derived as follows.
  - The variable deltaY is derived as specified in Equation G-292.
  - The corresponding fractional-sample position yf in base layer is derived as follows.

$$yf = ((2 * (y + mbIdxY * 16 – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 \quad \text{(G-373)}$$

  - The variables yInt and yFrac are derived as follows

$$yInt = (yf >> 4) – yB \quad \text{(G-374)}$$
$$yFrac = (yf + 16) \% 16 \quad \text{(G-375)}$$

  - Select a six-tap filter e[j] with j = -2..3 from Table G-8 using yFrac as phase, and derive $temp_L$[x, y] as

$$temp_L[x, y] = base_L[x, yInt - 2]*e[-2] + base_L[x, yInt - 1]*e[-1] + base_L[x, yInt]*e[0] + base_L[x, yInt + 1]*e[1] + base_L[x, yInt + 2]*e[2] + base_L[x, yInt + 3]*e[3] \quad \text{(G-376)}$$

- Each sample $pred_L$[x, y] with x = 0..15 and y = 0..15 is derived as follows.
  - The variable deltaX is derived as specified in Equation G-291
  - The corresponding fractional-sample position xf in base layer is derived as follows.

$$xf = ((2 * (x + mbIdxX * 16 – ScaledBaseLeftOffset) + 1) * deltaX – 30720) >> 12 \quad \text{(G-377)}$$

  - The variables xInt and xFrac are derived as follows

$$xInt = (xf >> 4) – xB \quad \text{(G-378)}$$
$$xFrac = (xf + 16) \% 16 \quad \text{(G-379)}$$

  - Select a six-tap filter e[j] with j = -2..3 from Table G-8 using xFrac as phase, and derive $pred_L$[x, y] as

$$pred_L[x, y] = Clip1_Y((temp_L[xInt - 2, y]*e[-2] + temp_L[xInt - 1, y]*e[-1] + temp_L[xInt, y]*e[0] + temp_L[xInt + 1, y]*e[1] + temp_L[xInt + 2, y]*e[2] + temp_L[xInt + 3, y]*e[3] + 512) / 1024) \quad \text{(G-380)}$$

When chroma_format_idc is not equal to 0, the chroma samples $pred_C$[x, y] (with C being Cb or Cr) with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1 are derived as follows.

– Let $temp_{Cb}$[x, y] and $temp_{Cr}$[x, y] with x = -0.. ..(xCB1 – xCB + 1) and y = 0..MbHeightC - 1 be temporary chroma sample arrays.

– Each $temp_C$[x, y] with C as Cb and Cr, x = –0..(xCB1 – xCB + 1), and y = 0..MbHeightC - 1 is derived as follows.

  – The variable deltaYC is derived as specified in G-302

  – The corresponding fractional-sample position yfC in base layer is derived as follows.

$$yfC = ((((4 * (y + mbIdxY * MbHeightC – ScaledBaseTopOffsetC) + (2 + scaledBaseChromaPhaseY)) * deltaYC) + 2048) >> 12) – 4 * (2 + baseChromaPhaseY) \quad \text{(G-381)}$$

  – The variables yIntC and yFracC are derived as follows

$$yIntC = (yfC >> 4) – yCB \quad \text{(G-382)}$$
$$yFracC = (yfC + 16) \% 16 \quad \text{(G-383)}$$

  – Derive $temp_C$[x, y] as

$$temp_C[x, y] = base_C[x, yIntC]* (16 – yFracC) + base_C[x, yIntC + 1]* yFracC \quad \text{(G-384)}$$

– Each sample $pred_C$[x, y] with C as Cb and Cr, x = 0..MbWidthC - 1 and y = 0..MbHeightC - 1 is derived as follows.

  – The variable deltaXC is derived as specified in Equation G-301.

  – The corresponding fractional-sample position xfC in base layer is derived as follows.

$$xfC = ((((4 * (x + mbIdxX * MbWidthC – ScaledBaseLeftOffsetC) + (2 + scaledBaseChromaPhaseX)) * deltaXC) + 2048) >> 12) – 4 * (2 + baseChromaPhaseX) \quad \text{(G-385)}$$

  – The variables xIntC and xFracC are derived as follows

$$xIntC = (xfC >> 4) – xCB \quad \text{(G-386)}$$
$$xFracC = (xfC + 16) \% 16 \quad \text{(G-387)}$$

  – Derive $pred_C$[x, y] as

$$pred_C[x, y] = Clip1_C(temp_C[xIntC, y]* (16 – xFracC) + temp_C[xIntC + 1, y]* xFracC + 128) / 256) \quad \text{(G-388)}$$

G.8.9.4 Upsampling process for Intra_Base prediction of interlaced pictures

[Ed. (JR) this text is highly redundant with the previous sub-clause, common part should be merged]

Inputs to this process are

– a botField parameter with a value of 1 or 0 to indicate whether the input data are related to a bottom field or not

– a macroblock address mbAddr

– integer luma sample positions in basePic (xB, yB) and (xB1, yB1)

– a luma sample array $base_L$[x, y] with x = –2..(xB1 – xB + 3) and y = –6..(yB1 – yB + 6)

– when chroma_format_idc is not equal to 0,

  – integer chroma sample positions in basePic (xCB, yCB) and (xCB1, yCB1)

  – two chroma sample arrays baseCb[x, y] and $base_{Cr}$[x, y] with x = –1..(xCB1 – xCB + 1) and y = –2..(yCB1 – yCB + 2)

Outputs of this process are

– a luma sample array $pred_L[x, y]$ with x = 0..15 and y = –3..10 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise.

– when chroma_format_idc is not equal to 0, two chroma sample arrays $pred_{Cb}[x, y]$ and $pred_{Cr}[x, y]$ with x = 0..MbWidthC – 1 and y = –1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise.

The variables mbIdxX and mbIdxY are derived as follows.

– If mb_adaptive_frame_field_flag is 1 and field_pic_flag is 0

  – mbIdxX is set equal to (mbAddr / 2) % PicWidthInMbs

  – mbIdxY is set equal to (mbAddr / 2) / PicWidthInMbs * 2 + (mbAddr % 2)

– Otherwise,

  – mbIdxX is set equal to mbAddr % PicWidthInMbs

  – mbIdxY is set equal to mbAddr / PicWidthInMbs

The luma samples $pred_L[x, y]$ are derived as follows.

– Let $tempBase_L[x, y]$ be a temporary luma sample array

– If BaseFrameMbsOnlyFlag and BaseFieldPicFlag are both equal to 0, $tempBase_L[x, y]$ with x = –2..(xB1 – xB + 3) and y = –2..(yB1 – yB + 6) / 2 are derived as follows

$$tempX = Clip_3(0, BasePicWidth – 1, x +xB) -xB \quad \text{(G-389)}$$
$$tempY = Clip_3(botField, BasePicHeight – 2 + botField, y *2 + botField – (yB \% 2) + yB) - yB \quad \text{(G-390)}$$
$$tempBase_L[x, y] = base_L[tempX, tempY] \quad \text{(G-391)}$$

– Otherwise, $tempBase_L[x, y]$ with x = –2..(xB1 – xB + 3) and y = –6..(yB1 – yB + 6) are derived as follows

$$tempX = Clip_3(0, BasePicWidth – 1, x +xB) -xB \quad \text{(G-392)}$$
$$tempY = Clip_3(0, BasePicHeight – 1, y + yB) -yB \quad \text{(G-393)}$$
$$tempBase_L[x, y] = base_L[x, y] \quad \text{(G-394)}$$

– Let tempScaledBaseTopOffset be defined as follows

$$tempScaledBaseTopOffset = ScaledBaseTopOffset / 2 \quad \text{(G-395)}$$

– Let tempBasePicHeight be defined as follows

$$tempBasePicHeight = BasePicHeight * (1 + BaseFieldPicFlag | BaseFrameMbsOnlyFlag) \quad \text{(G-396)}$$

– Let tempScaledBaseHeight be defined as follows

$$tempScaledBaseHeight = ScaledBaseHeight * (1 + pic_field_flag) \quad \text{(G-397)}$$

– Let deltaY be defined as follows.

$$deltaY = ((tempBasePicHeight << 15) + (tempScaledBaseHeight >> 1)) / tempScaledBaseHeight \quad \text{(G-398)}$$

– Let tempMbY be defined as follows

  – If field_pic_flag or mb_field_decoding_flag is equal to 1

tempMbY = (mbIdxY >> (1 – field_pic_flag)) * 16 (G-399)

– Otherwise

tempMbY = mbIdxY * 8 (G-400)

– Let $temp_L$[x, y] be a temporary luma sample array with x = –2..(xB1 – xB + 3) and y = –3..10 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise. Each $temp_L$[x, y] is derived as follows

  – The corresponding fractional-sample position yf in base layer is derived as follows.

yf = ((2 * (y + tempMbY – tempScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12 (G-401)

  – Let yInt and be derived as specified in Equations G-374 and G-375.

  – Select a six-tap filter e[j] with j = -2..3 from Table G-8 using yFrac as phase, and derive $temp_L$[x, y] as

$temp_L$[x, y] = $tempBase_L$[x, yInt - 2]*e[-2] + $tempBase_L$[x, yInt - 1]*e[-1] +
$tempBase_L$[x, yInt]*e[0] + $tempBase_L$[x, yInt + 1]*e[1] +
$tempBase_L$[x, yInt + 2]*e[2] + $tempBase_L$[x, yInt + 3]*e[3] (G-402)

– Each sample $pred_L$[x, y] is derived as follows with x = 0..15 and y = –3..10 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise.

  – Let deltaX be derived as specified in Equation G-291

  – The corresponding fractional-sample position xf in base layer is derived as specified in Equation G-293.

  – Let xInt and xFrac be derived as specified in Equations G-378 and G-379.

  – Derive $pred_L$[x, y] as specified in Equation G-380.

When chroma_format_idc is not equal to 0, the chroma samples $pred_C$[x, y] (with C being Cb or Cr) are derived as follows with x = 0..MbWidthC – 1 and y = –1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise.

– Let $tempBase_C$[x, y] be a temporary chroma sample array

– If BaseFrameMbsOnlyFlag and BaseFieldPicFlag are both equal to 0, $tempBase_L$[x, y] with x = –2..(xCB1 – xCB + 3) and y = –2..(yCB1 – yCB + 6) / 2 be

tempXC = $Clip_3$(0, BasePicWidthC – 1, x +xB) -xB (G-403)
tempYC = $Clip_3$(botField, BasePicHeightC – 2 + botField, y *2 + botField – (yCB % 2) +yB) –yB (G-404)
$tempBase_C$[x, y] = $base_C$[tempXC, tempYC] (G-405)

– Otherwise, $tempBase_C$[x, y] with x = –2..(xCB1 – xCB + 3) and y = –6..(yCB1 – yCB + 6) are drived as follows

tempXC = $Clip_3$(0, BasePicWidthC – 1, x +xB) -xB (G-406)
tempYC = $Clip_3$(0, BasePicHeightC – 1, y +yB) -yB (G-407)
$tempBase_C$[x, y] = $base_C$[tempXC, tempYC] (G-408)

– Let tempScaledBaseTopOffsetC be defined as follows

tempScaledBaseTopOffsetC = ScaledBaseTopOffsetC / 2 (G-409)

– Let tempBasePicHeightC be defined as follows

tempBasePicHeightC = BasePicHeightC * (1 + BaseFieldPicFlag | BaseFrameMbsOnlyFlag) (G-410)

– Let tempScaledBaseHeightC be defined as follows

tempScaledBaseHeightC = ScaledBaseHeightC * (1 + pic_field_flag) (G-411)

– Let tempScaledBaseChromaPhaseY be defined as follows

tempScaledBaseChromaPhaseY = ScaledBaseChromaPhaseY + (botField * 2 – 1) (G-412)

– Let tempBaseChromaPhaseY be defined as follows

  – If BaseFrameMbsOnlyFlag is equal to 1

    tempBaseChromaPhaseY = BaseChromaPhaseY * 2 (G-413)

  – Otherwise,

    tempBaseChromaPhaseY = BaseChromaPhaseY + (botField * 2 – 1) (G-414)

– Let deltaYC be defined as follows.

deltaYC = ((tempBasePicHeightC << 14) + (tempScaledBaseHeightC >> 1)) /
tempScaledBaseHeightC (G-415)

– Let tempMbYC be defined as follows

  – If field_pic_flag or mb_field_decoding_flag is equal to 1

    tempMbYC = (mbIdxY >> 1) * MbHeightC (G-416)

  – Otherwise

    tempMbYC = mbIdxY * MbHeightC / 2 (G-417)

– Let $temp_C$[x, y] be temporary chroma sample arrays with x = -1.. ..(xCB1 – xCB + 1) and y = –1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise. Each $temp_C$[x, y] is derived as follows

  – The corresponding fractional-sample position yfC in base layer is derived as follows.

    yfC = = ((((8 * (y + tempMbYC – tempScaledBaseTopOffsetC) +
    (4 + tempScaledBaseChromaPhaseY)) * deltaYC)
    + 1024) >> 11) – 2 * (4 + tempBaseChromaPhaseY) (G-418)

  – Let yIntC and yFracC be derived as specified in Equations G-382 and G-383

  – Derive $temp_C$[x, y] as

    $temp_C$[x, y] = $tempBase_C$[x, yIntC]*(16 – yFracC) + $tempBase_C$[x, yIntC + 1]*yFracC (G-419)

– Each sample $pred_C$[x, y] is derived as follows with C as Cb and Cr, x = 0..MbWidthC – 1 and y = –1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise.

– Let deltaXC be derived as specified in Equation G-301

– The corresponding fractional-sample position xfC in base layer is derived as specified in Equation G-303..

– Let xIntC and xFracC be derived as specified in Equations G-386 and G-387

– Let $pred_C$[x, y] be derived as specified in Equation G-388

G.8.9.5 Vertical upsampling process for Intra_Base prediction

Inputs to this process are

– a botField parameter with a value of either 1 or 0 to indicate whether the input data are from a bottom field or not

– a luma sample array $fieldPred_L$[x, y] with x = 0..15 and y = –3..10

– when chroma_format_idc is not equal to 0, two chroma sample arrays $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y] with x = 0..MbWidthC and y = –1..MbHeightC / 2

Outputs of this process are

– a luma sample array $pred_L$[x, y] with x = 0..15 and y = 0..15

– when chroma_format_idc is not equal to 0, two chroma sample arrays $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1

The luma samples $pred_L$[x, y] with x = 0..15 and y = 0..15 are derived as follows.

– If (y % 2) is equal to botField,

$$pred_L[x, y] = fieldPred_L[x, y >> 1] \quad \text{(G-420)}$$

– Otherwise

$$\begin{aligned} pred_L[x, y] = Clip1_Y((20 * (& fieldPred_L[x, (y >> 1) - botField] + \\ & fieldPred_L[x, (y >> 1) - botField + 1]) - \\ 5 * (& fieldPred_L[x, (y >> 1) - botField - 1] + \\ & fieldPred_L[x, (y >> 1) - botField + 2]) + \\ (& fieldPred_L[x, (y >> 1) - botField - 2] + \\ & fieldPred_L[x, (y >> 1) - botField + 3]) + 16) / 32) \end{aligned} \quad \text{(G-421)}$$

When chroma_format_idc is not equal to 0, the chroma samples $pred_C$[x, y] (with C being Cb or Cr) with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1 are derived as follows.

– If (y % 2) is equal to botField,

$$pred_C[x, y] = fieldPred_C[x, y >> 1] \quad \text{(G-422)}$$

– Otherwise

$$\begin{aligned} pred_C[x, y] = Clip1_C((& fieldPred_C[x, (y >> 1) - botField] + \\ & fieldPred_C[x, (y >> 1) - botField + 1] + 1) / 2) \end{aligned} \quad \text{(G-423)}$$

G.8.10 Resampling process for residual data

Inputs to this process are

– a macroblock address mbAddr

– An array of lumaTrafoBase specifying the transform for the luma component of the base slices

– an array of residual luma sample values $picResBase_L$ for the base picture

– an array of residual luma sample values $picRes_L$ with $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ elements

– when chroma_format_idc is not equal to 0, two arrays of residual chroma sample values $picResBase_{Cb}$ and $picResBase_{Cr}$ for the base picture

– when chroma_format_idc is not equal to 0, two arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$ with $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ elements

Output of this process are

– a modified version of the array of residual luma sample values $picRes_L$

– when chroma_format_idc is not equal to 0, modified versions of the arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$

Let $resPred_L[x, y]$, with x = 0..15 and y = 0..15, be a 16x16 array of residual luma sample values

When chroma_format_idc is not equal to 0, let $resPred_{Cr}[x, y]$ and $resPred_{Cb}[x, y]$, with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1, be two (MbWidthC)x(MbHeightX) arrays of residual chroma sample values.

The position of the upper-left luma sample of the macroblock with address mbAddr is derived by invoking the inverse macroblock scanning process in subclause 6.4.1 with mbAddr as input and the output being assigned to (xP, yP).

When chroma_format_idc is not equal to 0, the position (xC, yC) of the upper-left chroma samples of the current macroblock are derived by

$$xC = xP / SubWidthC \quad \text{(G-424)}$$
$$yC = yP / SubHeightC \quad \text{(G-425)}$$

Depending on SpatialScalabilityType the following applies.

– If SpatialScalabilityType is equal to 0, the following applies.

  – The residual prediction samples $resPred_L[x, y]$ with x, y = 0..15 are derived as follows.

$$resPred_L[\ x, y\ ] = picResBase_L[\ xP - ScaledBaseLeftOffset + x, yP - ScaledBaseTopOffset + y\ ] \quad \text{(G-426)}$$

  – When chroma_format_idc is not equal to 0, residual prediction samples $resPred_C[x, y]$, with x = 0..MbWidthC – 1, y = 0..MbHeightC - 1, and with C being replaced by Cb, Cr are derived as follows

$$resPred_C[\ x, y\ ] = S_{Base,res,C}[\ xC\ xC - ScaledBaseLeftOffsetC + x, yC - ScaledBaseTopOffsetC + y\ ] \quad \text{(G-427)}$$

– Otherwise, if SpatialScalabilityType is less than or equal to 3, the upsampling process for residual prediction in subclause G.8.10.1 is invoked with the position (xP, yP), the sample array $S_{Base,res,L}$, and when chroma_format_idc is not equal to 0, the position (xC, yC) and the sample arrays $S_{Base,res,Cb}$ and $S_{Base,res,Cr}$, as inputs and the outputs are $resPred_L[\ x, y\ ]$, and when chroma_format_idc is not equal to 0, $resPred_{Cb}[\ x, y\ ]$ and $resPred_{Cr}[\ x, y\ ]$.

– Otherwise, if SpatialScalabilityType is equal to 4, the following applies

  – The control parameter botField is derived as follows

$$botField = base_frame_and_bottom_field_coincided_flag + bottom_field_flag \quad \text{(G-428)}$$

  – The upsampling process for residual prediction in subclause G.8.10.2 is invoked with the control parameter botField, the sample array $S_{Base,res,L}$, and when chroma_format_idc is not equal to 0, the sample arrays $S_{Base,res,Cb}$ and $S_{Base,res,Cr}$, as inputs and the outputs are $fieldResPred_L[\ x, y\ ]$, and when chroma_format_idc is not equal to 0, $fieldResPred_{Cb}[\ x, y\ ]$ and $fieldResPred_{Cr}[\ x, y\ ]$.

  – Depending on field_pic_flag and mb_field_decoding_flag the following applies.

    – If field_pic_flag or mb_field_decoding_flag is equal to 1, the values of array $fieldResPred_L[\ x, y\ ]$, and when chroma_format_idc is not equal to 0, $fieldResPred_{Cb}[\ x, y\ ]$ and $fieldResPred_{Cr}[\ x, y\ ]$, are asigned to $resPred_L[\ x, y\ ]$, and when chroma_format_idc is not equal to 0, $resPred_{Cb}[\ x, y\ ]$ and $resPred_{Cr}[\ x, y\ ]$, respectively.

      – Each Intra prediction luma sample $resPred_L[\ x, y\ ]$ with x = 0..15 and y = 0..15 is derived by

$resPred_L[x, y] = fieldResPred_L[x, y]$ (G-429)

– When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $resPred_{Cb}[x, y]$ and $resPred_{Cr}[x, y]$ with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

$resPred_{Cb}[x, y] = fieldResPred_{Cb}[x, y]$ (G-430)
$resPred_{Cr}[x, y] = fieldResPred_{Cr}[x, y]$ (G-431)

– Otherwise, the vertical upsampling process for residual prediction in subclause G.8.10.4 is invoked with the control parameter botField, the sample array $fieldResPred_L[x, y]$, and when chroma_format_idc is not equal to 0, the sample arrays $fieldResPred_{Cb}[x, y]$ and $fieldResPred_{Cr}[x, y]$, as inputs and the outputs are $resPred_L[x, y]$, and when chroma_format_idc is not equal to 0, $resPred_{Cb}[x, y]$ and $resPred_{Cr}[x, y]$.

– Otherwise, if SpatialScalabilityType is equal to 5, the following applies

– The control parameter botField is derived as follows

– If BaseFieldPicFlag is equal to 0

botField = base_bottom_field_coincided_flag (G-432)

– Otherwise,

botField = BaseBottomFieldFlag (G-433)

– The upsampling process for residual prediction in subclause G.8.10.2 is invoked with the control parameter botField, the sample array $S_{Base,res,L}$, and when chroma_format_idc is not equal to 0, the sample arrays $S_{Base,res,Cb}$ and $S_{Base,res,Cr}$, as inputs and the outputs are $fieldResPred_L[x, y]$, and when chroma_format_idc is not equal to 0, $fieldResPred_{Cb}[x, y]$ and $fieldResPred_{Cr}[x, y]$.

– The vertical upsampling process for residual prediction in subclause G.8.10.4 is invoked with the control parameter botField, the sample array $fieldResPred_L[x, y]$, and when chroma_format_idc is not equal to 0, the sample arrays $fieldResPred_{Cb}[x, y]$ and $fieldResPred_{Cr}[x, y]$, as inputs and the outputs are $resPred_L[x, y]$, and when chroma_format_idc is not equal to 0, $resPred_{Cb}[x, y]$ and $resPred_{Cr}[x, y]$.

– Otherwise (SpatialScalabilityType is equal to 6), the following applies

– The control parameter botField is derived as follows

– If field_pic_flag is equal to 1 [Ed. SSUN Note: this implies the base_bottom_field be available.][Ed. (HS): What does that mean for the text ????]

botField = bottom_field_flag (G-434)

– Otherwise, if BaseFieldPicFlag is equal to 1

botField = BaseBottomFieldFlag (G-435)

– Otherwise,

botField = mb_field_decoding_flag & (CurrMdAddr % 2) (G-436)

– The upsampling process for residual prediction in subclause G.8.10.2 is invoked with the control parameter botField, the sample array $S_{Base,res,L}$, and when chroma_format_idc is not equal to 0, the sample arrays $S_{Base,res,Cb}$ and $S_{Base,res,Cr}$, as inputs and the outputs are $fieldResPred_L[x, y]$, and when chroma_format_idc is not equal to 0, $fieldResPred_{Cb}[x, y]$ and $fieldResPred_{Cr}[x, y]$.

– If BaseFieldPicFlag, field_pic_flag, and mb_field_decoding_flag are all equal to 0, the following applies

– The control parameter botField is updated as follows

botField = 1 (G-437)

- The upsampling process for residual prediction in subclause G.8.10.2 is invoked with the control parameter botField, the sample array $S_{Base,res,L}$, and when chroma_format_idc is not equal to 0, the sample arrays $S_{Base,res,Cb}$ and $S_{Base,res,Cr}$, as inputs and the outputs are $field2ResPred_L$[x, y], and when chroma_format_idc is not equal to 0, $field2ResPred_{Cb}$[x, y] and $field2ResPred_{Cr}$[x, y].
- The values of array $fieldResPred_L$[x, y] and $field2ResPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldResPred_{Cb}$[x, y], $fieldResPred_{Cr}$[x, y], $field2ResPred_{Cb}$[x, y], and $field2ResPred_{Cr}$[x, y], are asigned to $resPred_L$[x, y], and when chroma_format_idc is not equal to 0, $resPred_{Cb}$[x, y] and $resPred_{Cr}$[x, y], respectively.
- Each Intra prediction luma sample $resPred_L$[x, y] with x = 0..15 and y = 0..15 is derived by

  $resPred_L$[x, y] = y%2==0? $fieldResPred_L$[x, y >> 1] : $field2ResPred_L$[x, y >> 1] (G-438)

- When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $resPred_{Cb}$[x, y] and $resPred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

  $resPred_{Cb}$[x, y] = y%2==0? $fieldResPred_{Cb}$[x, y >> 1] : $field2ResPred_{Cb}$[x, y >> 1] (G-439)
  $resPred_{Cr}$[x, y] = y%2==0? $fieldResPred_{Cr}$[x, y >> 1] : $field2ResPred_{Cr}$[x, y >> 1] (G-440)

- Otherwise, if field_pic_flag or mb_field_decoding_flag is equal to 1, the values of array $fieldPred_L$[x, y], and when chroma_format_idc is not equal to 0, $fieldPred_{Cb}$[x, y] and $fieldPred_{Cr}$[x, y], are asigned to $pred_L$[x, y], and when chroma_format_idc is not equal to 0, $pred_{Cb}$[x, y] and $pred_{Cr}$[x, y], respectively.
  - Each Intra prediction luma sample $resPred_L$[x, y] with x = 0..15 and y = 0..15 is derived by

    $resPred_L$[x, y] = $fieldResPred_L$[x, y] (G-441)

  - When chroma_format_idc is not equal to 0, each Intra prediction chroma sample $resPred_{Cb}$[x, y] and $resPred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 is derived by

    $resPred_{Cb}$[x, y] = $fieldResPred_{Cb}$[x, y] (G-442)
    $resPred_{Cr}$[x, y] = $fieldResPred_{Cr}$[x, y] (G-443)

- Otherwise, the following applies
  - The vertical upsampling process for residual prediction in subclause G.8.10.4 is invoked with the control parameter botField, the sample array $fieldResPred_L$[x, y], and when chroma_format_idc is not equal to 0, the sample arrays $fieldResPred_{Cb}$[x, y] and $fieldResPred_{Cr}$[x, y], as inputs and the outputs are $resPred_L$[x, y], and when chroma_format_idc is not equal to 0, $resPred_{Cb}$[x, y] and $resPred_{Cr}$[x, y].

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $resPred_L$, $picRes_L$ and, when chroma_format_idc is not equal to 0, $resPred_{Cb}$, $resPred_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as inputs and the outputs are a modified version of $picRes_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picRes_{Cb}$, and $picRes_{Cr}$.

G.8.10.1 Upsampling process for residual prediction

Input of this process are

- the position of the upper-left luma sample of the macroblock (xP, yP)
- the array $S_{Base,res,L}$[x, y] as reconstructed luma residual samples of the picture basePic
- when chroma_format_idc is not equal to 0,
  - the position of the upper-left luma sample of the macroblock (xC, yC)
  - two arrays $S_{Base,res,Cb}$[x, y] and $S_{Base,res,Cr}$[x, y] as reconstructed chroma residual samples of the picture basePic, one for each of the chroma components Cb and Cr.

Output of this process are

– an array $resPred_L$[x, y], with x = 0..15 and y = 0..15, of the residual prediction luma samples for the current macroblock.

– when chroma_format_idc is not equal to 0, two arrays $resPred_{Cr}$[x, y] and $resPred_{Cb}$[x, y], with x = 0..MbWidthC - 1 and y = 0..MbHeightC - 1, of the residual prediction chroma samples for the current macroblock, one for each of the chroma components Cb and Cr.

The following parameters are derived based on syntax elements chroma_phase_x_plus1, chroma_phase_y_plus1, base_chroma_phase_x_plus1, base_chroma_phase_y_plus1

scaledBaseChromaPhaseX = chroma_phase_x_plus1 – 1 (G-444)
scaledBaseChromaPhaseY = chroma_phase_y_plus1 – 1 (G-445)
baseChromaPhaseX = base_chroma_phase_x_plus1 – 1 (G-446)
baseChromaPhaseY = base_chroma_phase_y_plus1 – 1 (G-447)

The residual prediction samples resPredL[x, y] with x, y = 0..15 are derived as follows.

– The variables deltaX and deltaY are derived as specified in Equations G-291 and G-292

– Let arrays basePosX4[x] and basePosY4[y] with x = 0..15 and y = 0..15 be derived as follows.

basePosX4[x] = max(0, ((2 * (x + xP – ScaledBaseLeftOffset) + 1) * deltaX – 30720) >> 12)(G-448)

basePosY4[y] = max(0, ((2 * (y + yP – ScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12)(G-449)

– The full-sample position (xB, yB) is derived by

xB = basePosX4[0] >> 4 (G-450)
yB = basePosY4[0] >> 4 (G-451)

– The full-sample position (xB1, yB1) is derived by

xB1 = (basePosX4[15] + 15) >> 4 (G-452)
yB1 = (basePosY4[15] + 15) >> 4 (G-453)

– An array TransBlkBase[x, y] with x = 0..xB1 – xB and y = 0..yB1 – yB is derived as follows.

  – A basePic macroblock address TempMbAddrBase is derived as

  TempMbAddrBase = (y + yB) / 16 * BasePicWidthInMbs + (x + xB) / 16 (G-454)

  – If transformTypeBase[currMbAddrBase] is equal to T_4x4

  TransBlkBase[x, y] = ((((y + yB) / 16) % 2) * 2 + (((x + xB) / 16) % 2)) * 16 +
  (((y + yB) / 4) % 4) * 4 + (((x + xB) / 4) % 4) (G-455)

  – Otherwise (transformTypeBase[currMbAddrBase] is equal to T_8x8)

  TransBlkBase[x, y] = ((((y + yB) / 16) % 2) * 2 + (((x + xB) / 16) % 2)) * 16 +
  (((y + yB) / 8) % 2) * 2 + (((x + xB) / 8) % 2) (G-456)

  – An array r[x, y] with x = 0..xB1 – xB and y = 0..yB1 – yB is derived by

  r[x, y] = $picResBase_L$[xB + x, yB + y] (G-457)

– The upsampling process as specified in G.8.10.3 is invoked with (xB, yB), (16, 16), (0, 16), basePosX4, basePosY4, (xB1 – xB + 1, yB1 – yB + 1), TransBlkBase, and r as input and $resPred_L$[x, y] with x, y = 0..15 as output.

When chroma_format_idc is not equal to 0, residual prediction samples $resPred_C$[x, y], with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1, and with C being replaced by Cb, Cr, are derived as follows

– Let deltaXC and deltaYC be defined as specified in Equations G-301 and G-302.

– Let the arrays basePosCX4[x] and basePosCY4[y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 be derived as follows.

```
basePosCX4[x] = max( 0, ( ( ( ( 4 * ( x + xC – ScaledBaseLeftOffsetC ) +
      ( 2 + scaledBaseChromaPhaseX ) ) * deltaXC )
      + 2048 ) >> 12 ) – 4 * ( 2 + baseChromaPhaseX ) )                    (G-458)
```

```
basePosCY4[y] = max( 0, ( ( ( ( 4 * ( y + yC – ScaledBaseTopOffsetC ) +
      ( 2 + scaledBaseChromaPhaseY ) ) * deltaYC )
      + 2048 ) >> 12 ) – 4 * ( 2 + baseChromaPhaseY ) )                    (G-459)
```

– The full-sample position (xCB, yCB) is derived by

```
xCB = basePosCX4[0] >> 4                                                   (G-460)
yCB = basePosCY4[0] >> 4                                                   (G-461)
```

– The full-sample position (xCB1, yCB1) is derived by

```
xCB1 = ( basePosCX4[ MbWidthC – 1 ] + 15 ) >> 4                            (G-462)
yCB1 = ( basePosCY4[ MbHeightC –1 ] + 15 ) >> 4                            (G-463)
```

– An array TransBlkBaseC[x, y] with x = 0..xCB1 – xCB and y = 0..yCB1 – yCB is derived as follows.

```
BaseMbIdxX = ( x + xCB ) / BaseMbWidthC                                    (G-464)
```

```
BaseMbIdxY = ( y + yCB ) / BaseMbHeightC                                   (G-465)
```

```
TransBlkBaseC[x, y] = ( ( BaseMbIdxY % 2 ) * 2 + ( BaseMbIdxX % 2 ) ) * 16 +
                      ( ( ( y + yCB – BaseMbIdxY * BaseMbHeightC ) / 8 ) % 2 ) * 2 +
                      ( ( ( x + xCB – BaseMbIdxX * BaseMbWidthC ) / 8 ) % 2 )   (G-466)
```

– An array r[x, y] with x = 0..xCB1 – xCB and y = 0..yCB1 – yCB is derived by

$$r[\ x, y\ ] = picResBase_C[\ (\ xCB + x, yCB + y\ ] \qquad \text{(G-467)}$$

– The upsampling process as specified in G.8.10.3 is invoked with (xCB, yCB), (MbWidthC, MbHeightC) , (0, MbHeightC), basePosCX4, basePosCY4, (xCB1 – xCB + 1, yCB1 – yCB + 1), TransBlkBaseC, and r as input and $resPred_C$[x, y] with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1 as output.

G.8.10.2 Upsampling process for residual prediction in interlaced pictures

[Ed. (JR) this text is highly redundant with the previous sub-clause, common part should be merged]

Input of this process are

– a botField parameter with a value of 1 or 0 to indicate whether the input data are related to a bottom field or not

– the array $S_{Base,res,L}$[x, y] as reconstructed luma residual samples of the picture basePic

– when chroma_format_idc is not equal to 0, two arrays $S_{Base,res,Cb}$[x, y] and $S_{Base,res,Cr}$[x, y] as reconstructed chroma residual samples of the picture basePic, one for each of the chroma components Cb and Cr.

Output of this process are

– an array $resFieldPred_L$[x, y] of the residual prediction luma samples for the current macroblock, with x = 0..15 and y = -1..8 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise.

– when chroma_format_idc is not equal to 0, two arrays $resFieldPred_{Cr}$[x, y] and $resFieldPred_{Cb}$[x, y] of the residual prediction chroma samples for the current macroblock, one for each of the chroma components Cb and Cr, with x = 0..MbWidthC - 1 and y = -1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise.

The residual prediction samples resFieldPredL[x, y] are derived as follows with x, y = -1..8 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise.

– Let $tempResBase_L$[x, y] be a temporary luma sample array

  – If BaseFrameMbsOnlyFlag and BaseFieldPicFlag are both equal to 0, $tempResBase_L$[x, y] with x = 0..BasePicWidth and y = -1..BasePicHeight / 2 be

    – If y<0

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, botField\ ] \quad \text{(G-468)}$$

    – Otherwise, if y< BasePicHeight / 2

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, y*2 + botField\ ] \quad \text{(G-469)}$$

    – Otherwise

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, BasePicHeight - 2 + botField\ ] \quad \text{(G-470)}$$

  – Otherwise, $tempResBase_L$[x, y] with x = 0..BasePicWidth and y = -1..BasePicHeight are drived as follows

    – If y<0

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, 0\ ] \quad \text{(G-471)}$$

    – Otherwise, if y< BasePicHeight

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, y\ ] \quad \text{(G-472)}$$

    – Otherwise

$$tempResBase_L[\ x, y\ ] = S_{Base,res,L}[\ x, BasePicHeight - 1\ ] \quad \text{(G-473)}$$

– Let tempScaledBaseTopOffset be defined as follows

$$tempScaledBaseTopOffset = ScaledBaseTopOffset / 2 \quad \text{(G-474)}$$

– Let tempBasePicHeight be defined as follows

$$tempBasePicHeight = BasePicHeight * (\ 1 + BaseFieldPicFlag \mid BaseFrameMbsOnlyFlag\ ) \quad \text{(G-475)}$$

– Let tempScaledBaseHeight be defined as follows

$$tempScaledBaseHeight = ScaledBaseHeight * (\ 1 + pic_field_flag\ ) \quad \text{(G-476)}$$

– Let deltaX and deltaY be defined as follows.

deltaX = ((BasePicWidth << 15) + (ScaledBaseWidth >> 1)) / ScaledBaseWidth (G-477)
deltaY = = ((tempBasePicHeight << 15) + (tempScaledBaseHeight >> 1)) / tempScaledBaseHeight (G-478)

– Let tempMbY be defined as follows

  – If field_pic_flag or mb_field_decoding_flag is equal to 1

tempMbY = (MbIdxY >> (1 – field_pic_flag)) * 16 (G-479)

  – Otherwise

tempMbY = MbIdxY * 8 (G-480)

– Let arrays basePosX4[x] and basePosY4[y] be derived as follows with x = 0..15 and y = -1..8 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..15 otherwise.

basePosX4[x] = max(0, ((2 * (x + xP – ScaledBaseLeftOffset) + 1) * deltaX – 30720) >> 12)(G-481)
basePosY4[y] = max(0, ((2 * (y + tempMbY – tempScaledBaseTopOffset) + 1) * deltaY – 30720) >> 12) (G-482)

– When both field_pic_flag and mb_field_decoding_flag are equal to 0, the full-sample position (xB, yB) is derived as follows.

xB = basePosX4[0] >> 4 (G-483)
yB = basePosY4[-1] >> 4 (G-484)

– Otherwise

xB = basePosX4[0] >> 4 (G-485)
yB = basePosY4[0] >> 4 (G-486)

– When both field_pic_flag and mb_field_decoding_flag are equal to 0, the full-sample position (xB1, yB1) is derived as follows.

xB1 = (basePosX4[15] + 15) >> 4 (G-487)
yB1 = (basePosY4[8] + 15) >> 4 (G-488)

– Otherwise

xB1 = (basePosX4[15] + 15) >> 4 (G-489)
yB1 = (basePosY4[15] + 15) >> 4 (G-490)

– An array TransBlkBase[x, y] with x = 0..xB1 – xB and y = 0..yB1 – yB is derived as follows.

TransBlkBase[x, y] = ((((y + yB) / 16) % 2) * 2 + (((x + xB) / 16) % 2)) * 16 + (((y + yB) / 8) % 2) * 2 + (((x + xB) / 8) % 2) (G-491)

– An array r[x, y] with x = 0..xB1 – xB and y = 0..yB1 – yB is derived by

r[x, y] = $tempResBase_L$[xB + x, yB + y] (G-492)

– Let tempBlockY be defined as follows

tempBlockY = 8 * (1 + field_pic_flag | mb_field_decoding_flag) (G-493)

– Let (ys, ye) be defined as follows
  – if both field_pic_flag and mb_field_decoding_flag are equal to 0

    ys = -1, ye = tempBlockY+1 (G-494)

  – otherwise (field_pic_flag or mb_field_decoding_flag is equal to 1),

    ys = 0, ye = tempBlockY (G-495)

– The upsampling process as specified in G.8.10.3 is invoked with (xB, yB), (16, tempBlockY), (ys, ye), basePosX4, basePosY4, (xB1 – xB + 1, yB1 – yB + 1), TransBlkBase, and r as input and $resFieldPred_L$[x, y] as output with x, y = ys...ye-1.

When chroma_format_idc is not equal to 0, residual prediction samples $resFieldPred_C$[x, y], with C being replaced by Cb, Cr are derived as follows and with x = 0..MbWidthC – 1, y = -1..MbHeightC / 2 when both field_pic_flag and mb_field_decoding_flag are equal to 0 or y = 0..MbHeightC – 1 otherwise,

– Let $tempResBase_C$[x, y] be a temporary chroma sample array
  – If BaseFrameMbsOnlyFlag and BaseFieldPicFlag are both equal to 0, $tempResBase_C$[x, y] with x = 0..BasePicWidthC and y = -1..BasePicHeightC / 2 be
    – If y<0

      $tempResBase_C[x, y] = S_{Base,res,C}[x, botField]$ (G-496)

    – Otherwise, if y< BasePicHeightC / 2

      $tempResBase_C[x, y] = S_{Base,res,C}[x, y *2 + botField]$ (G-497)

    – Otherwise

      $tempResBase_C[x, y] = S_{Base,res,C}[x, BasePicHeightC – 2 +botField]$ (G-498)

  – Otherwise, $tempResBase_C$[x, y] with x x = 0..BasePicWidthC and y = -1..BasePicHeightC are drived as follows
    – If y<0

      $tempResBase_C[x, y] = S_{Base,res,C}[x, 0]$ (G-499)

    – Otherwise, if y< BasePicHeightC

      $tempResBase_C[x, y] = S_{Base,res,C}[x, y]$ (G-500)

    – Otherwise

      $tempResBase_C[x, y] = S_{Base,res,C}[x, BasePicHeightC – 1]$ (G-501)

– Let tempScaledBaseTopOffsetC be defined as follows

  tempScaledBaseTopOffsetC = ScaledBaseTopOffsetC / 2 (G-502)

– Let tempBasePicHeightC be defined as follows

tempBasePicHeightC = BasePicHeightC * (1 + BaseFieldPicFlag | BaseFrameMbsOnlyFlag) (G-503)

– Let tempScaledBaseHeightC be defined as follows

tempScaledBaseHeightC = ScaledBaseHeightC * (1 + pic_field_flag) (G-504)

– Let tempScaledBaseChromaPhaseY be defined as follows

tempScaledBaseChromaPhaseY = ScaledBaseChromaPhaseY + (botField * 2 – 1) (G-505)

– Let tempBaseChromaPhaseY be defined as follows

  – If BaseFrameMbsOnlyFlag is equal to 1

tempBaseChromaPhaseY = BaseChromaPhaseY * 2 (G-506)

  – Otherwise,

tempBaseChromaPhaseY = BaseChromaPhaseY + (botField * 2 – 1) (G-507)

– Let deltaXC and deltaYC be defined as follows.

deltaXC = ((BasePicWidthC << 14) + (ScaledBaseWidthC >> 1)) / ScaledBaseWidthC (G-508)
deltaYC = ((tempBasePicHeightC << 14) +
(tempScaledBaseHeightC >> 1)) / tempScaledBaseHeightC (G-509)

– Let tempMbYC be defined as follows

  – If field_pic_flag or mb_field_decoding_flag is equal to 1

tempMbYC = (MbIdxY >> (1 – field_pic_flag)) * MbHeightC (G-510)

  – Otherwise

tempMbYC = MbIdxY * MbHeightC / 2 (G-511)

– Let arrays basePosCX4[x] and basePosCY4[y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1 be derived as follows.

basePosCX4[x] = max(0, ((((4 * (x + xC – ScaledBaseLeftOffsetC) +
(2 + scaledBaseChromaPhaseX)) * deltaXC)
+ 2048) >> 12) – 4 * (2 + baseChromaPhaseX)) (G-512)
basePosCY4[y] = max(0, ((((4 * (y + tempMbYC – tempScaledBaseTopOffsetC) +
(2 + tempScaledBaseChromaPhaseY)) * deltaYC)
+ 2048) >> 12) – 4 * (2 + tempBaseChromaPhaseY)) (G-513)

– When both field_pic_flag and mb_field_decoding_flag are equal to 0, the full-sample position (xCB, yCB) is derived as follows.

xCB = basePosCX4[0] >> 4 (G-514)
yCB = basePosCY4[-1] >> 4 (G-515)

– Otherwise

xCB = basePosCX4[0] >> 4 (G-516)
yCB = basePosCY4[0] >> 4 (G-517)

– When both field_pic_flag and mb_field_decoding_flag are equal to 0, the full-sample position (xCB1, yCB1) is derived as follows.

xCB1 = (basePosCX4[MbWidthC – 1] + 15) >> 4 (G-518)
yCB1 = (basePosCY4[MbHeightC / 2] + 15) >> 4 (G-519)

– Otherwise

xCB1 = (basePosCX4[MbWidthC – 1] + 15) >> 4 (G-520)
yCB1 = (basePosCY4[MbHeightC –1] + 15) >> 4 (G-521)

– An array TransBlkBaseC[x, y] with x = 0..xCB1 – xCB and y = 0..yCB1 – yCB is derived as follows.

BaseMbIdxX = (x + xCB) / BaseMbWidthC (G-522)
BaseMbIdxY = (y + yCB) / BaseMbHeightC (G-523)
TransBlkBaseC[x, y] = ((BaseMbIdxY % 2) * 2 + (BaseMbIdxX % 2)) * 16 +
(((y + yCB – BaseMbIdxY * BaseMbHeightC) / 8) % 2) * 2 +
(((x + xCB – BaseMbIdxX * BaseMbWidthC) / 8) % 2) (G-524)

– An array r[x, y] with x = 0..xCB1 – xCB and y = 0..yCB1 – yCB is derived by

r[x, y] = $tempResBase_C$[xCB + x, yCB + y] (G-525)

– Let tempBlockY be defined as follows

tempBlockCY = MbHeightC * (1 + field_pic_flag | mb_field_decoding_flag) / 2 (G-526)

– Let (ys, ye) be defined as follows

  – if both field_pic_flag and mb_field_decoding_flag are equal to 0

  ys = -1, ye = tempBlockCY +1 (G-527)

  – otherwise (field_pic_flag or mb_field_decoding_flag is equal to 1),

  ys = 0, ye = tempBlockCY (G-528)

– The upsampling process as specified in G.8.10.3 is invoked with (xCB, yCB), (MbWidthC, tempBlockCY), (ys, ye), basePosCX4, basePosCY4, (xCB1 – xCB + 1, yCB1 – yCB + 1), TransBlkBaseC, and r as input and $resFieldPred_C$[x, y] as output with x = 0..MbWidthC – 1, y = ys..ye-1.

G.8.10.3 Bilinear interpolation process for residual prediction

Input to this process are

– two variables (x0, y0) represent an integer position offset

– two variables (m, n) representing the output block size

– two variables (ys, ye) representing the range of the y position

– an array posX[x] with x = 0..m – 1

– an array posY[y] with y = ys..ye-1.

– two variables (mb, nb) representing the base block size

– an array transBlkIdx[x, y] with x = 0..mb – 1 and y = 0..nb – 1

– an array resBase[x, y] with x = 0..mb – 1 and y = 0..nb – 1.

Output of this process is an array resInterp[x, y] with x = 0..m – 1 and y = ys..ye – 1.

Each output sample resInterp[x, y] with x = 0..m – 1 and y = ys..ye – 1 is derived as follows.

– Let the integer positions (x1, y1) and (x2, y2) derived as

x1 = (posX[x] >> 4) – x0 (G-529)
y1 = (posY[y] >> 4) – y0 (G-530)

  – If (posX[x] % 16) is not equal to 0,

    x2 = x1 + 1 (G-531)

  – Otherwise,

    x2 = x1 (G-532)

  – If (posY[y] % 16) is not equal to 0

    y2 = y1 + 1 (G-533)

  – Otherwise

    y2 = y1 (G-534)

– Let the variable temp1 be derived as follows

  – If transBlkIdx[x1, y1] is equal to transBlkIdx[x2, y1]

    temp1 = r[x1, y1] * (16 – (posX[x] % 16)) + r[x2, y1] * (posX[x] % 16) (G-535)

  – Otherwise, if (posX[x] % 16) < 8

    temp1 = r[x1, y1] * 16 (G-536)

  – Otherwise

    temp1 = r[x2, y1] * 16 (G-537)

– Let the variable temp2 be derived as follows.

  – If transBlkIdx[x1, y2] is equal to transBlkIdx[x2, y2]

    temp2 = r[x1, y2] * (16 – (posX[x] % 16)) + r[x2, y2] * (posX[x] % 16) (G-538)

  – Otherwise, if (posX[x] % 16) < 8

    temp2 = r[x1, y2] * 16 (G-539)

  – Otherwise

    temp2 = r[x2, y2] * 16 (G-540)

– Let resInterp be derived as follows.

  – If transBlkIdx[x1, y1] is equal to transBlkIdx[x1, y2]

    resInterp[x, y] = (temp1 * (16 – (posY[y] % 16)) + temp2 * (posY[y] % 16) + (128)) >> 8) (G-541)

  – Otherwise, if (posY[x] % 16) < 8

    resInterp[x, y] = (temp1 + 8) >> 4) (G-542)

  – Otherwise

    resInterp[x, y] = (temp2 + 8) >> 4) (G-543)

G.8.10.4 Vertical upsampling process for residual prediction

Inputs to this process are

– a botField parameter with a value of either 1 or 0 to indicate whether the input data are from a bottom field or not

– a luma sample array $fieldResPred_L$[x, y] with x = 0..15 and y = –1..8

– when chroma_format_idc is not equal to 0, two chroma sample arrays $fieldResPred_{Cb}$[x, y] and $fieldResPred_{Cr}$[x, y] with x = 0..MbWidthC and y = –1..MbHeightC / 2

Outputs of this process are

– a luma sample array $resPred_L$[x, y] with x = 0..15 and y = 0..15

– when chroma_format_idc is not equal to 0, two chroma sample arrays $resPred_{Cb}$[x, y] and $resPred_{Cr}$[x, y] with x = 0..MbWidthC – 1 and y = 0..MbHeightC – 1

The luma samples $resPred_L$[x, y] with x = 0..15 and y = 0..15 are derived as follows.

– If (y % 2) is equal to botField,

  $resPred_L$[x, y] = $fieldResPred_L$[x, y >> 1] (G-544)

– Otherwise,

  $resPred_L$[x, y] = ($fieldResPred_L$[x, (y >> 1) – botField] +
  $fieldResPred_L$[x, (y >> 1) – botField + 1] + 1) / 2 (G-545)

When chroma_format_idc is not equal to 0, the chroma samples $resPred_C$[x, y] (with C being Cb or Cr) with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1 are derived as follows.

– If (y % 2) is equal to botField,

  $resPred_C$[x, y] = $fieldResPred_C$[x, y >> 1] (G-546)

– Otherwise,

  $resPred_C$[x, y] = ($fieldResPred_C$[x, (y >> 1) – botField] +
  $fieldResPred_C$[x, (y >> 1) – botField + 1] + 1) / 2 (G-547)

G.8.11 Transformation, scaling, and picture construction processes

Subclause G.8.11.1 specifies the initialization process for macroblock, sub-macroblock, and luma transform type.

Subclause G.8.11.2 specifies the transform coefficient scaling process.

Subclause G.8.11.3 specifies the progressive refinement process for scaled transform coefficients.

Subclause G.8.11.4 specifies the residual accumulation process.

Subclause G.8.11.5 specifies the residual construction process.

Subclause G.8.11.6 specifies the picture construction process prior to deblocking filter process.

Subclause G.8.11.7 specifies the sample construction process.

G.8.11.1 Initialization process for macroblock, sub-macroblock, and luma transform type

Inputs to this process are

- a macroblock address mbAddr
- an array mbTypeBase specifying the macroblock types for the base macroblocks
- an array subMbTypeBase containing lists of 4 elements specifying the sub-macroblock types for the base macroblocks
- two arrays predFlagL0Base and predFlagL1Base containing arrays of prediction utilization flags for each base macroblock
- syntax elements for a macroblock

Outputs of this process are

- a variable fieldMb specifying whether the macroblock is a field or frame macroblock
- a variable mbType specifying the macroblock type
- a list subMbType of 4 elements specifying the sub-macroblock types (when appropriate)
- a variable lumaTrafo specifying the luma transform type

Depending on base_mode_flag the following applies.

- If base_mode_flag is equal to 1, the following applies.
  - The derivation process for macroblock type and sub-macroblock type in inter-layer prediction as specified in subclause G.6.4 is invoked with mbAddr, mbTypeBase, subMbTypeBase, predFlagL0Base, predFlagL1Base as inputs and the outputs are assigned to mbType and subMbType. The variable mbType shall not be marked as not available after invoking the process in subclause G.6.4.
- Otherwise (base_mode_flag is equal to 0), the following applies.
  - The variable mbType is set equal to mb_type.
  - For mbPartIdx = 0..3, the variable subMbType[mbPartIdx] is set equal to sub_mb_type[mbPartIdx].

Depending on the mb_type and transform_size_8x8_flag the following applies.

- If mb_type is equal to I_PCM, lumaTrafo is set equal to T_IPCM.
- Otherwise, if MbPartPredMode(mb_type, 0) is equal to Intra_16x16, lumaTrafo is set equal to T_16x16.
- Otherwise (mb_type is not equal to I_PCM and MbPartPredMode(mb_type, 0) is not equal to Intra_16x16), the following applies.
  - If transform_size_8x8_flag is equal to 1, lumaTrafo is set equal to T_8x8.
  - Otherwise (transform_size_8x8_flag is equal to 0), lumaTrafo is set equal to T_4x4.

Depending on mb_field_decoding_flag, the following applies.

- If mb_field_decoding_flag is equal to 1, fieldMb is set equal to 1.
- Otherwise(mb_field_decoding_flag is equal to 0), fieldMb is set equal to 0.

G.8.11.2 Transform coefficient scaling process

Input to this process is a variable lumaTrafo specifying the transform that shall be applied for the luma component of the current macroblock.

Outputs of this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements

– a list of significance indicators sigBCoeff with 256 + 2 * MbWidthC * MbHeightC elements

– a list of context identifiers ctx4x4Id with 16 elements

All elements of the list of scaled transform coefficient values sTCoeff are initialized with 0

sTCoeff[k] = 0 with k = 0...256 + 2 * MbWidthC * MbHeightC – 1 (G-548)

The progressive refinement process for scaled transform coefficients as specified in subclause G.8.11.3 is invoked with fieldMb, lumaTrafo and sTCoeff as input and a modified version of sTCoeff as output.

The list of significance indicators is derived by

sigBCoeff[k] = (sTCoeff[k] != 0 ? 1 : 0) with k = 0...256 + 2 * MbWidthC * MbHeightC – 1 (G-549)

Depending on lumaTrafo the following applies.

– If lumaTrafo is equal to T_4x4 the following applies.

  – For luma4x4BlkIdx = 0..15, the process specified in subclause 9.3.3.1.1.9 is invoked with ctxBlockCat equal to 2 and luma4x4BlkIdx as the inputs and the output is assigned to ctx4x4Id[luma4x4BlkIdx].

– Otherwise (lumaTrafo is not equal to T_4x4), all elements ctx4x4Id[luma4x4BlkIdx] with luma4x4BlkIdx = 0..15 are set equal to 0.

G.8.11.3 Progressive refinement process for scaled transform coefficients

Inputs to this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a variable lumaTrafo specifying the transform that shall be applied for the luma component of the current macroblock

– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements.

Output of this process is a modified version of the list of scaled transform coefficient values sTCoeff.

The derivation process for chroma quantization parameters and scaling function as specified in subclause 8.5.7 is invoked.

Depending on lumaTrafo the following applies.

– If lumaTrafo is equal to T_4x4, the progressive refinement process for scaled transform coefficients of 4x4 luma blocks as specified in subclause G.8.11.3.1 is invoked with fieldMb, sTCoeff as input and a modified version of sTCoeff as output.

– Otherwise, if lumaTrafo is equal to T_8x8, the progressive refinement process for scaled transform coefficients of 8x8 luma blocks as specified in subclause G.8.11.3.2 is invoked with fieldMb, sTCoeff as input and a modified version of sTCoeff as output.

– Otherwise, lumaTrafo is equal to T_16x16, the progressive refinement process for scaled luma transform coefficients of Intra_16x16 macroblocks as specified in subclause G.8.11.3.3 is invoked with fieldMb, sTCoeff as input and a modified version of sTCoeff as output.

– Otherwise (lumaTrafo is equal to T_IPCM), the list of scaled transform coefficient values is not modified.

When chroma_format_idc is not equal to 0 and lumaTrafo is not equal to T_IPCM, the following applies.

– For each chroma component, with iCbCr set equal to 0 for Cb and iCbCr set equal to 1 for Cr, the progressive refinement process for scaled chroma transform coefficients as specified in subclause G.8.11.3.4 is invoked with fieldMb, sTCoeff and iCbCr as input and a modified version of sTCoeff as output.

G.8.11.3.1 Progressive refinement process for scaled transform coefficient of 4x4 luma blocks

Inputs to this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a list of scaled transform coefficient values sTCoeff

Output of this process is a modified version of the list of scaled transform coefficient values sTCoeff.

For each 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following applies.

– The inverse transform coefficient scanning process as specified in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the 4x4 array c with elements c_{ij} as the output. For the invocation of process 8.5.5, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

– The scaling process for residual 4x4 blocks as specified in subclause 8.5.10.1 is invoked with the 4x4 array c as input and the 4x4 array d with elements d_{ij} as the output.

– The list of scaled transform coefficient values sTCoeff is modified by

$$\text{sTCoeff[16 * luma4x4BlkIdx + 4 * i + j] += } d_{ij} \quad \text{with i, j = 0...3} \qquad \text{(G-550)}$$

The bitstream shall not contain data that result in any element sTCoeff[k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_Y)}$ to $2^{(7 + \text{BitDepth}_Y)}-1$, inclusive.

G.8.11.3.2 Progressive refinement process for scaled transform coefficient of 8x8 luma blocks

Inputs to this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a list of scaled transform coefficient values sTCoeff

Output of this process is a modified version of the list of scaled transform coefficient values sTCoeff.

For each 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following applies.

– The inverse transform coefficient scanning process for 8x8 luma transform coefficients as specified in subclause 8.5.6 is invoked with LumaLevel8x8[luma8x8BlkIdx] as the input and the 8x8 array c with elements c_{ij} as the output. For the invocation of process 8.5.6, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

– The scaling process for residual 8x8 luma blocks as specified in subclause 8.5.11.1 is invoked with the 8x8 array c as input and the 8x8 array d with elements d_{ij} as the output.

– The list of scaled transform coefficient values sTCoeff is modified by

$$\text{sTCoeff[64 * luma8x8BlkIdx + 8 * i + j] += } d_{ij} \quad \text{with i, j = 0...7} \qquad \text{(G-551)}$$

The bitstream shall not contain data that result in any element sTCoeff[k] with k = 0..255 that exceeds the range of integer values from $-2^{(7 + \text{BitDepth}_Y)}$ to $2^{(7 + \text{BitDepth}_Y)}-1$, inclusive.

G.8.11.3.3 Progressive refinement process for scaled luma transform coefficients of Intra_16x16 macroblocks

Inputs to this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a list of scaled transform coefficient values sTCoeff

Output of this process is a modified version of the list of scaled transform coefficient values sTCoeff.

The inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with Intra16x16DCLevel as the input and the 4x4 array c with elements c_{ij} as the output. For the invocation of process 8.5.5, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

Let d be a 4x4 array of scaled luma DC transform coefficient values with elements d_{ij}, which are derived by

$d_{ij} = c_{ij} * (LevelScale(QP'_Y \% 6, 0, 0) << (QP'_Y / 6))$ with i, j = 0..3 (G-552)

The list of scaled transform coefficient values sTCoeff is modified by

$sTCoeff[128 * (i / 2) + 64 * (j / 2) + 32 * (i \% 2) + 16 * (j \% 2)] += d_{ij}$ with i, j = 0...3 (G-553)

For each 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following applies.

– The variable lumaList, which is a list of 16 entries, is derived as follows.

lumaList[0] = 0 (G-554)
lumaList[k] = Intra16x16ACLevel[luma4x4BlkIdx][k – 1] with k = 1..15 (G-555)

– The inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with lumaList as the input and the 4x4 array c with elements c_{ij} as the output. For the invocation of process 8.5.5, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

– The scaling process for residual 4x4 blocks as specified in subclause 8.5.10.1 is invoked with the 4x4 array c as input and the 4x4 array d with elements d_{ij} as the output.

NOTE 1 – The element d_{00} is always equal to 0 after the invocation of the scaling process for residual 4x4 blocks.

– The list of scaled transform coefficient values sTCoeff is modified by

$sTCoeff[16 * luma4x4BlkIdx + 4 * i + j] += d_{ij}$ with i, j = 0...3 (G-556)

NOTE 2 – Since the element d_{00} is always equal to 0, the element sTCoeff[16 * luma4x4BlkIdx] is not modified at this point.

The bitstream shall not contain data that result in any element sTCoeff[16* b + k] with k = 1..15 and b = 0..15 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

G.8.11.3.4 Progressive refinement process for scaled chroma transform coefficients

Inputs to this process are

– a variable fieldMb specifying whether a macroblock is a field or a frame macroblock

– a list of scaled transform coefficient values sTCoeff

– a chroma component identifier iCbCr

Output of this process is a modified version of the list of scaled transform coefficient values sTCoeff.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

Let the variable cO be set equal to 256 + iCbCr * MbWidthC * MbHeightC.

Depending on chroma_format_idc the following applies.

– If chroma_format_idc is equal to 1 the following applies.

  – The 2x2 array c with elements c_{ij} is derived by

  c_{ij} = ChromaDCLevel[iCbCr][2 * i + j] with i, j = 0..1 (G-557)

  – Let d be a 2x2 array of scaled chroma DC transform coefficient values with elements d_{ij}, which are derived by

  $d_{ij} = c_{ij} * (LevelScale(QP'_C \% 6, 0, 0) << (QP'_C / 6))$ with i, j = 0..1 (G-558)

  – The list of scaled transform coefficient values sTCoeff is modified by

  $sTCoeff[cO + 32 * i + 16 * j] += d_{ij}$ with i, j = 0...1 (G-559)

– Otherwise, if chroma_format_idc is equal to 2 the following applies.

  – Let scan422ChromaDC be a list of the eight following values

scan422ChromaDC = { 0, 2, 1, 5, 3, 6, 4, 7 } (G-560)

– The 2x4 array c with elements c_{ij} is derived by

c_{ij} = ChromaDCLevel[iCbCr][scan422ChromaDC[2 * i + j]] with i = 0..3 and j = 0..1 (G-561)

– Let d be a 2x4 array of scaled chroma DC transform coefficient values with elements d_{ij}, which are derived by

$d_{ij} = c_{ij}$ * (LevelScale((QP'_C + 3) % 6, 0, 0) << ((QP'_C + 3) / 6)) with i = 0..3 and j = 0..1 (G-562)

– The list of scaled transform coefficient values sTCoeff is modified by

sTCoeff[cO + 32 * i + 16 * j] += d_{ij} with i = 0..3 and j = 0...1 (G-563)

– Otherwise (chroma_format_idc is equal to 3), the following applies.

  – The inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with ChromaDCLevel[iCbCr] as the input and the 4x4 array c with elements c_{ij} as the output. For the invocation of process 8.5.5, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

  – Let d be a 4x4 array of scaled chroma DC transform coefficient values with elements d_{ij}, which are derived by

$d_{ij} = c_{ij}$ * (LevelScale(QP'_C % 6, 0, 0) << (QP'_C / 6)) with i, j = 0..3 (G-564)

  – The list of scaled transform coefficient values sTCoeff is modified by

sTCoeff[cO + 128 * (i / 2) + 64 * (j / 2) + 32 * (i % 2) + 16 * (j % 2)] += d_{ij} with i, j = 0...3 (G-565)

For each 4x4 chroma block indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the following applies.

– The variable chromaList, which is a list of 16 entries, is derived as follows.

chromaList[0] = 0 (G-566)
chromaList[k] = ChromaACLevel[chroma4x4BlkIdx][k – 1] with k = 1..15 (G-567)

– The inverse scanning process for transform coefficients as specified in subclause 8.5.5 is invoked with chromaList as the input and the 4x4 array c with elements c_{ij} as the output. For the invocation of process 8.5.5, if fieldMb is equal to 1, the macroblock is considered as field macroblock; otherwise (fieldMb is equal to 0), it is considered as frame macroblock.

– The scaling process for residual 4x4 blocks as specified in subclause 8.5.10.1 is invoked with the 4x4 array c as input and the 4x4 array d with elements d_{ij} as the output.

NOTE 1 – The element d_{00} is always equal to 0 after the invocation of the scaling process for residual 4x4 blocks.

– The list of scaled transform coefficient values sTCoeff is modified by

sTCoeff[cO + 16 * chroma4x4BlkIdx + 4 * i + j] += d_{ij} with i, j = 0...3 (G-568)

NOTE 2 – Since the element d_{00} is always equal to 0, the element sTCoeff[cO + 16 * chroma4x4BlkIdx] is not modified at this point.

The bitstream shall not contain data that result in any element sTCoeff[cO + 16* b + k] with k = 1..15 and b = 0..numChroma4x4Blks – 1 that exceeds the range of integer values from $-2^{(7 + BitDepth'_C)}$ to $2^{(7 + BitDepth'_C)}-1$, inclusive.

G.8.11.4 Residual accumulation process

Inputs to this process are

– a macroblock address mbAddr

– a variable lumaTrafo specifying the luma transform type

– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements

– an array of residual luma sample values $picRes_L$ with ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) elements

– when chroma_format_idc is not equal to 0, two arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$ with $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ elements

Outputs of this process are

– a modified version of the array of residual luma sample values $picRes_L$

– when chroma_format_idc is not equal to 0, modified versions of the arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$

The macroblock residual construction process as specified in subclause G.8.11.5.1 is invoked with lumaTrafo and sTCoeff as the inputs and the outputs are a 16x16 arrays of residual luma sample values $mbRes_L$ and, when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays of residual chroma sample values $mbRes_{Cb}$ and $mbRes_{Cr}$.

The macroblock sample extraction process as specified in subclause G.8.11.7.2 is invoked with mbAddr, $picRes_L$, and when chroma_format_idc is equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are assigned to $mbResBase_L$ and, when chroma_format_idc is not equal to 0, $mbResBase_{Cb}$ and $mbResBase_{Cr}$.

All elements of the 16x16 array $mbRes_L$ with x, y = 0..15 are modified by

$$mbRes_L[\ x, y\ ] = Clip3(\ yMin, yMax, mbRes_L[\ x, y\ ] + mbResBase_L[\ x, y\ ]\ ) \quad \text{(G-569)}$$

with

$$yMin = -(\ 1 << BitDepth_Y\ ) + 1 \quad \text{(G-570)}$$
$$yMax = (\ 1 << BitDepth_Y\ ) - 1 \quad \text{(G-571)}$$

All elements of the (MbWidthC)x(MbHeightC) arrays $mbRes_C$ with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1, and C being replaced by Cb or Cr are modified by

$$mbRes_C[\ x, y\ ] = Clip3(\ cMin, cMax, mbRes_C[\ x, y\ ] + mbResBase_C[\ x, y\ ]\ ) \quad \text{(G-572)}$$

with

$$cMin = -(\ 1 << BitDepth_C\ ) + 1 \quad \text{(G-573)}$$
$$cMax = (\ 1 << BitDepth_C\ ) - 1 \quad \text{(G-574)}$$

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $mbRes_L$, $picRes_L$ and, when chroma_format_idc is not equal to 0, $mbRes_{Cb}$, $mbRes_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as inputs and the outputs are a modified version of $picRes_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picRes_{Cb}$, and $picRes_{Cr}$.

G.8.11.5 Residual construction process

Inputs to this process are

– a macroblock address mbAddr

– a variable lumaTrafo specifying the luma transform type

– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements

– an array of residual luma sample values $picRes_L$ with $(PicWidthInSamples_L)x(PicHeightInSamples_L)$ elements

– when chroma_format_idc is not equal to 0, two arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$ with $(PicWidthInSamples_C)x(PicHeightInSamples_C)$ elements

Outputs of this process are

– a modified version of the array of residual luma sample values $picRes_L$

– when chroma_format_idc is not equal to 0, modified versions of the arrays of residual chroma sample values $picRes_{Cb}$ and $picRes_{Cr}$

The macroblock residual construction process as specified in subclause G.8.11.5.1 is invoked with lumaTrafo and sTCoeff as the inputs and the outputs are a 16x16 arrays of residual luma sample values $mbRes_L$ and, when

chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays of residual chroma sample values $mbRes_{Cb}$ and $mbRes_{Cr}$.

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $mbRes_L$, $picRes_L$ and, when chroma_format_idc is not equal to 0, $mbRes_{Cb}$, $mbRes_{Cr}$, $picRes_{Cb}$, and $picRes_{Cr}$ as inputs and the outputs are a modified version of $picRes_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picRes_{Cb}$, and $picRes_{Cr}$.

G.8.11.5.1 Macroblock residual construction process

Inputs to this process are

– a variable lumaTrafo specifying the luma transform type

– a list of scaled transform coefficient values sTCoeff with 256 + 2 * MbWidthC * MbHeightC elements

Outputs of this process are

– a 16x16 array of residual luma sample values for the current macroblock $mbRes_L$

– when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays of residual chroma sample values $mbRes_{Cr}$ and $mbRes_{Cb}$.

Depending on lumaTrafo, the 16x16 array of residual luma samples $mbRes_L$ is derived as follows.

– If lumaTrafo is equal to T_4x4, the residual construction process for 4x4 luma blocks as specified in subclause G.8.11.5.2 is invoked with sTCoeff as the input and $mbRes_L$ as the output.

– Otherwise, if lumaTrafo is equal to T_8x8, the residual construction process for 8x8 luma blocks as specified in subclause G.8.11.5.3 is invoked with sTCoeff as the input and $mbRes_L$ as the output.

– Otherwise, if lumaTrafo is equal to T_16x16, the residual construction process for the luma component of Intra_16x16 macroblocks as specified in subclause G.8.11.5.4 is invoked with sTCoeff as the input and $mbRes_L$ as the output.

– Otherwise (lumaTrafo is equal to T_IPCM), all elements of the 16x16 array $mbRes_L$ are set equal to 0.

When chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays of residual chroma samples $mbRes_{Cb}$ and $mbRes_{Cr}$ are derived as follows.

– If lumaTrafo is equal to T_IPCM, all elements of the (MbWidthC)x(MbHeightC) arrays $mbRes_{Cb}$ and $mbRes_{Cr}$ are set equal to 0.

– Otherwise (lumaTrafo is not equal to T_IPCM), the following applies.

  – The residual construction process for the chroma component as specified in subclause G.8.11.5.5 is invoked with sTCoeff and iCbCr equal to 0 as inputs and $mbRes_{Cb}$ as the output.

  – The residual construction process for the chroma component as specified in subclause G.8.11.5.5 is invoked with sTCoeff and iCbCr equal to 1 as inputs and $mbRes_{Cr}$ as the output.

G.8.11.5.2 Residual construction process for 4x4 luma blocks

Input to this process is a list of scaled transform coefficient values sTCoeff.

Output of this process is a 16x16 array of residual luma sample values for the current macroblock $mbRes_L$.

For each 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following applies.

– Let d be a 4x4 array of scaled luma transform coefficient values with elements d_{ij}, which are derived by

$$d_{ij} = sTCoeff[\ 16 * luma4x4BlkIdx + 4 * i + j\ ] \quad \text{with } i, j = 0...3 \qquad \text{(G-575)}$$

– The transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with d as the input and the 4x4 array r with elements r_{ij} as the output.

– The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx and the output is assigned to (xP, yP).

– The elements of the array $mbRes_L$, which cover the 4x4 luma block indexed by luma4x4BlkIdx, are derived by

$mbRes_L[xP + j, yP + i] = r_{ij}$ with i, j = 0...3 (G-576)

G.8.11.5.3 Residual construction process for 8x8 luma blocks

Input to this process is a list of scaled transform coefficient values sTCoeff.

Output of this process is a 16x16 array of residual luma sample values for the current macroblock $mbRes_L$.

For each 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following applies.

– Let d be an 8x8 array of scaled luma transform coefficient values with elements d_{ij}, which are derived by

d_{ij} = sTCoeff[64 * luma8x8BlkIdx + 8 * i + j] with i, j = 0...7 (G-577)

– The transformation process for residual 8x8 luma blocks as specified in subclause 8.5.11.2 is invoked with d as the input and the 8x8 array r with elements r_{ij} as the output.

– The inverse 8x8 luma block scanning process as specified in subclause 6.4.4 is invoked with luma8x8BlkIdx and the output is assigned to (xP, yP).

– The elements of the array $mbRes_L$, which cover the 8x8 luma block indexed by luma8x8BlkIdx, are derived by

$mbRes_L[xP + j, yP + i] = r_{ij}$ with i, j = 0...7 (G-578)

G.8.11.5.4 Residual construction process for the luma component of Intra_16x16 macroblocks

Input to this process is a list of scaled transform coefficient values sTCoeff.

Output of this process is a 16x16 array of residual luma sample values for the current macroblock $mbRes_L$.

Let c be a 4x4 array of scaled luma DC coefficients with the elements c_{ij}, which are derived by

c_{ij} = sTCoeff[128 * (i / 2) + 64 * (j / 2) + 32 * (i % 2) + 16 * (j % 2)] with i, j = 0...3 (G-579)

The transformation process for luma DC transform coefficients for Intra_16x16 macroblock type as specified in subclause 8.5.8.1 is invoked with c as input and the 4x4 array f of DC values for luma 4x4 blocks with elements f_{ij} as output.

Let dcY be a 4x4 array of scaled DC values for 4x4 luma blocks with elements dcY_{ij}, which are derived by

dcY_{ij} = (f_{ij} + (1 << 5)) >> 6 with i, j = 0...3 (G-580)

The bitstream shall not contain data that results in any element dcY_{ij} of dcY with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + BitDepth_Y)}$ to $2^{(7 + BitDepth_Y)}-1$, inclusive.

NOTE – Successive invocations of subclause G.8.11.3.3 (as part of an invocation of subclause G.8.11.2) and this subclause for Intra_16x16 macroblocks produce results for all elements dcY_{ij} of the 4x4 array of scaled luma DC values dcY that are mathematically identical to the results that are obtained by an invocation of subclause 8.5.8.

For each 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following applies.

– Let d be a 4x4 array of scaled luma transform coefficient values with elements d_{ij}

– The element d_{00} of d is derived by

$d_{00} = dcY_{kl}$ with k = 2 * (luma4x4BlkIdx / 8) + (luma4x4BlkIdx % 4) / 2
and l = 2 * ((luma4x4BlkIdx % 8) / 4) + (luma4x4BlkIdx % 2) (G-581)

– The elements d_{ij} of d with i, j = 0..3 and i + j > 0 of d are derived by

d_{ij} = sTCoeff[16 * luma4x4BlkIdx + 4 * i + j] (G-582)

– The transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with d as the input and the 4x4 array r with elements r_{ij} as the output.

– The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx and the output is assigned to (xP, yP).

– The elements of the array $mbRes_L$, which cover the 4x4 luma block indexed by luma4x4BlkIdx, are derived by

$mbRes_L[\ xP + j, yP + i\ ] = r_{ij}$ with i, j = 0...3 (G-583)

G.8.11.5.5 Residual construction process for the chroma component

Inputs to this process are

– a list of scaled transform coefficient values sTCoeff

– a chroma component identifier iCbCr

Output of this process is a (MbWidthC)x(MbHeightC) array of residual chroma sample values for the current macroblock and the given chroma component $mbRes_C$.

Let the variable numChroma4x4Blks be set equal to (MbWidthC / 4) * (MbHeightC / 4).

Let the variable cO be set equal to 256 + iCbCr * MbWidthC * MbHeightC.

Let c be a (MbWidthC / 4)x(MbHeightC / 4) array of scaled chroma DC coefficients with the elements c_{ij}, which are derived as follows

– If chroma_format_idc is equal to 1 or 2,

c_{ij} = sTCoeff[cO + 32 * i + 16 * j] with i = 0..(MbHeightC / 4) – 1 and j = 0...1 (G-584)

– Otherwise (chroma_format_idc is equal to 3),

c_{ij} = sTCoeff[cO + 128 * (i / 2) + 64 * (j / 2) + 32 * (i % 2) + 16 * (j % 2)] with i, j = 0...3 (G-585)

The transformation process for chroma DC transform coefficients as specified in subclause 8.5.9.1 is invoked with c as input and the (MbWidthC / 4)x(MbHeightC / 4) array f of DC values for chroma 4x4 blocks with elements f_{ij} as output.

Let dcC be a (MbWidthC / 4)x(MbHeightC / 4) array of scaled DC values for 4x4 chroma blocks with elements dcC_{ij}, which are derived as follows.

– If chroma_format_idc is equal to 1,

$dcC_{ij} = f_{ij} >> 5$ with i, j = 0...1 (G-586)

– Otherwise (chroma_format_idc is equal to 2 or 3),

$dcC_{ij} = (f_{ij} + (1 << 5)) >> 6$ with i = 0...3 and j = 0..(MbWidthC / 4) – 1 (G-587)

The bitstream shall not contain data that results in any element dcC_{ij} of dcC with i = 0..(MbHeightC / 4) – 1 and j = 0..(MbWidthC / 4) – 1 that exceeds the range of integer values from $-2^{(7 + BitDepth'_C)}$ to $2^{(7 + BitDepth'_C)}-1$, inclusive.

NOTE – Successive invocations of subclause G.8.11.3.4 and this subclause for a chroma component produce results for all elements dcC_{ij} of the (MbWidthC / 4)x(MbHeightC / 4) array of scaled chroma DC values dcC that are mathematically identical to the results that are obtained by an invocation of subclause 8.5.9.

For each 4x4 chroma block indexed by chroma4x4BlkIdx = 0..numChroma4x4Blks – 1, the following applies.

– Let d be a 4x4 array of scaled luma transform coefficient values with elements d_{ij}

– The element d_{00} of d is derived as follows

  – If chroma_format_idc is equal to 1 or 2,

  $d_{00} = dcY_{kl}$ with k = chroma4x4BlkIdx / 2
  and l = chroma4x4BlkIdx % 2 (G-588)

  – Otherwise (chroma_format_idc is equal to 3),

  $d_{00} = dcY_{kl}$ with k = 2 * (chroma4x4BlkIdx / 8) + (chroma4x4BlkIdx % 4) / 2
  and l = 2 * ((chroma4x4BlkIdx % 8) / 4) + (chroma4x4BlkIdx % 2) (G-589)

– The elements d_{ij} of d with i, j = 0..3 and i + j > 0 of d are derived by

d_{ij} = sTCoeff[16 * chroma4x4BlkIdx + 4 * i + j] (G-590)

– The transformation process for residual 4x4 blocks as specified in subclause 8.5.10.2 is invoked with d as the input and the 4x4 array r with elements r_{ij} as the output.

– Depending on chroma_format_idc a chroma location (xP, yP) is derived as follows.

  – If chroma_format_idc is equal to 1 or 2,

    xP = 4 * (chroma4x4BlkIdx % 2) (G-591)
    yP = 4 * (chroma4x4BlkIdx / 2) (G-592)

  – Otherwise (chroma_format_idc is equal to 3), the inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with chroma4x4BlkIdx and the output is assigned to (xP, yP).

– The elements of the array $mbRes_C$, which cover the 4x4 chroma block indexed by chroma4x4BlkIdx, are derived by

    $mbRes_C[xP + j, yP + i] = r_{ij}$ with i, j = 0...3 (G-593)

G.8.11.6 Picture construction process prior to deblocking filter process

Inputs to this process are

– a macroblock address mbAddr

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma samples

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples

Outputs of this process are constructed samples prior to the deblocking filter process.

The macroblock sample extraction process as specified in subclause G.8.11.7.2 is invoked with mbAddr, $picSamples_L$, and when chroma_format_idc is equal to 0, $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $mbSamples_L$ and, when chroma_format_idc is not equal to 0, $mbSamples_{Cb}$ and $mbSamples_{Cr}$.

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $mbSamples_L$, S'_L and, when chroma_format_idc is not equal to 0, $mbSamples_{Cb}$, $mbSamples_{Cr}$, S'_{Cb}, and S'_{Cr} as inputs and the outputs are a modified version of S'_L and, when chroma_format_idc is not equal to 0, modified versions of S'_{Cb}, and S'_{Cr}.

G.8.11.7 Sample construction process

Inputs to this process are

– a macroblock address mbAddr

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picRes_L$ of residual luma samples

– an ($PicWidthInSamples_L$)x($PicHeightInSamples_L$) array $picSamples_L$ of luma samples

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picRes_{Cb}$ and $picRes_{Cr}$ of residual chroma samples

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples

Outputs of this process are

– a modified version of the array $picSamples_L$ of luma samples

– when chroma_format_idc is not equal to 0, modified versions of the arrays $picSamples_{Cb}$ and $picSamples_{Cr}$ of chroma samples

The macroblock sample extraction process as specified in subclause G.8.11.7.2 is invoked with mbAddr, $picRes_L$, and when chroma_format_idc is equal to 0, $picRes_{Cb}$ and $picRes_{Cr}$ as the inputs and the outputs are assigned to $mbResBase_L$ and, when chroma_format_idc is not equal to 0, $mbResBase_{Cb}$ and $mbResBase_{Cr}$.

The macroblock sample extraction process as specified in subclause G.8.11.7.2 is invoked with mbAddr, $picSamples_L$, and when chroma_format_idc is equal to 0, $picSamples_{Cb}$ and $picSamples_{Cr}$ as the inputs and the outputs are assigned to $mbSamples_L$ and, when chroma_format_idc is not equal to 0, $mbSamples_{Cb}$ and $mbSamples_{Cr}$.

All elements of the 16x16 array $mbSamples_L$ with x, y = 0..15 are modified by

$$mbSamples_L[\ x, y\ ] = Clip1_Y(\ mbSamples_L[\ x, y\ ] + mbRes_L[\ x, y\ ]\ ) \qquad \text{(G-594)}$$

All elements of the (MbWidthC)x(MbHeightC) arrays $mbSamples_C$ with x = 0..MbWidthC – 1, y = 0..MbHeightC – 1, and C being replaced by Cb or Cr are modified by

$$mbSamples_C[\ x, y\ ] = Clip1_C(\ mbSamples_C[\ x, y\ ] + mbRes_C[\ x, y\ ]\ ) \qquad \text{(G-595)}$$

The picture sample array construction process as specified in subclause G.8.11.7.1 is invoked with mbAddr, $mbSamples_L$, $picSamples_L$ and, when chroma_format_idc is not equal to 0, $mbSamples_{Cb}$, $mbSamples_{Cr}$, $picSamples_{Cb}$, and $picSamples_{Cr}$ as inputs and the outputs are a modified version of $picSamples_L$ and, when chroma_format_idc is not equal to 0, modified versions of $picSamples_{Cb}$, and $picSamples_{Cr}$.

G.8.11.7.1 Picture sample array construction process

Inputs to this process are

– a macroblock address mbAddr

– a 16x16 array $mbArray_L$ of luma sample values for the macroblock mbAddr

– a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ of luma sample values

– when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ of chroma sample values for the macroblock mbAddr

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ of chroma sample values

Outputs of this process are

– a possibly modified version of the array $picArray_L$ of luma sample values

– when chroma_format_idc is not equal to 0, possibly modified versions of the arrays $picArray_{Cb}$ and $picArray_{Cr}$ of chroma sample values

The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with mbAddr, mbW equal to 16, mbH equal to 16, $mbArray_L$, and $picArray_L$ as the inputs and a possibly modified version of $picArray_L$ as the output.

When chroma_format_idc is not equal to 0, the following applies.

– The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with mbAddr, mbW equal to MbWidthC, mbH equal to MbHeightC, $mbArray_{Cb}$, and $picArray_{Cb}$ as the inputs and a possibly modified version of $picArray_{Cb}$ as the output.

– The picture sample array construction process for a signal component as specified in subclause G.8.11.7.3 is invoked with mbAddr, mbW equal to MbWidthC, mbH equal to MbHeightC, $mbArray_{Cr}$, and $picArray_{Cr}$ as the inputs and a possibly modified version of $picArray_{Cr}$ as the output.

G.8.11.7.2 Macroblock sample array extraction process

Inputs to this process are

– a macroblock address mbAddr

– a ($PicWidthInSamples_L$)x($PicWidthInHeight_L$) array $picArray_L$ of luma sample values

– when chroma_format_idc is not equal to 0, two ($PicWidthInSamples_C$)x($PicHeightInSamples_C$) arrays $picArray_{Cb}$ and $picArray_{Cr}$ of chroma sample values

Outputs of this process are

– a 16x16 array $mbArray_L$ of luma sample values for the macroblock mbAddr

– when chroma_format_idc is not equal to 0, two (MbWidthC)x(MbHeightC) arrays $mbArray_{Cb}$ and $mbArray_{Cr}$ of chroma sample values for the macroblock mbAddr

The macroblock sample array extraction process for a signal component as specified in subclause G.8.11.7.4 is invoked with mbAddr, mbW equal to 16, mbH equal to 16, and $picArray_L$ as the inputs and the output is assigned to $mbArray_L$.

When chroma_format_idc is not equal to 0, the following applies.

– The macroblock sample array extraction process for a signal component as specified in subclause G.8.11.7.4 is invoked with mbAddr, mbW equal to MbWidthC, mbH equal to MbHeightC, and $picArray_{Cb}$ as the inputs and the output is assigned to $mbArray_{Cb}$.

– The macroblock sample array extraction process for a signal component as specified in subclause G.8.11.7.4 is invoked with mbAddr, mbW equal to MbWidthC, mbH equal to MbHeightC, and $picArray_{Cr}$ as the inputs and the output is assigned to $mbArray_{Cr}$.

G.8.11.7.3 Picture sample array construction process for a signal component

Inputs to this process are

– a macroblock address mbAddr

– two variables mbW and mbH specifying the width and height of the macroblock signal in samples, respectively

– a (mbW)x(mbH) array mbArray of sample values for the macroblock mbAddr

– a (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray of sample values

Output of this process is a possibly modified version of picArray.

A location (xP, yP) is derived as follows.

– The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr and the output is assigned to (xO, yO).

– The location (xP, yP) is derived by

$$xP = (xO * mbWidth) >> 4 \quad \text{(G-596)}$$
$$yP = ((yO * mbHeight) >> 4) + (yO \% 2) \quad \text{(G-597)}$$

Depending on the variable MbaffFrameFlag and the macroblock with macroblock address mbAddr, the array picArray is modified as follows.

– If MbaffFrameFlag is equal to 1 and the macroblock with macroblock address mbAddr is a field macroblock,

$$picArray[xP + x, yP + 2 * y] = mbArray[x, y] \quad \text{with } x = 0..mbW - 1, y = 0..mbH - 1 \quad \text{(G-598)}$$

– Otherwise (MbaffFrameFlag is equal to 0 or the macroblock with macroblock address mbAddr is a frame macroblock),

$$picArray[xP + x, yP + y] = mbArray[x, y] \quad \text{with } x = 0..mbW - 1, y = 0..mbH - 1 \quad \text{(G-599)}$$

G.8.11.7.4 Macroblock sample array extraction process for a signal component

Inputs to this process are

– a macroblock address mbAddr

– two variables mbW and mbH specifying the width and height of the macroblock signal in samples, respectively

– a (mbW * PicWidthInMbs)x(mbH * PicHeightInMbs) array picArray of sample values

Output of this process is a (mbW)x(mbH) array mbArray of sample value for the macroblock mbAddr.

A location (xP, yP) is derived as follows.

– The inverse macroblock scanning process as specified in subclause 6.4.1 is invoked with CurrMbAddr and the output is assigned to (xO, yO).

– The location (xP, yP) is derived by

$$xP = (xO * mbWidth) >> 4 \quad \text{(G-600)}$$
$$yP = ((yO * mbHeight) >> 4) + (yO \% 2) \quad \text{(G-601)}$$

Depending on the variable MbaffFrameFlag and the macroblock with macroblock address mbAddr, the samples of the array mbArray are derived as follows.

– If MbaffFrameFlag is equal to 1 and the macroblock with macroblock address mbAddr is a field macroblock,

$$mbArray[x, y] = picArray[xP + x, yP + 2 * y] \quad \text{with } x = 0..mbW - 1, y = 0..mbH - 1 \quad \text{(G-602)}$$

– Otherwise (MbaffFrameFlag is equal to 0 or the macroblock with macroblock address mbAddr is a frame macroblock),

$$mbArray[x, y] = picArray[xP + x, yP + y] \quad \text{with } x = 0..mbW - 1, y = 0..mbH - 1 \quad \text{(G-603)}$$

G.8.12 Decoding process for reference picture lists construction

This process is invoked at the beginning of decoding each P, EP, B, or EB slice, or at the beginning of a PR slice with dependency_id equal to DependencyIdmax, motion_refinement_flag equal to 1, and BaseSliceType equal to P, EP, B, or EB.

Inputs to this process are

– a variable currDependencyId

– a variable useBaseRepFlag (if available)

Outputs of this process are

– a reference picture list refPicList0

– for B and EB slices, and for PR slices with BaseSliceType equal to B or EB, a reference picture list refPicList1

NOTE – Assume that the current slice has temporal_level equal to tL. The output reference picture list refPicList0 or refPicList1 shall not contain reference pictures that have temporal_level greater than tL.

When the variable useBaseRepFlag is not specified as input to this process, it shall be inferred to be equal to 0.

A variable biPred is derived as follows:

– If the current slice is a B or EB slice or if the current slice is a PR slice with BaseSliceType equal to B or EB, biPred is set equal to 1.

– Otherwise, biPred is set equal to 0.

Decoded reference pictures are marked as "used for short-term reference" or "used for long-term reference" as specified by the bitstream and specified in subclause 8.2.5. Short-term reference pictures are identified by the value of frame_num that is decoded in the slice header(s) with dependency_id equal to currDependencyId. Long-term reference pictures are assigned a long-term frame index as specified by the bitstream and specified in subclause 8.2.5.

Subclause G.8.12.1 is invoked with currDependencyId and useBaseRepFlag as inputs to specify

– the assignment of variables FrameNum, FrameNumWrap, and PicNum to each of the short-term reference pictures, and

– the assignment of variables LongTermPicNum to each of the long-term reference pictures

NOTE – During the invocation of subclause G.8.12.1 the variables FrameNum, FrameNumWrap, PicNum, and LongTermPicNum can be marked as “unspecified”. Furthermore, the variables LongTermFrameIdx can be temporarily marked as “unspecified” for this subclause and all invocations inside this subclause.

Reference pictures are addressed through reference indices as specified in subclause 8.4.2.1. A reference index is an index into a reference picture list. When biPred is equal to 0, a single reference picture list refPicList0 is constructed. When decoding a B or EB slice, a second independent reference picture list refPicList1 is constructed in addition to refPicList0.

At the beginning of decoding of each slice, reference picture list refPicList0, and for biPred equal to 1 refPicList1, are derived as follows.

– An initial reference picture list RefPicList0 and for biPred equal to 1 RefPicList1 are derived by invoking the initialization process for reference picture lists as specified in subclause 8.2.4.2. During the initialization process in subclause 8.2.4.2 all short-term reference pictures for which FrameNum, FrameNumWrap, and PicNum have been marked as "unspecified" by the invocation of subclause G.8.12.1 and all long-term pictures for which LongTermFrameIdx and LongTermPicNum have been marked as "unspecified" by the invocation of subclause G.8.12.1 are considered as non-existing. The reference picture lists are constructed by using only short-term pictures with values of FrameNum, FrameNumWrap, and PicNum that are not marked as "unspecified" and long-term pictures with values of LongTermFrameIdx and LongTermPicNum that are not marked as "unspecified".

– The initial reference picture list RefPicList0 and for biPred equal to 1 RefPicList1 are modified by invoking the reordering process for reference picture lists as specified in subclause 8.2.4.3. During the reordering process in subclause 8.2.4.2 all short-term reference pictures for which FrameNum, FrameNumWrap, and PicNum have been marked as "unspecified" by the invocation of subclause G.8.12.1 and all long-term pictures for which LongTermFrameIdx and LongTermPicNum have been marked as "unspecified" by the invocation of subclause G.8.12.1 are considered as non-existing. Short-term pictures with values of FrameNum, FrameNumWrap, and PicNum that are not marked as "unspecified" and long-term pictures with values of LongTermFrameIdx and LongTermPicNum that are not marked as "unspecified" shall not be referred to by reference picture list reordering commands.

– RefPicList0 is assigned to refPicList0.

– For biPred equal to 1, RefPicList1 is assigned to refPicList1.

NOTE – By the invocation of the process in subclause G.8.12.1 the parameters of some reference picture might have been marked as "unspecified". Since, these pictures in not considered in the construction process for reference picture lists, the reference picture lists refPicList0 and for biPred equal to 1 refPicList1 are dependent on the value of the input parameter currDependencyId.

The number of entries in the modified reference picture list refPicList0 is num_ref_idx_l0_active_minus1 + 1, and for biPred equal to 1 the number of entries in the modified reference picture list refPicList1 is num_ref_idx_l1_active_minus1 + 1. A reference picture may appear at more than one index in the modified reference picture lists refPicList0 or refPicList1.

For all long-term reference pictures for which LongTermFrameIdx has been marked as "unspecified" during the invocation of subclause G.8.12.1, this marking of LongTermFrameIdx is removed.

G.8.12.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause G.8.12 is invoked.

Inputs to this process are

– a variable currDependencyId

– a variable useBaseRepFlag

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.4.2 and the modification process for reference picture lists in subclause 8.2.4.3.

A variable maxFrameNum is derived as follows.

– Let log2MaxFrameNumMinus4 be the value of the syntax element log2_max_frame_num_minus4 that has been decoded in the sequence parameter set, which is referenced by the slice header with dependency_id equal to currDependencyId.

– The variable maxFrameNum is derived by

$$maxFrameNum = (1 << (log2MaxFrameNumMinus4 + 4)) \quad \text{(G-604)}$$

To each short-term and long-term reference picture a variable useFlag is assigned as follows.

– If slices with dependency_id equal to currDependencyId have not been present in the access unit for the corresponding reference picture, useFlag is set equal to 0.

– Otherwise (slices with dependency_id equal to currDependencyId have been present in the access unit for the corresponding reference picture), the following applies.

- If useBaseRepFlag is equal to 0, the following applies.
  - If the reference picture is not marked as “base representation”, useFlag is set equal to 1.
  - Otherwise (the reference picture is marked as “base representation”, useFlag is set equal to 0.
- Otherwise, if the reference picture is a short-term reference picture, the following applies.
  - If the short-term reference picture is marked as “base representation” or if there does not exist a short-term reference picture with an identical value frame_num coded in slice header(s) with dependency_id equal to currDependencyId that is marked as “base representation”, useFlag is set equal to 1.
  - Otherwise (the short-term picture is not marked as base representation and there exist a short-term reference picture with an identical value frame_num coded in slice header(s) with dependency_id equal to currDependencyId that is marked as “base representation”), useFlag is set equal to 0.
- Otherwise (useBaseRepFlag is equal to 1 and the reference picture is a long-term reference picture), the following applies.
  - If the long-term reference picture is marked as “base representation” or if there does not exist a long-term reference picture with an identical value of LongTermFrameIdx that is marked as “base representation”, useFlag is set equal to 1.
  - Otherwise (the long-term picture is not marked as base representation and there exist a long-term reference picture with an identical value of LongTermFrameIdx that is marked as “base representation”), useFlag is set equal to 0.

To each short-term reference picture the variables FrameNum and FrameNumWrap are assigned as follows.

- If useFlag is equal to 0, FrameNum and FrameNumWrap for the short-term picture are marked as “unspecified”.
- Otherwise, FrameNum is set equal to the syntax element frame_num that has been decoded in the slice header(s) with dependency_id equal to currDependencyId of the corresponding short-term reference picture. And the FrameNumWrap is derived by

```
if( FrameNum > frame_num )
        FrameNumWrap = FrameNum – maxFrameNum                    (G-605)
else
        FrameNumWrap = FrameNum
```

  where the value of frame_num used in Equation G-605 is the frame_num in the slice header(s) with dependency_id equal to currDependencyId for the current access unit.

NOTE – The values of FrameNum and FrameNumWrap are dependent on the value of currDependencyId. For the calculation of FrameNum and FrameNumWrap only syntax elements of NAL units with dependency_id equal to currDependencyId are used.

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.5). For each long-term picture, the following applies.

- When useFlag is equal to 0, LongTermFrameIdx is temporarily marked as “unspecified”. This marking is removed after the invocation of the decoding process for reference picture lists construction as specified in subclause G.8.12.

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned. The values of these variables depend on the value of field_pic_flag and bottom_field_flag for the current slice with dependency_id equal to currDependencyId and they are set as follows.

- If field_pic_flag is equal to 0, the following applies.
  - For each short-term reference frame or complementary reference field pair the following applies
    - If FrameNumWrap is marked as “unspecified”, PicNum is marked as “unspecified”.
    - Otherwise (FrameNumWrap is not marked as “unspecified”), PicNum is derived by

```
PicNum = FrameNumWrap                                            (G-606)
```

- For each long-term reference frame or long-term complementary reference field pair the following applies
  - If LongTermFrameIdx is marked as “unspecified”, LongTermPicNum is marked as “unspecified”.
  - Otherwise (LongTermFrameIdx is not marked as “unspecified”), LongTermPicNum is derived by

    LongTermPicNum = LongTermFrameIdx (G-607)

- Otherwise (field_pic_flag is equal to 1), the following applies.
  - Let current field refer to the slices of the current access unit with dependency_id equal to currDependencyId.
  - For each short-term reference field the following applies.
    - If FrameNumWrap is marked as “unspecified”, PicNum is marked as unspecified.
    - Otherwise (FrameNumWrap is not marked as “unspecified”), the following applies.
      - If the reference field has the same parity as the current field

        PicNum = 2 * FrameNumWrap + 1 (G-608)

      - Otherwise (the reference field has the opposite parity of the current field),

        PicNum = 2 * FrameNumWrap (G-609)

  - For each long-term reference field the following applies.
    - If LongTermFrameIdx is marked as “unspecified”, LongTermPicNum is marked as “unspecified”.
    - Otherwise (LongTermFrameIdx is not marked as “unspecified”), the following applies.
      - If the reference field has the same parity as the current field

        LongTermPicNum = 2 * LongTermFrameIdx + 1 (G-610)

      - Otherwise (the reference field has the opposite parity of the current field),

        LongTermPicNum = 2 * LongTermFrameIdx (G-611)

G.8.13 Decoded reference picture marking process

[Ed. (HS): We need to add some definitions in F.3 which specify “base pictures” and a collective term for “base picture” and “normal” decoded pictures. The current text of this subclause uses some ad-hoc terms for “base representations”, which are marked as “used for reference” but are not output. – When these definitions are added, the wording of this subclause needs to be modified.]

This process is invoked for a decoded picture when nal_ref_idc is not equal to 0.

Input to this process are

- a variable dOiDXmax
- syntax elements of the slice header(s) with dependency_id equal to DependencyIdmax and quality_level equal to 0
- when dOiDXmax is equal to 0, the syntax element key_pic_flag and the syntax structure dec_ref_pic_marking_base( ) in the syntax structure(s) slice_layer_in_scalable_extension_rbsp( )

All syntax elements of the dec_ref_pic_marking( ) syntax structure and the dec_ref_pic_marking_base( ) syntax structure that are used in this subclause, the subclauses that are invoked inside this subclause, and all further invocations are syntax elements of the slice header(s) with dependency_id equal to DependencyIdmax and quality_level equal to 0, and, when dOiDXmax is equal to 0, the syntax element key_pic_flag and the syntax structure dec_ref_pic_marking_base( ) in the syntax structure(s) slice_layer_in_scalable_extension_rbsp( ).

NOTE – Slice data NAL units with quality_level greater than 0 do not contain the syntax structure dec_ref_pic_marking( ) or dec_ref_pic_marking_base( ). The syntax elements specifying the decoded reference picture marking are inferred from slice data NAL units with dependency_id equal to DependencyIdmax and quality_level equal to 0.

A decoded picture with nal_ref_idc not equal to 0, referred to as a reference picture, is marked as “used for short-term reference” or "used for long-term reference". When store_base_rep_falg is equal to 1 and quality_level is greater than 0, a second representation of the decoded picture also referred to as decoded base picture is marked as “used for short-term reference” or “used for long-term reference” and additionally marked as “base representation”. Pictures that are marked as “base representation” are only used as references for Inter prediction of following pictures with use_base_representation_flag equal to 1. These pictures are not used for Inter prediction of pictures with use_base_representation_flag equal to 0, and these pictures do not represent an output of the decoding process.

For a decoded reference frame, both of its fields are marked the same as the frame. For a complementary reference field pair, the pair is marked the same as both of its fields. A picture that is marked as "used for short-term reference" is identified by its FrameNum and, when it is a field, by its parity, and, when it is a decoded base picture, by the marking “base representation”. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx and, when it is a field, by its parity, and, when it is a decoded base picture, by the marking “base representation”.

While decoded pictures are represented by the sample arrays S_L and, when chroma_format_idc is not equal to 0, S_{Cb} and S_{Cr}, decoded base pictures are represented by the sample arrays B_L and, when chroma_format_idc is not equal to 0, B_{Cb} and B_{Cr}. When decoded base pictures are referenced in the Inter prediction process via subclause 8.4.2.1, the samples arrays B_L, B_{Cb}, and B_{Cr} are referred to as S_L, S_{Cb}, and S_{Cr}, respectively.

Frames or complementary field pairs marked as “used for short-term reference” or as "used for long-term reference" can be used as a reference for inter prediction when decoding a frame until the frame, the complementary field pair, or one of its constituent fields is marked as “unused for reference”. A field marked as “used for short-term reference” or as "used for long-term reference" can be used as a reference for inter prediction when decoding a field until marked as “unused for reference”.

A picture can be marked as "unused for reference" by the sliding window reference picture marking process, a first-in, first-out mechanism specified in subclause 8.2.5.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.5.4.

A short-term reference picture is identified for use in the decoding process by its variables FrameNum and FrameNumWrap and its picture number PicNum, and, when it is a decoded base picture, and by the marking as “base representation”. A long-term reference picture is identified for use in the decoding process by its long-term picture number LongTermPicNum, and, when it is a decoded base picture, by the marking as “base representation”. When the current picture is not an IDR picture, subclause 8.2.4.1 is invoked with DependencyIdmax as input to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum [Ed. (HS): Here, the frame_num’s of slice headers with dependency_id equal to DependencyIdmax shall be used. This invocation needs to be checked carefully, it might be necessary to invoke the corresponding subclause in Annex F and to modify this subclause].

Decoded reference picture marking proceeds in the following ordered steps.

- All slices of the current access unit are decoded.
- Depending on EnhIdr, the following applies.
  - If EnhIdr is equal to 1, the following applies.
    - All reference pictures shall be marked as “unused for reference”.
    - Depending on long_term_reference_flag, the following applies.
      - If long_term_reference_flag is equal to 0, the following applies.
        - The IDR picture shall be marked as “used for short-term reference” and MaxLongTermFrameIdx shall be set equal to “no long-term frame indices”
        - When store_base_rep_flag is equal to 1 and quality_level is greater than 0, the decoded base picture of the IDR picture shall be marked as “used for short-term reference” and as “base representation”.
      - Otherwise (long_term_reference_flag is equal to 1), the following applies.

- The IDR picture shall be marked as "used for long-term reference", the LongTermFrameIdx for the IDR picture shall be set equal to 0, and MaxLongTermFrameIdx shall be set equal to 0.
- When store_base_rep_flag is equal to 1 and quality_level is greater than 0, the decoded base picture of the IDR picture shall be marked as “used for long-term reference” and as “base representation”, and the LongTermFrameIdx for the IDR picture shall be set equal to 0.

– Otherwise (EnhIdr is equal to 0), the following applies.

  – When use_base_representation_flag is equal to 1, the following applies.

    – A variable tempAdaptiveMarking is set to 0.

    – When base_adaptive_ref_pic_marking_mode_flag is equal to 1, the following applies.

      – The variable tempAdaptiveMarking is set to 1.

      – The adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 is invoked with the syntax elements from the dec_ref_pic_marking_base( ) syntax structure and base_adaptive_ref_pic_marking_mode_flag replacing adaptive_ref_pic_marking_mode_flag. Only the reference pictures that are marked as “base representation” shall be considered in the invocation of the process. When during the invocation of this process an identified reference picture to be marked as “unused for reference” is not present, the variable tempAdaptiveMarking is set to 0.

    – When the variable tempAdaptiveMarking is equal to 0, the following applies.

      – The sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked with base_adaptive_ref_pic_marking_mode_flag replacing adaptive_ref_pic_marking_mode_flag. When during the invocation of this process there is a decoded picture not marked as "base representation" and a decoded base picture marked as "base representation", and both of which have the smallest value of FrameNumWrap, then only the picture marked as "base representation" shall be marked as “unused for reference”.

  – When quality_level is greater than 0, the decoded base picture of the current decoded picture shall be marked as “used for short-term reference” and as “base representation”.

  – If adaptive_ref_pic_marking_mode_flag is equal to 0, the following applies.

    – The sliding window decoded reference picture marking process as specified in subclause 8.2.5.3 is invoked. When during the invocation of this process there is a decoded picture not marked as "base representation" and a decoded base picture marked as "base representation", and both of which have the smallest value of FrameNumWrap, then only the picture not marked as "base representation" shall be marked as “unused for reference”..

    – The current decoded picture is marked as “used for short-term reference”.

  – Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the following applies.

    – The adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 with the syntax elements from the dec_ref_pic_marking( ) syntax structure is invoked. Only the reference pictures that are not marked as “base representation” shall be considered in the invocation of the process. All pictures for which LongTermFrameIdx is greater than MaxLongTermFrameIdx and that are marked as "used for long-term reference" and “base representation” are marked as “unused for reference”. When during the invocation of this process a LongTermFrameIdx is assigned to a picture and there exists a decoded base picture with an identical value of PicNum (before assigning the long-term frame index), the LongTermFrameIdx is also assigned to this decoded base picture, and the marking is changed from “used for short-term reference” to “used for long-term reference”.

    – When the current picture was not marked as "used for long-term reference" by memory_management_control_operation equal to 6, it is marked as "used for short-term reference".

NOTE – When use_base_representation_flag is equal to 1 and quality_level is greater than 0, num_ref_frames shall not be less than 2.

NOTE – When the adaptive memory control decoded reference picture marking process as specified in subclause 8.2.5.4 is invoked to mark reference pictures as “unused for reference” after decoding of a reference picture with temporal_level equal to tL, no reference picture with temporal_level less than tL shall be marked as “unused for reference”.

G.8.14 Deblocking filter process in scalable extension

The deblocking filter process as described in 8.7 is invoked, except that the invocation of the derivation process for the luma content dependent boundary filtering strength described in subclause 8.7.2.1 shall be replaced by the invocation of the extended derivation process for the luma content dependent boundary filtering strength described in subclause G.8.14.1. [Ed. (JR): any idea to have this written in a more cleared way is welcome.]

G.8.14.1 Extended derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p_0 and q_0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

The variable bS is derived as follows.

- If SpatialScalabilityType is not equal to 0 and either luma sample p_0 or q_0 is in a macroblock coded using the I_BL macroblock prediction mode, the variable bS is derived as follows.
  - If either luma samples p_0 or q_0 is in a macroblock coded using an intra prediction mode other than the I_BL mode, a value of bS equal to 4 shall be the output.
  - Otherwise, if all of the following conditions are true, a value of bS equal to 1 shall be the output.
    - the luma samples p_0 or q_0 are both in macroblocks coded using the I_BL macroblock prediction mode
    - [Ed (JR) : this condition is not needed: “the luma samples p_0 and q_0 are in the same slice or in two separate slices with same base_id.”]
    - the luma block containing sample p_0 or the luma block containing sample q_0 contains non-zero transform coefficient levels.
  - Otherwise, if one of the following conditions is true, a value of bS equal to 2 shall be the output.
    - the luma block containing sample p_0 or the luma block containing sample q_0 contains non-zero transform coefficient levels.
    - the syntax element nal_unit_type is equal to 20 and residual_prediction_flag is equal to 1 for the luma block containing sample p0 or the luma block containing sample q0 and the prediction array $picRes_X$ as derived in subclause G.8.10 contains non-zero samples, with X indicating the applicable component L, Cb, or Cr.
  - Otherwise, if one of the following conditions is true, a value of bS equal to 1 shall be the output.
    - either luma samples p_0 or q_0 is in a macroblock coded using an inter prediction mode.
    - [Ed (JR) : this condition s not needed: “the luma samples p_0 and q_0 are in two separate slices with different base_id.”]
  - Otherwise, a value of bS equal to 0 shall be the output.
- Otherwise, if the samples p_0 and q_0 are both in macroblocks coded using the I_BL macroblock prediction mode [Ed. (JR) useless test, because not allowed: “and the luma samples p_0 and q_0 are in the same slice or in two separate slices with same base_id,”] a value of bS equal to 1 shall be the output:
- Otherwise the following applies
  - The derivation process for the luma content dependent boundary filtering strength described in subclause 8.7.2.1 is invoked with the sample values p_0 and q_0 and verticalEdgeFlag as input and bS as output
  - If all the following conditions are true, a value of bS equal to 2 is the output:
    - bS is smaller than 2
    - the syntax element nal_unit_type is equal to 20 and residual_prediction_flag is equal to 1 for the luma block containing sample p0 or the luma block containing sample q0 and the prediction array $picRes_X$ as derived in subclause G.8.10 contains non-zero samples, with X indicating the applicable component L, Cb, or Cr.

- Otherwise the value of bS is the output.

G.9 Parsing process

Inputs to this process are bits from the RBSP, a request for a value of a syntax element, and values of prior parsed syntax elements (if applicable).

Output of this process is the value of the syntax element.

This process is invoked for all syntax elements in the syntax tables in subclause G.7.3 with descriptors equal to ue(v), me(v), se(v), te(v), ce(v), ge(v), and ae(v).

When the parsing process is invoked for the first request for a value of a syntax element in the slice data, the following applies.

- If entropy_coding_mode_flag is equal to 0 and the slice_type is equal to PR, the initialisation process of the CAVLC parsing process for progressive refinement slices is invoked as specified in subclause G.9.1.
- Otherwise, if entropy_coding_mode_flag is equal to 1, the initialisation process of the CABAC parsing process is invoked by the following two ordered steps:
  a) The initialisation process as specified in subclause 9.3.1 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, a slice_type equal to EB is interpreted as B, and a slice_type equal to PR is interpreted as P.
  b) The initialisation process as specified in subclause G.9.2.1 is invoked.
- Otherwise (entropy_coding_mode_flag is equal to 0 and the slice_type is not equal to PR), no further initialisation process is needed.

Depending on the descriptor, the value of a syntax element is derived as follows.

- If the syntax element is coded as ue(v), me(v), se(v), or te(v), the value of the syntax element is derived by invoking the process as specified in subclause 9.1.
- Otherwise, if the syntax element is coded as ce(v), the value of the syntax element is derived by invoking the process as specified in subclause 9.2.
- Otherwise, if the syntax element is coded as ge(v), the value of the syntax element is derived by invoking the process as specified in subclause G.9.1.
- Otherwise (the syntax element is coded as ae(v) and entropy_coding_mode_flag is equal to 1), the value of the syntax element is derived as follows.
  - Let the variable alternativeCabacParsingFlag be derived as follows.
    - If any of the following conditions is true, alternativeCabacParsingFlag is set equal to 1.
      - The syntax element is equal to base_mode_flag, motion_prediction_flag_l0, motion_prediction_flag_l1, residual_prediction_flag, cycle_start_symbol, cycle_start_symbol_ext, coded_block_pattern_bit, coeff_sign_flag, coeff_refinement_flag, or coeff_refinement_direction_flag.
      - The slice_type is equal to PR and the syntax element is equal to significant_coeff_flag, last_significant_coeff_flag, or coeff_abs_level_minus1 [Ed. (DM) Case of coded_block_flag has to be checked].
    - Otherwise, alternativeCabacParsingFlag is set equal to 0.
  - Depending on the value of alternativeCabacParsingFlag, the following applies.
    - If alternativeCabacParsingFlag is equal to 1, the following two ordered steps are carried out.
      i. The binarization process as specified in subclause G.9.2.2 is invoked.
      ii. The decoding process flow as specified in G.9.2.3 is invoked.
    - Otherwise (alternativeCabacParsingFlag is equal to 0), the following ordered steps are carried out.

i. The binarization process as specified in subclause 9.3.2 is invoked, where a slice_type equal to EI is interpreted as I, a slice_type equal to EP is interpreted as P, a slice_type equal to EB is interpreted as B, and a slice_type of PR interpreted as P.

ii. The decoding process flow as specified in subclause 9.3.3 is invoked.

iii. When the syntax element is equal to mb_type and the decoded value of mb_type is equal to I_PCM, the arithmetic decoding engine shall be initialised after decoding of any pcm_alignment_zero_bit and all pcm_sample_luma and pcm_sample_chroma data as specified in subclause 9.3.1.2 [Ed. (DM) Do we allow I_PCM together with EI?].

G.9.1 CAVLC parsing process for progressive refinement slices

[Ed. (JR) The description has not been checked.]

This process is invoked when parsing syntax elements with descriptor ge(v) in subclauses G.7.3.7.4, G.7.3.7.5, G.7.3.7.6, G.7.3.7.8, and G.7.3.7.12 when entropy_coding_mode_flag is equal to zero.

Inputs to this process are a request for a value of a syntax element and values of prior parsed syntax elements.

Output of this process is the value of the syntax element.

When starting the parsing of the slice data of a slice in subclause G.7.3.7, the initialisation process of the CAVLC parsing process is invoked as specified in subclause G.9.1.1. Separate instances of the internal variables are maintained for each of the syntax elements coded_block_pattern_bit, coded_block_flag_luma, coded_block_flag_chromaAC, coeff_refinement_flag, and coeff_refinement_direction_flag.

For syntax elements of type coded_block_flag_luma, the CAVLC parsing process is invoked as specified in subclause G.9.1.4.

For syntax elements of type coded_block_flag_chromaAC, the CAVLC parsing process is invoked as specified in subclause G.9.1.5.

For syntax elements of type coeff_sig_vlc_symbol, the CAVLC parsing process is invoked as specified in subclause G.9.1.6.

For syntax elements of type coeff_refinement_flag, the CAVLC parsing process is invoked as specified in subclause G.9.1.7

For syntax elements of type coeff_refinement_direction_flag, the CAVLC parsing process is invoked as specified in subclause XXX

G.9.1.1 Initialisation process

Outputs of this process are initialised CAVLC internal variables.

Table G-9 contains the values used to initialise the CAVLC internal variables for each syntax element in subclauses G.7.3.7.4, G.7.3.7.5, and G.7.3.7.6.

Table G-9 - Initialisation values for CAVLC internal variables for coded_block_flag_luma and coded_block_flag_chromaAC

CAVLC internal variable name	value of internal vartiable
scaleFactor	1
scaleLimit	512
vlcTable	0
flipFlag	0
symCount0	0
symCount1	0

Table G-10 Table G-10contains the values used to initialise the CAVLC internal variables for parsing syntax element coeff_ref_vlc_symbol G.7.3.7.12.

Table G-10 - Initialisation values for CAVLC internal variables for coeff_ref_vlc_symbol

CAVLC internal variable name	Value of internal variable
refVlcSelector	0
refSymCount0	0
refSymCount1	0
refSymCount2	0
refCoeffIdx	3

G.9.1.2 Bit reading process for coded_block_flag_luma and coded_block_flag_chromaAC

Input to this process is maxBits.

Output of this process is a bit value bBit.

Note: only the instances of the internal variable symCount0, symCount1, bufBits, bufLength corresponding to the syntax element currently being decoded are modified in this subclause.

When the value of internal variable bufLength is equal to zero, the following applies:

- A variable rBits is decoded using one of the three VLCs specified in Table G-11 depending on the value of the internal variable vlcTable.
- When the internal variable flipFlag is set to 1, each bit in the variable rBits is inverted.
- For each bit in the variable rBits the following applies
  - If the bit is equal to zero, the internal variable symCount0 is incremented by 1
  - Otherwise (the bit is equal to 1), symCount1 is incremented by 1.
- Let vlcSize be the number of bits in variable rBits.
- The internal variable bufLength is incremented by vlcSize.

$$bufLength = bufLength + vlcSize \tag{G-612}$$

- The internal variable bufBits is left-shifted by vlcSize and incremented by the value of the variable rBits.

$$bufBits = (bufBits << vlcSize) + rBits \tag{G-613}$$

- The value of flipFlag is computed as follows:
  - If symCount0 is less than symCount1, the value of internal variable flipFlag is set to 1,
  - Otherwise it is set to 0.
- Let the variable allowedVlcIdx[ ] be set depending on the value of maxBits as specified in Table G-12.
- The internal variable vlcTable is updated according to Table G-13, where the value scA and scB are determined as follows
  - If flipFlag is equal to zero, the value of symCount0 is used for scA and the value of symCount1 is used for scB.
  - Otherwise (flipFlag is equal to 1), the value of symCount1 is used for scA and the value of symCount0 is used for scB.
- When the sum of symCount0 and symCount1 is greater than the value of internal variable scaleLimit, both of symCount0 and symCount1 are right-shifted by a number of bits equal to the value of internal variable scaleFactor.

symCount0 = symCount0 >> scaleFactor (G-614)
symCount1 = symCount1 >> scaleFactor (G-615)

The bit value bBit is set equal to the most significant bit of bufBits.

The variable bufLength is decremented by 1.

The most significant bit of the variable bufBits is removed.

Table G-11 – Variable length codes used in parsing progressive refinement syntax elements

rBits (return value)	vlcTable = 0	vlcTable = 1
0	0	-
1	1	-
0000	-	0
0001	-	100
0010	-	101
0011	-	11100
0100	-	1100
0101	-	11101
0110	-	111100
0111	-	11111100
1000	-	1101
1001	-	111101
1010	-	1111100
1011	-	11111101
1100	-	1111101
1101	-	11111110
1110	-	111111110
1111	-	111111111

Table G-12 - Derivation of variable allowedVlcIdx[ ]

maxBits < 4	maxBits = 4
{ 0 }	{ 0, 1 }

Table G-13 - Update table for variable vlcTable for coded_block_flag_luma and coded_block_flag_chromaAC

allowedVlcIdx[ ]	scA < 2 * scB	2 * scB <= scA
{ 0, 1 }	0	1
{ 0 }	0	0

G.9.1.3 Derivation process for coded block flag syntax elements

Inputs to this process are

– cbfCurrent[ ],

– the existing value of the coded block flag values for each block in the current sub-macroblock,

– the cbfBase[ ],

– the base layer coded block flag values of each block in the current sub-macroblock.

Output of this process is an updated value of the coded block flag, cbfCurrent.

When cbfCurrent is equal to NotCoded, the new value of cbfCurrent is derived as follows:

1. Let the variable cbpValue be set equal to 0.
2. Let the variable blkIdx be set equal to 0.
3. Let the variable baseCbfCount be set equal to 0.
4. When the value of cbfBase[blkIdx] is equal to the value of cbpValue, baseCbfCount is incremented by 1.
5. The variable blkIdx is incremented by 1.
6. When the value of variable blkIdx is less than 4, the derivation process continues from step 4.
7. The value of variable blkIdx is set equal to 0.
8. When the value of cbfBase[blkIdx] is equal to the value of cbpValue, the process of subclause G.9.1.2 is invoked with the value of baseCbfCount as input and of the output is assigned to cbfCurrent[blkIdx]
9. The process of subclause G.9.1.3 is invoked
10. The variable baseCbfCount is decremented by 1.
11. The variable blkIdx is incremented by 1.
12. When the value of variable blkIdx is less than 4, the derivation process continues from step 8.
13. The value of cbpValue is incremented by 1.
14. When cbpValue is less than 2, the derivation process continues from step 2.

G.9.1.4 Derivation process for the syntax element coded_block_flag_luma

Inputs to this process are:

– a vector of coded_block_flag_luma values for the current sub-macroblock,

– a vector cbfBase containing luma coded block flag values for the corresponding sub-macroblock in the base layer,

– a sub-macroblock index i4x4.

Output of this process is the coded_block_flag_luma[i4x4] syntax element.

The derivation process of subclause G.9.1.3 is invoked, with cbfCurrent[ ] set to the values of coded_block_flag_luma[ ], and cbfBase[ ] set to the luma coded block flag values of the base layer.

G.9.1.5 Derivation process for the syntax element coded_block_flag_chromaAC

Inputs to this process are:

– a vector of coded_block_flag_chromaAC values for the current sub-macroblock,

– a vector cbfBase containing chroma AC coded block flag values for the corresponding sub-macroblock in the base layer,

– a component index iCbCr,

– a sub-macroblock index i4x4.

Output of this process is the coded_block_flag_chromaAC[iCbCr][i4x4] syntax element.

The derivation process of subclause G.9.1.3 is invoked, with cbfCurrent[ ] set to the values of coded_block_flag_chromaAC[iCbCr][ ], and cbfBase[ ] set to the chroma AC coded block flag values of the base layer.

G.9.1.6 Derivation process for the syntax element coeff_sig_vlc_symbol

Inputs to this process is sigVlcSelector which is a codebook index

Output of this process is coeff_sig_vlc_symbol syntax element.

The VLC codebook to be used in decoding process is selected based on codebook index sigVlcSelector. The syntax element coeff_sig_vlc_symbol is decoded using the selected codebook.

Fives codebooks and their corresponding decoding processes are defined as follows, where leadingZeroBits is the number of leading zero bits and codeLength is the number of bits in a codeword. The process of section 9.1 shall be used to calculate leadingZeroBits. Let lastTwoBits be the last two bits of the codeword.

Codebook 0: codeLength = leadingZeroBits + 1,

coeff_sig_vlc_symbol = codeLength - 1;

Codebook 1: codeLength = leadingZeroBits + 2,

coeff_sig_vlc_symbol = codeLength * 2 – lastTwoBits - 1;

Codebook 2: codeLength = (leadingZeroBits >> 1) << 1) + 2,

coeff_sig_vlc_symbol = (codeLength >> 1) * 3 – lastTwoBits;

Codebook 3: codeLength = leadingZeroBits ? (((leadingZeroBits – 1) >> 1) << 1) + 3) : 1

coeff_sig_vlc_symbol = (codeLength = = 1) ? 0 : ((codeLength >> 1) * 3 – lastTwoBits + 1);

Codebook 4: codeLength = leadingZeroBits ? (leadingZeroBits + 2) : 1,

coeff_sig_vlc_symbol = (codeLength = = 1) ? 0 : (codeLength * 2 – lastTwoBits - 2);

The examples of codeword for each codebook are given in the Table G-14.

NOTE – Codebook 3 can be derived from codebook 2 by inserting codeword “1” at the beginning of the array, and add prefix bit “0” to all other codewords. Similarly, codebook 4 can be derived from codebook 1 by inserting codeword “1” at the beginning of the array, and add prefix bit “0” to all other codewords.

Table G-14 – Codebook table for coeff_sig_vlc_symbol depending on sigVlcSelector (informative)

coeff_sig_vlc_symbol	sigVlcSelector				
	0	1	2	3	4
0	1	11	11	1	1
1	01	10	10	011	011
2	001	011	01	010	010
3	0001	010	0011	001	0011
4	00001	0011	0010	00011	0010
5	000001	0010	0001	00010	00011
6	0000001	00011	000011	00001	00010
7	00000001	00010	000010	0000011	000011
8	.....	.....	.....	.....	.....

G.9.1.7 Derivation process for the syntax element coeff_refinement_ flag and coeff_refinement_direction_flag

Subclause G.9.1.7.1specifies the derivation process for the syntax element coeff_refinement_ flag.

Subclause G.9.1.7.2 specifies the derivation process for the syntax element coeff_refinement_direction_flag.

Subclause G.9.1.7.3 specifies the updating process parsing coeff_refinement_ flag and coeff_refinement_direction_flag

G.9.1.7.1 Derivation process for the syntax element coeff_refinement_ flag

Output of this process is coeff_refinement_ flag syntax element.

The coeff_refinement_ flag syntax element is derived as follows:

- When the CAVLC internal variable refCoeffIdx is equal to 3, the following applies
  - A local variable coeffRefVlcSymbolGroup is decoded using one of the two VLCs specified in Table G-15 depending on the value of the CAVLC internal variable refVlcSelector.
  - The first elelement of coeffRefVlcSymbolGroup is stored in coeffRefVlcSymbolBuf [0], the second elelement of coeffRefVlcSymbolGroup is stored in coeffRefVlcSymbolBuf [1], and the third elelement of coeffRefVlcSymbolGroup is stored in coeffRefVlcSymbolBuf [2].
  - The CAVLC internal variable refCoeffIdx is set to 0
- Depending on the value of coeffRefVlcSymbolBuf[refCoeffIdx] the following applies,
  - If coeffRefVlcSymbolBuf[refCoeffIdx] is equal to 0, the following applies,
    - coeff_refinement_ flag is set equal to 0
    - refSymCount0 is incremented by 1
    - The CAVLC internal variable refCoeffIdx is incremented by 1
  - Otherwise (coeffRefVlcSymbolBuf[refCoeffIdx] is not equal to 0), coeff_refinement_ flag is set equal to 1.

NOTE –In Table G-15, 3 coeff_ref_vlc_symbol's are listed from left to right in the decoding order. Take "0, 1, 2" as an example, coeff_ref_vlc_symbol of the first refinement coefficient is 0, and coeff_ref_vlc_symbol of the second refinement coefficient is 1, and coeff_ref_vlc_symbol of the third refinement coefficient is 2.

Table G-15 – Codeword tables for coeff_ref_vlc_symbol group

Group of 3 coeff_ref_vlc_symbol's: coeffRefVlcSymbolGroup	Bit string with refVlcSelector = 0	Bit string with refVlcSelector = 1
0, 0, 0	1	1
0, 0, 1	0011	00111
0, 0, 2	00101	00110
0, 1, 0	011	0111
0, 1, 1	000101	0001001
0, 1, 2	00000101	0001000
0, 2, 0	00100	0110
0, 2, 1	0000101	0000111
0, 2, 2	000000101	001001
1, 0, 0	010	0101
1, 0, 1	000100	0000110
1, 0, 2	00000100	0000101
1, 1, 0	000011	001000
1, 1, 1	000000100	00000111
1, 1, 2	0000000011	00000110

1, 2, 0	0000100	000111
1, 2, 1	0000000010	00000101
1, 2, 2	000000000011	00000100
2, 0, 0	00011	0100
2, 0, 1	0000011	0000100
2, 0, 2	000000011	000110
2, 1, 0	00000011	000101
2, 1, 1	0000000001	00000011
2, 1, 2	000000000010	00000010
2, 2, 0	000000010	00101
2, 2, 1	000000000001	00000001
2, 2, 2	000000000000	00000000

G.9.1.7.2 Derivation process for the syntax element coeff_refinement_direction_flag

Output of this process is coeff_refinement_direction_ flag syntax element.

The coeff_refinement_direction_ flag syntax element is derived as follows

– If coeffRefVlcSymbolBuf[refCoeffIdx] is equal to 1, coeff_refinement_direction_ flag is set equal to 0 and refSymCount1 is incremented by 1.

– Otherwise (coeffRefVlcSymbolBuf[refCoeffIdx] is equal to 2), coeff_refinement_direction_ flag is set equal to 1 and refSymCount2 is incremented by 1.

NOTE –This process is only called when coeffRefVlcSymbolBuf[refCoeffIdx] is not equal to 0.

– The CAVLC internal variable refCoeffIdx is incremented by 1

G.9.1.7.3 Updating process for parsing coeff_refinement_ flag and coeff_refinement_direction_flag

This process is invoked when finishing the parsing of a macroblock in subclause G.7.3.7.

Input of this process are the CAVLC internal variables.

Outputs of this process are updated CAVLC internal variables, refVlcSelector, refCoefIdx, refSymCount0, refSymCount1, and refSymCount2 corresponding to coeff_ref_vlc_symbol.

CAVLC internal variables are updated as follows.

– The internal variable refVlcSelector is updated as follows.

  – If refSymCount0 is larger or equal to 2 * (refSymCount1 + refSymCount2) and refSymCount1 is larger or equal than 2 * refSymCount2, refVlcSelector is set equal to 0.

  – Otherwise (refSymCount0 < 2 * (refSymCount1 + refSymCount2) or refSymCount1 < 2 * refSymCount2), refVlcSelector is set equal to 1.

– All of refSymCount0, refSymCount1 and refSymCount2 are right-shifted by 1 bit.

$$refSymCount0 = refSymCount0 >> 1 \quad \text{(G-616)}$$
$$refSymCount1 = refSymCount1 >> 1 \quad \text{(G-617)}$$
$$refSymCount2 = refSymCount2 >> 1 \quad \text{(G-618)}$$

– refCoeffIdx is set equal to 3.

G.9.2 Alternative CABAC parsing process for slice data in scalable extension

Subclause G.9.2.1 specifies the initialisation process for the alternative CABAC parsing process for slice data in scalable extension.

Subclause G.9.2.2 specifies the binarization process for the alternative CABAC parsing process for slice data in scalable extension.

Subclause G.9.2.3 specifies the decoding process flow for the alternative CABAC parsing process for slice data in scalable extension.

G.9.2.1 Initialisation process

[Ed. (DM) This process should handle only those syntax elements that are not treated by the existing initialisation process in subclause 9.3.1]

The specifications in subclause 9.3.1 shall apply. Additionally the Table G-16 is added to the Table 9-11 for the initialization of the ctxIdx for the slice type related to the SVC extension.

Table G-17 and Table G-18 contain additional values of the variables n and m used in the initialisation of context variables that are assigned to all syntax elements in subclauses 7.3.4 and 7.3.5 except for the end-of-slice flag.

Table G-16 - Association of ctxIdx and syntax elements for each slice type in the initialisation process [Ed (JR) modified Table 9-11 to add EI, EP, and EB slice type, note that syntax elements existing in I, P, B slice and used in PR slice should also have an entry in the table]

	Syntax element	Table	Slice type			
			PR	EI	EP	EB
macroblock_layer_in_scalable_extension ( )	base_mode_flag	Table G-17		460-462	460-462	460-462
mb_pred_in_scalable_extension( ) and sub_mb_pred_in_scalable_extension( )	motion_prediction_flag_l0	Table G-17			463	463
	motion_prediction_flag_l1	Table G-17			463	463
residual_in_scalable_extension( )	residual_prediction_flag	Table G-18			467-470	467-470
	smoothed_reference_flag	Table G-17			464	464
luma_coefficient() and chromaDC_coefficient() and chromaAC_coefficient()	coded_block_pattern_bit[]	Table 9-18	73-76			
significant_coefficient_cabac()	significant_coeff_flag	Table G-18	471-482			
luma_coefficient_ref ( ) and chromaDC_coefficient_ref () and chromaAC_coefficient_ref ()	coeff_refinement_flag	Table G-17	465			
	coeff_refinement_direction_flag	Table G-17	466			

Table G-17 - Values of variables m and n for ctxIdx from 460 to 466 [Ed. Note(HS): The initialization values currently represent a simple uniform distribution. Suitable initialization values still need to be determined.]

Initialisation variables	ctxIdx						
	460	461	462	463	464	465	466
m	0	0	0	0	0	0	0
n	64	64	64	64	40	64	64

Table G-18 - Values of variables m and n for ctxIdx from 467 to 482 [Ed. Note(HS): The initialization values are just copied from the corresponding context with levelListIdx-1. Suitable initialization values still need to be determined.]

ctxIdx	I or SI base layer slices [Ed. Note(HS): That needs to be changed]		Value of cabac_init_idc					
			0		1		2	
	m	N	m	n	m	n	m	n
467	0	100	0	100	0	100	0	100
468	0	80	0	80	0	80	0	80
469	0	50	0	50	0	50	0	50
470	0	50	0	50	0	50	0	50
471	-2	78	-2	76	-46	127	-1	74
472	14	62	-7	100	-44	127	-5	90
473	18	59	13	68	18	55	13	91
474	0	95	9	63	7	61	8	77
475	12	72	9	69	0	89	-22	127
476	-11	74	-20	63	2	59	-11	68
477	3	68	-5	91	-8	95	-10	99
478	9	64	17	54	7	69	15	57
479	-1	83	6	64	-2	75	2	70
480	2	95	6	70	14	60	12	70
481	12	70	26	43	18	50	25	42
482	2	59	2	59	2	58	-11	68

G.9.2.2 Binarization process

[Ed. (DM) Needs some further editing work]

The specifications in subclause 9.3.2 shall apply with the following additions.

The possible values of the context index ctxIdx are in the range 0 to 482, inclusive.

Associated with each binarization or binarization part of a syntax element of the scalable extension is a specific value of the context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in Table G-19 and Table 9-25.

The specification of the signed truncated unary (STU) binarization process is given in subclauses G.9.2.2.1.

Table G-19 - Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset [Ed. (DM) Table is incomplete due to missing data for last_significant_coeff_flag, coeff_abs_level_minus1 for PR slices.]

Syntax element	Type of binarization	maxBinIdxCtx	ctxIdxOffset
base_mode_flag	FL, cMax=1	0	460

motion_prediction_flag_l0, motion_prediction_flag_l1	FL, cMax=1	0	463
residual_prediction_flag	FL, cMax=1	0	467
smoothed_reference_flag	FL, cMax=1	0	464
cycle_start_symbol	FL, cMax=1	na	na,(uses DecodeBypass)
cycle_start_symbol_ext	FL, cMax=1	na	na,(uses DecodeBypass)
coded_block_pattern_bit	FL, cMax=1	3	73
coeff_refinement_flag	FL, cMax=1	0	465
coeff_refinement_direction_flag	FL, cMax=1	0	466
significant_coeff_flag (PR slices)	FL, cMax=1	0	471
coeff_sign_flag (PR slices)	FL, cMax=1	na	na,(uses DecodeBypass)

G.9.2.2.1 Signed truncated unary (STU) binarization process

Input to this process is a request for a STU binarization for a syntax element and cMax.

Output of this process is the STU binarization of the syntax element.

A STU bin string is a concatenation of a prefix bit string and (when present) a suffix bit string. The prefix of the binarization is specified by invoking the TU binarization process for the prefix part Abs(synElVal) of a syntax element value synElVal as specified in subclause 9.3.2.2.

The STU bin string is derived as follows.

– If the prefix bit string is equal to the bit string that consists of a single bit with value equal to 0, the bin string of a syntax element having value synElVal consists only of a prefix bit string,

– Otherwise, the bin string of the STU suffix part of a syntax element value synElVal consists of a single bit which is specified by bs = ((synElVal > 0) ? 0 : 1).

G.9.2.3 Decoding process flow

[Ed. (DM) Needs some further editing work]

The specifications in subclause 9.3.3 shall apply except that the subclause G.9.2.3.1 shall be invoked instead of subclause 9.3.3.1.

G.9.2.3.1 Derivation process for ctxIdx

Inputs to this process are binIdx, maxBinIdxCtx and ctxIdxOffset.

Output of this process is ctxIdx.

Table G-20 and Table 9-30 show the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, coded_block_flag_luma, coded_block_flag_chromaAC, coded_block_flag_chromaDC, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxOffset associated with the given bin string or part thereof. The ctxIdx is determined as follows.

– If the ctxIdxOffset is listed in Table G-20 or Table 9-30, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table G-20 or Table 9-30. When more than one value is listed in Table G-20 or Table 9-30 for a binIdx, the assignment process for ctxIdxInc for that binIdx is further specified in the subclauses given in parenthesis of the corresponding table entry.

– Otherwise (ctxIdxOffset is not listed in Table G-20 or Table 9-30), the ctxIdx is specified to be the sum of the following terms: ctxIdxOffset and ctxIdxBlockCatOffset(ctxBlockCat) as specified in Table 9-31 and

ctxIdxInc(ctxBlockCat). Subclause G.9.2.3.3 specifies which ctxBlockCat is used. Subclause 9.3.3.1.1.9 specifies the assignment of ctxIdxInc(ctxBlockCat) for coded_block_flag, coded_block_flag_luma, coded_block_flag_chromaAC and coded_block_flag_chromaDC, and subclause G.9.2.3.3 specifies the assignment of ctxIdxInc(ctxBlockCat) for significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1.

All bins with binIdx greater than maxBinIdxCtx are parsed using ctxIdx assigned to maxBinIdxCtx.

All entries in Table G-20 and Table 9-30 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

ctxIdx = 276 is assigned to the binIdx of mb_type indicating the I_PCM mode. For parsing the value of the corresponding bins from the bitstream, the arithmetic decoding process for decisions before termination as specified in subclause 9.3.3.2.4 is applied.

Table G-20 - Assignment of ctxIdxInc to binIdx for the ctxIdxOffset values related to the syntax elements base_mode_flag and residual_prediction_flag

ctxIdxOffset	binIdx						
	0	**1**	**2**	**3**	**4**	**5**	**>= 6**
460	0,1,2 (subclause G.9.2.3.2.2)	na	na	na	na	na	Na
469	0,1 (subclause G.9.2.3.4)	na	na	na	na	na	Na

G.9.2.3.2 Assignment process of ctxIdxInc using neighbouring syntax elements

Subclause G.9.2.3.2.1 specifies the derivation process of ctxIdxInc for the syntax element coded_block_pattern_bit.

Subclause G.9.2.3.2.1 specifies the derivation process of ctxIdxInc for the syntax element base_mode_flag.

G.9.2.3.2.1 Derivation process of ctxIdxInc for the syntax element coded_block_pattern_bit

The specification of subclause 9.3.3.1.1.4 shall apply. [Ed. (JR) check if this is sufficient (DM) I would say: No!]

G.9.2.3.2.2 Derivation process of ctxIdxInc for the syntax element base_mode_flag

Output of this process is ctxIdxInc.

When MbaffFrameFlag is equal to 1 and mb_field_decoding_flag has not been decoded (yet) for the current macroblock pair with top macroblock address 2 * (CurrMbAddr / 2), the inference rule for the syntax element mb_field_decoding_flag as specified in subclause 7.4.4 shall be applied.

The derivation process for neighbouring macroblocks specified in subclause 6.4.8.1 is invoked and the output is assigned to mbAddrA and mbAddrB.

Let the variable condTermFlagN (with N being either A or B) be derived as follows.

– If mbAddrN is not available or base_mode_flag for the macroblock mbAddrN is equal to 1, condTermFlagN is set equal to 0.

– Otherwise (mbAddrN is available and base_mode_flag for the macroblock mbAddrN is equal to 0), condTermFlagN is set equal to 1.

The variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{condTermFlagA} + \text{condTermFlagB} \qquad \text{(G-619)}$$

G.9.2.3.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-33.

Let the variable levelListIdx be set equal to the index of the list of transform coefficient levels as specified in subclause 7.4.5.3.

If any of the following conditions is true, Table G-21 contains the specification of ctxIdxInc for the given values of levelListIdx.

– ctxBlockCat < 5, ctxBlockCat != 3, and levelListIdx = = maxNumCoeff – 1

– ctxBlockCat = = 3 and levelListIdx = = 4 * NumC8x8 – 1

– ctxBlockCat = = 5 and levelListIdx = = 63

– Otherwise, the following applies.

– For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat < 5 and ctxBlockCat != 3, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{levelListIdx} \tag{G-620}$$

where levelListIdx ranges from 0 to maxNumCoeff – 2, inclusive.

– For the syntax elements significant_coeff_flag and last_significant_coeff_flag in blocks with ctxBlockCat = = 3, the variable ctxIdxInc is derived by

$$\text{ctxIdxInc} = \text{Min}(\text{levelListIdx} / \text{NumC8x8}, 2) \tag{G-621}$$

where levelListIdx ranges from 0 to 4 * NumC8x8 – 2, inclusive.

– For the syntax elements significant_coeff_flag and last_significant_coeff_flag in 8x8 luma blocks with ctxBlockCat = = 5, Table 9-34 contains the specification of ctxIdxInc for the given values of levelListIdx, where levelListIdx ranges from 0 to 62, inclusive.

Table G-21 - Determination of ctxIdxInc for special values of levelListIdx for the syntax element significant_coeff_flag depending on field_pic_flag, mb_field_decoding_flag, and ctxBlockCat

field_pic_flag || mb_field_decoding_flag	ctxBlockCat	ctxIdxInc
0	0	365
0	1	351
0	2	338
0	3	324
0	4	322
0	5	73
1	0	199
1	1	185
1	2	172
1	3	158
1	4	156
1	5	45

Let the variables numDecodAbsLevelEq1 and numDecodAbsLevelGt1 be derived as follows.

– numDecodAbsLevelEq1 and numDecodAbsLevelGt1 are set to zero.

– If binIdx is equal to 0, ctxIdxInc is derived by

ctxIdxInc = ((numDecodAbsLevelGt1 != 0) ? 0: Min(4, 1 + numDecodAbsLevelEq1)) (G-622)

– Otherwise (binIdx is greater than 0), ctxIdxInc is derived by

ctxIdxInc = 5 + Min(4 – (ctxBlockCat = = 3), numDecodAbsLevelGt1) (G-623)

G.9.2.3.4 Derivation process of ctxIdxInc for the syntax element residual_prediction_flag

Output of this process is ctxIdxInc.

Depending on slice_type, the following applies.

– If slice_type is not equal to 3 (PR, progressive refinement slice), the variable ctxIdxInc is derived as follows.
  – If base_mode_flag is equal to 1, ctxIdxInc is set equal to 0.
  – Otherwise (base_mode_flag is equal to), ctxIdxInc is set equal to 1.

– Otherwise (slice_type is equal to 3), the following applies.
  – Let currQualityLevel be equal to quality_level.
  – Let baseMbSet be the set of macroblocks with macroblock address equal to mbAddr, and quality_level less than currQualityLevel.
  – The variable hasBaseCoeffs is derived as follows.
    – If CodedBlockPatternLuma or CodedBlockPatternChroma of any of the macroblocks of the set baseMbSet is not equal to 0, hasBaseCoeffs is set equal to 1.
    – Otherwise (CodedBlockPatternLuma and CodedBlockPatternChroma for all macroblocks of the set baseMbSet are equal to 0), has BaseCoeffs is set equal to 0.
  – Depending on hasBaseCoeffs, ctxIdxInc is derived as follows.
    – If hasBaseCoeffs is equal to 1, ctxIdxInc is set equal to 2.
    – Otherwise (hasBaseCoeffs is equal to 0), ctxIdxInc is set equal to 3.

G.10 Supplemental enhancement information

G.10.1 SEI payload syntax

G.10.1.1 Scalability information SEI message syntax

scalability_info(payloadSize) {	C	Descriptor
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i]	5	u(8)
priority_id[i]	5	u(6)
discardable_flag[i]	5	u(1)
temporal_level[i]	5	u(3)
dependency_id[i]	5	u(3)
quality_level[i]	5	u(2)
sub_pic_layer_flag[i]	5	u(1)
sub_region_layer_flag[i]	5	u(1)
iroi_slice_division_info_present_flag[i]	5	u(1)
profile_level_info_present_flag[i]	5	u(1)
bitrate_info_present_flag[i]	5	u(1)
frm_rate_info_present_flag[i]	5	u(1)
frm_size_info_present_flag[i]	5	u(1)
layer_dependency_info_present_flag[i]	5	u(1)
init_parameter_sets_info_present_flag[i]	5	u(1)
exact_interlayer_pred_flag[i]	5	u(1)
if (profile_level_info_present_flag[i]) {		
layer_profile_idc[i]	5	u(8)
layer_constraint_set0_flag[i]	5	u(1)
layer_constraint_set1_flag[i]	5	u(1)
layer_constraint_set2_flag[i]	5	u(1)
layer_constraint_set3_flag[i]	5	u(1)
reserved_zero_4bits /* equal to 0 */	5	u(4)
layer_level_idc[i]	5	u(8)
} else		
profile_level_info_src_layer_id_delta [i]		ue(v)
if(bitrate_info_present_flag[i]) {		
avg_bitrate[i]	5	u(16)
max_bitrate_layer[i]	5	u(16)
max_bitrate_decoded_picture[i]	5	u(16)
max_bitrate_calc_window[i]	5	u(16)
}		
if(frm_rate_info_present_flag[i]) {		
constant_frm_rate_idc[i]	5	u(2)
avg_frm_rate[i]	5	u(16)
} else		
frm_rate_info_src_layer_id_delta[i]	5	ue(v)

if(frm_size_info_present_flag[i] || iroi_slice_division_info_present_flag[i]) {		
frm_width_in_mbs_minus1[i]	5	ue(v)
frm_height_in_mbs_minus1[i]	5	ue(v)
} else		
frm_size_info_src_layer_id_delta[i]	5	ue(v)
if(sub_region_layer_flag[i]) {		
base_region_layer_id[i]	5	u(8)
dynamic_rect_flag[i]	5	u(1)
if(dynamic_rect_flag[i]) {		
horizontal_offset[i]	5	u(16)
verticial_offset[i]	5	u(16)
region_width[i]	5	u(16)
region_height[i]	5	u(16)
}		
} else		
sub_region_info_src_layer_id_delta[i]	5	ue(v)
if(sub_pic_layer_flag[i])		
roi_id [i]	5	u(3)
if (iroi_slice_division_info_present_flag[i]) {		
iroi_grid_slice_flag[i]	5	u(2)
if (iroi_grid_slice_flag[i] == 0) {		
grid_slice_width_in_mbs_minus1[i]	5	ue(v)
grid_slice_height_in_mbs_minus1[i]	5	ue(v)
} else {		
num_slice_minus1[i]	5	ue(v)
for (j = 0; j <= num_slice_minus1[i]; j++) {		
first_mb_in_slice[i][j]	5	ue(v)
slice_width_in_mbs_minus1[i][j]	5	ue(v)
slice_height_in_mbs_minus1[i][j]	5	ue(v)
}		
}		
}		
if(layer_dependency_info_present_flag[i]) {		
num_directly_dependent_layers[i]	5	ue(v)
for(j = 0; j < num_directly_dependent_layers[i]; j++)		
directly_dependent_layer_id_delta _minus1[i][j]	5	ue(v)
} else		
layer_dependency_info_src_layer_id_delta[i]	5	ue(v)
if(init_parameter_sets_info_present_flag[i]) {		
num_ init_seq_parameter_set_minus1[i]	5	ue(v)
for(j = 0; j <= num_seq_parameter_set_minus1[i]; j++)		
init_seq_parameter_set_id_delta[i][j]	5	ue(v)
num_ init_pic_parameter_set_minus1[i]	5	ue(v)
for(j = 0; j <= num_pic_parameter_set_minus1[i]; j++)		
init_pic_parameter_set_id_delta[i][j]	5	ue(v)

} else		
init_parameter_sets_info_src_layer_id_delta[i]	5	ue(v)
}		
}		

G.10.1.2 Scalability information layers not present SEI message syntax

scalability_info_layers_notpresent(payloadSize) {	**C**	**Descriptor**
num_layers	5	ue(v)
for(i = 0; i < num_layers; i++) {		
layer_id[i]	5	u(8)
}		
}		

G.10.1.3 Scalability information dependency change SEI message syntax

scalability_info_dependency_change (payloadSize) {	**C**	**Descriptor**
num_layers_minus1	5	ue(v)
for(i = 0; i <= num_layers_minus1; i++) {		
layer_id[i]	5	u(8)
layer_dependency_info_present_flag[i]	5	u(1)
if (layer_depdendency_info_present_flag[i]) {		
num_directly_dependent_layers[i]	5	ue(v)
for (j = 0; j < num_directly_dependent_layers[i]; j++)		
directly_dependent_layer_id_delta_minus1[i][j]	5	ue(v)
} else {		
layer_dependency_info_src_layer_id_delta_minus1[i]	5	ue(v)
}		
}		
}		

G.10.1.4 Sub-picture scalable layer SEI message syntax

sub_pic_scalable_layer(payloadSize) {	**C**	**Descriptor**
layer_id	5	ue(v)
}		

G.10.1.5 Non-required picture SEI message syntax

non_required_picture(payloadSize) {	C	Descriptor
num_info_entries_minus1	5	ue(v)
for(i = 0; i <= num_info_entries_minus1; i++) {		
entry_dependency_id[i **]**	5	u(3)
num_non_required_pics_minus1[i **]**	5	ue(v)
for(j = 0; j <= num_non_required_pics_minus1**[** i **]**; j++) {		
non_required_pic_dependency_id[i][j]	5	u(3)
non_required_pic_quality_level[i][j]	5	u(2)
non_required_pic_fragment_order[i][j]	5	u(2)
}		
}		
}		

G.10.1.6 Quality layer information SEI message syntax

quality_layers_info(payloadSize) {	C	Descriptor
dependency_id	All	u(3)
num_quality_layers	5	u(4)
for(i = 0; i < num_quality_layers; i++) {		
quality_layer[i]	5	u(6)
delta_quality_layer_byte_offset[i]	5	ue(v)
}		
}		

G.10.1.7 Scalable nesting SEI message syntax

scalable_nesting(payloadSize) {	C	Descriptor
all_pictures_in_au_flag	5	u(1)
if(all_pictures_in_au_flag = = 0) {		
num_pictures_minus1	5	ue(v)
for(i = 0; i <= num_pictures_minus1; i++) {		
dependency_id[i]	5	u(3)
quality_level[i]	5	u(2)
}		
}		
while(!byte_aligned( ))		
sei_nesting_zero_bit /* equal to 0 */	5	f(1)
sei_message( )	5	
}		

G.10.2 SEI payload semantics

G.10.2.1 Scalability information SEI message semantics

When present, this SEI message shall appear in an IDR access unit. The semantics of the message, with the exception of the layer dependency information, are valid until the next SEI message of the same type. Layer dependency information for a particular layer consists of the information resulting from the decoding of the syntax elements num_directly_dependent_layer, directly_dependent_layer_id_delta_minus1, and layer_dependency_info_ src_layer_id_delay.

NOTE – The layer dependency information may change with a subsequent scalability_info_dependency_change SEI message. However, if, according to the layer dependency information signalled in the most recent scalability information SEI message, layer A does not directly or indirectly depend on layer B, then any succeeding scalability information dependency change SEI message does not change the layer dependency relationship between layers A and B. Specific layers can be indicated as no longer present in the bistream with the scalability_info_layers_notpresent SEI message.

num_layers_minus1 plus 1 indicates the number of scalable layers or presentation points supported by the bitstream. The value of num_layers_minus1 is in the scope of 0 to 255, inclusive.

layer_id[i **]** indicates the identifier of the scalable layer.

Each scalable layer is associated with a layer identifier. The layer identifier is assigned as follows. A larger value of layer identifier indicates a higher layer. A value 0 indicates the lowest layer. Decoding and presentation of a layer is independent of any higher layer but may be dependent on a lower layer. Therefore, the lowest layer can be decoded and presented independently, decoding and presentation of layer 1 may be dependent on layer 0, decoding and presentation of layer 2 may be dependent on layers 0 and 1, and so on. The representation of a scalable layer requires the presence of the scalable layer itself and all the lower layers on which the scalable layer are directly or indirectly dependent. In the following, a scalable layer and all the lower layers on which the scalable layer are directly or indirectly dependent are collectively refered to as the scalable layer representation.

priority_id[i], **temporal_level[** i **]**, **dependency_id[** i **]** and **quality_level[** i **]** are equal to priority_id, temporal_level, dependency_id and quality_level, respectively, of the NAL units in the scalable layer with layer identifier equal to layer_id[i].

discardable_flag[i] is equal to 1 if all NAL units in the scalable layer with layer identifier equal to layer_id[i] have discardable_flag equal to 1. discardable_flag[i] is equal to 0 if any NAL unit in the scalable layer with layer identifier equal to layer_id[i] has discardable_flag equal to 0.

sub_pic_layer_flag[i **]** equal to 1 indicates that the scalable layer with layer identifier equal to layer_id[i] consists of sub-pictures, each sub-picture consists of a subset of coded slices of an access unit. A value 0 indicates that the scalable layer consists of entire access units.

NOTE – The mapping of each sub-picture of a coded picture to a scalable layer is signalled by the sub-picture scalable layer information SEI message.

sub_region_layer_flag[i **]** equal to 1 indicates that the sub-region information for the scalable layer with layer identifier equal to layer_id[i] is present in the SEI message. A value 0 indicates that sub-region information for the scalable layer is the same as another layer indicated by sub_region_info_src_layer_id_delta[i] (when sub_region_info_src_layer_id_delta[i] is greater than 0) or that the sub-region information for the scalable layer is not present in the SEI message (when sub_region_info_src_layer_id_delta[i] is equal to 0).

iroi_slice_division_info_present_flag[i **]** equal to 1 indicates that interactive region-of-interest (IROI) slice division information and frame size information for the scalable layer with layer identifier equal to layer_id[i] is present in the SEI message. A value 0 indicates that the IROI slice division information for the scalable layer with layer identifier equal to layer_id[i] is not present in the SEI message

profile_level_info_present_flag[i **]** equal to 1 indicates the presence of the profile and level information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the profile and level information for the scalable layer with layer identifier equal to layer_id[i] is the same as another layer indicated by profile_level_info_src_layer_id_delta [i]. (when profile_level_info_src_layer_id_delta[i] is greater than 0) or that the profile and level information for the scalable layer is not present in the SEI message (when profile_level_info_src_layer_id_delta[i] is equal to 0).

decoding_dependency_info_present_flag[i **]** equal to 1 indicates the presence of the decoding dependency information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the decoding dependency information for the scalable layer with layer identifier equal to layer_id[i] is the same as another layer indicated by decoding_dependency_info_src_layer_id_delta_minus1[i].

bitrate_info_present_flag[i **]** equal to 1 indicates the presence of the bitrate information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the bitrate information for the scalable layer with layer identifier equal to layer_id[i] is not present in the SEI message.

frm_rate_info_present_flag[i **]** equal to 1 indicates the presence of the frame rate information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the frame rate information for the scalable layer is the same as another layer indicated by frm_rate_info_src_layer_id_delta[i]. (when frm_rate_info_src_layer_id_delta[i] is greater than 0) or that the frame rate information for the scalable layer is not present in the SEI message (when frm_rate_info_src_layer_id_delta[i] is equal to 0).

frm_size_info_present_flag[i **]** equal to 1 indicates the presence of the frame size information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the frame size information for the scalable layer is the same as another layer indicated by frm_size_info_src_layer_id_delta_minus1[i].

layer_dependency_info_present_flag[i **]** equal to 1 indicates the presence of the layer dependency information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the layer dependency information for the scalable layer is the same as another layer indicated by layer_dependency_info_src_layer_id_delta[i]. (When layer_dependency_info_src_layer_id_delta[i] is greater than 0) or that the layer dependency information for the scalable layer is not present in the SEI message (When layer_dependency_info_src_layer_id_delta[i] is equal to 0).

init_parameter_sets_info_present_flag[i **]** equal to 1 indicates the presence of the initial parameter sets information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the initial parameter sets information for the scalable layer is the same as another layer indicated by init_parameter_set _info_src_layer_id_delta[i] (when init_parameter_set_info_src_layer_id_delta[i] is greater than 0) or that the initial parameter sets information for the scalable layer is not present in the SEI message (when init_parameter_set_info_src_layer_id_delta[i] is equal to 0)..

NOTE – The initial parameter sets refers to those parameter sets that can be put in the beginning of the bitstream or that can be transmitted in the beginning of the session.

exact_interlayer_pred_flag[i] equal to 1 indicates that there is no mismatch in the inter-layer prediction reference signal when decoding the layer with layer identifier equal to layer_id[i] compared to the inter-layer prediction reference signal when the layer was encoded. A value 0 indicates that the mismatch exists

NOTE – The mismatch of the inter-layer prediction reference signal for decoding and decoding may be present because of discarding or truncation of progressive refinement NAL units.

layer_profile_idc[i], **layer_constraint_set0_flag**[i], **layer_constraint_set1_flag**[i], **layer_constraint_set2_flag**[i], **layer_constraint_set3_flag**[i], **and layer_level_idc**[i] indicate the profile and level compliancy of the bitstream of the representation of scalable layer with layer identifier equal to layer_id[i]. The semantics of layer_profile_idc[i], layer_constraint_set0_flag[i], layer_constraint_set1_flag[i], layer_constraint_set2_flag[i], layer_constraint_set3_flag[i], and layer_level_idc[i] are identical to the semantics of profile_idc, constraint_set0_flag, constraint_set1_flag, constraint_set2_flag, constraint_set3_flag and level_idc, respectively, with the bitstream in question being the one of this scalable layer representation.

profile_level_info_src_layer_id_delta[i] greater than 0 specifies that the layer with layer identifier equal to layer_id[i] has the same profile and level information as the layer with layer identifier equal to (layer_id[i] – profile_level_info_src_layer_id_delta[i]). A value 0 indicates that the profile and level information for the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

avg_bitrate[i **]** indicates the average bit rate, in units of 1000 bits per second, of the bitstream of the representation of scalable layer with layer identifier equal to layer_id[i]. The semantics of avg_bitrate[i] is identical to the semantics of average_bit_rate in the sub-sequence layer characteristics SEI message when accurate_statistics_flag is equal to 1, except that herein the target bitstream being the bitstream of the scalable layer representation.

max_bitrate_layer[i **]** indicates the maximum bit rate, in units of 1000 bits per second, of the bitstream of the scalable layer with layer identifier equal to layer_id[i], calculated based on a time window specified by max_bitrate_calc_window[i].

max_bitrate_decoded_picture[i **]** indicates the maximum bit rate, in units of 1000 bits per second, of the bitstream of the representation of the scalable layer with layer identifier equal to layer_id[i], calculated based on a time window specified by max_bitrate_calc_window[i].

max_bitrate_calc_window[i **]** indicates the length of the time window, in units of 1/100 second, based on which max_bitrate1[i] and max_bitrate2[i] are calculated.

constant_frm_rate_idc[i **]** indicates whether the frame rate of the representation of the scalable layer with layer identifier equal to layer_id[i] is constant. If the value of avg_frm_rate as specified in below is constant whichever a temporal section of the scalable layer representation is used for the calculation, then the frame rate is constant, otherwise the frame rate is non-constant. Value 0 denotes a non-constant frame rate, value 1 denotes a constant frame rate, and value 2 denotes that it is not clear whether the frame rate is constant. The value of constantFrameRate is in the range of 0 to 2, inclusive.

avg_frm_rate[i **]** indicates the average frame rate, in units of frames per second, of the bitstream of the representation of scalable layer with layer identifier equal to layer_id[i]. The semantics of avg_frm_rate[i] is identical to the semantics of average_frame_rate in sub-sequence layer characteristics SEI message when accurate_statistics_flag is equal to 1, except that herein the target bitstream being the bitstream of the scalable layer representation.

frm_rate_info_src_layer_id_delta greater than 0 specifies that the layer with layer identifier equal to layer_id[i] has the same frame rate information as the layer with layer identifier equal to (layer_id[i] – frm_rate_info_src_layer_id_delta [i]). A value 0 indicates that the frame rate information of the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

frm_width_in_mbs_minus1[i **]** plus 1 indicates the maximum width, in macroblocks, of a coded frame in the representation of the scalable layer with layer identifier equal layer_id[i].

The variable PicWidthInMbs[i] is derived as (frm_width_in_mbs_minus1[i]+1).

frm_height_in_mbs_minus1[i **]** plus 1 indicates the maximum height, in macroblocks, of a coded frame in the representation of the scalable layer with layer identifier equal layer_id[i].

The variable PicHeightInMbs[i] is derived as (frm_height_in_mbs_minus1[i]+1).

The variable PicSizeInMbs[i] specifies the picture size in unit of macroblocks and is derived as PicWidthInMbs[i]* PicHeightInMbs[i].

frm_size_info_src_layer_id_delta[i] specifies that the layer with layer identifier equal to layer_id[i] has the same frame size information as the layer with layer identifier equal to (layer_id[i] – frm_size_info_src_layer_id_delta [i]). A value 0 indicates that the frame size information of the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

base_region_layer_id[i **]** plus 1 indicates the layer identifier value of the scalable layer wherein the represented region is used as the base region for derivation of the region represented by the scalable layer with layer identifier equal to layer_id[i].

dynamic_rect_flag[i **]** equal to 1 indicates that the region represented by the scalable layer with layer identifier equal to layer_id[i] is a dynamically changed rectangular part of the base region. Otherwise the region represented by the current scalable layer is a fixed rectangular part of the base region.

horizontal_offset[i **]** and **verticial_offset[** i **]** give the horizontal and vertical offsets, respectively, of the top-left pixel of the rectangular region represented by the representation of the scalable layer with layer identifier equal to layer_id[i], in relative to the top-left pixel of the base region, in luma samples of the base region.

region_width[i **]** and **region_height[** i **]** give the width and height, respectively, of the rectangular region represented by the representation of the scalable layer with layer identifier equal to layer_id[i], in luma samples of the base region.

sub_region_info_src_layer_id_delta[i] specifies that the layer with layer identifier equal to layer_id[i] has the same sub-region information as the layer with layer identifier equal to (layer_id[i] – sub_region_info_src_layer_id_delta [i]). A value 0 indicates that the sub-region information of the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

roi_id [i] indicates the region-of-interest identifier of the region represented by the scalable layer with layer identifier equal to layer_id[i].

iroi_grid_slice_flag[i] specifies how the IROI slice division information is coded for the scalable layer with layer identifier equal to layer_id[i].

iroi_grid_slice_flag[i] equal to 1 specifies slice division into rectangular grids of identical size.

iroi_grid_slice_flag[i] equal to 0 specifies slice division into rectangular grids of different sizes.

grid_slice_width_in_mbs_minus1[i] plus 1 and **grid_slice_height_in_mbs_minus1**[i] plus 1 specify the width and height, respectively, in unit of macroblocks, of the grid slices in the scalable layer with layer identifier equal to layer_id[i]. All the grid slices have identical width and height, with the following exceptions:

– When (PicWidthInMbs[i] % (grid_slice_width_in_mbs_minus1[i] + 1)) is not equal to 0, the very right column of grid slices of the scalable layer have a width equal to:

(PicWidthInMbs[i] % (grid_slice_width_in_mbs_minus1[i] + 1)) macroblocks.

– When (PicHeightInMbs[i] % (grid_slice_height_in_mbs_minus1[i] + 1)) is not equal to 0, the very bottom row of grid slices of the scalable layer have a height equal to:

(PicHeightInMbs[i] % (grid_slice_height_in_mbs_minus1[i] + 1)) macroblocks.

num_slice_minus1[i] plus 1 specifies the number of slices in a coded picture of the scalable layer with layer identifier equal to layer_id[i].

first_mb_in_slice[i][j] specifies the macroblock address of the first macroblock in raster scan order of the j-th slice with slice identification number equal to j in the scalable layer with layer identifier equal to layer_id[i].

slice_width_in_mbs_minus1[i][j] plus 1 and **slice_height_in_mbs_minus1**[i][j] plus 1 specify the width and height, respectively, in unit of macroblocks, of the j-th slice with slice identification number equal to j in the scalable layer with layer identifier equal to layer_id[i].

num_directly_dependent_layers[i **]** indicates the number of scalable layers that the scalable layer with layer identifier equal to layer_id[i] is directly dependent on. Layer A is directly dependent on layer B if there is at least one coded picture in layer A using inter-layer prediction from layer B. The value of num_directly_dependent_layers is in the scope of 0 to 255, inclusive.

directly_dependent_layer_id_delta_minus1[i **][** j **]** indicates the difference between the layer identifier of the jth scalable layer that the scalable layer with layer identifier equal to layer_id[i] is directly dependent on and i. The layer identifier of the directly dependent scalable layer is equal to (directly_dependent_layer_id_delta + i).

layer_dependency_info_src_layer_id_delta[i] specifies that the layer with layer identifier equal to layer_id[i] has the same layer dependency information as the layer with layer identifier equal to (layer_id[i] – layer_dependency_info_src_layer_id_delta[i]). A value 0 indicates that the layer dependency information of the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

num_init_seq_parameter_set_minus1[i **]** plus 1 indicates the number of initial sequence parameter sets for decoding the representation of the scalable layer with layer identifier equal to layer_id[i].

init_seq_parameter_set_id_delta[i **][** j **]** indicates the value of the seq_parameter_set_id of the jth initial sequence parameter set for decoding the representation of the scalable layer with layer identifier equal to layer_id[i] if j is equal to 0. If j is larger than 0, init_seq_parameter_set_id_delta[i][j] indicates the difference between the value of the seq_parameter_set_id of the jth initial sequence parameter set and the value of the seq_parameter_set_id of the $(j-1)^{th}$ initial sequence parameter set. The initial sequence parameter sets are logically ordered in ascending order of the value of seq_parameter_set_id.

num_init_pic_parameter_set_minus1[i **]** plus 1 indicates the number of initial picture parameter sets for decoding the representation of the scalable layer with layer identifier equal to layer_id[i].

init_pic_parameter_set_id_delta[i **][** j **]** indicates the value of the pic_parameter_set_id of the jth initial picture parameter set for decoding the representation of the scalable layer with layer identifier equal to layer_id[i] if j is equal to 0. If j is larger than 0, init_pic_parameter_set_id_delta[i][j] indicates the difference between the value of the pic_parameter_set_id of the jth initial picture parameter set and the value of the pic_parameter_set_id of the $(j-1)^{th}$ initial picture parameter set. The initial picture parameter sets are logically ordered in ascending order of the value of pic_parameter_set_id.

init_parameter_sets_info_src_layer_id_delta [i] specifies that the layer with layer identifier equal to layer_id[i] has the same initial parameter sets information as the layer with layer identifier equal to (layer_id[i] – init_parameter_sets_info_src_layer_id_delta[i]). A value 0 indicates that the initial parameter sets information of the layer with layer identifier equal to layer_id[i] is not present in the SEI message.

G.10.2.2 Scalability information layers not present SEI message semantics

When present, this SEI message will take effect for all NAL units that follow. This SEI message indicates the layers that will not be present in the bitstream, and is always interpreted with respect to the most recent scalability_info SEI

message. The scalability information effective after the application of this message remains valid until the next SEI message of the type scalability_info, scalability_info_layers_notpresent, or scalability_info_dependency_change.

NOTE – scalability_info_layers_notpresent SEI messages do not have a cumulative effect and instead always refer to the most recent scalability_info SEI message.

num_layers indicates the number of scalable layers or presentation points that will not be present in the bitstream after decoding of this SEI message. The value of num_layers is in the scope of 0 to 255, inclusive. When num_layers has the value 0 it indicates that all layers are present.

layer_id[i **]** indicates the identifier of the scalable layer that is indicated as not present in the bitstream. The layer_id values refer to, and must be among, the layer_id values indicated by the most recent scalability_info SEI message.

G.10.2.3 Scalability information dependency change SEI message semantics

When present, this SEI message will take effect for all NAL units that follow. It indicates that the layer dependency information for the specified layers will change as specified by the contents of this message, and is always interpreted with respect to the most recent scalability information SEI message. If, according to the layer dependency information signaled in the most recent scalability information SEI message, layer A does not directly or indirectly depend on layer B, then the scalability information dependency change SEI message shall not change the layer dependency relationship between layers A and B. The scalability information state effective after the application of this message remains valid until the next SEI message of the type scalability_info, scalability_info_layers_notpresent, or scalability_info_dependency_change.

NOTE – scalability_info_dependency_change SEI messages do not have a cumulative effect and instead always refer to the most recent scalability_info SEI message.

num_layers_minus1 plus 1 indicates the number of scalable layers or presentation points for which the layer dependency information will change after decoding of this SEI message. The value of num_layers_minus1 is in the scope of 0 to 255, inclusive.

layer_id[i] indicates the identifier of the scalable layer for which the layer dependency information will change.

layer_dependency_info_present_flag[i] equal to 1 indicates the presence of layer dependency information for the scalable layer with layer identifier equal to layer_id[i] in the SEI message. A value 0 indicates that the layer dependency information for the scalable layer with layer identifier equal to layer_id[i] is not present in the SEI message, and is the same as another layer indicated by layer_dependency_info_src_layer_id_delta_minus1[i].

num_directly_dependent_layers[i] indicates the number of scalable layers that the scalable layer with layer identifier equal to layer_id[i] is directly dependent on. Layer A is directly dependent on layer B if there is at least one coded picture in layer A using inter-layer prediction from layer B. The value of num_directly_dependent_layers is in the scope of 0 to 255, inclusive.

directly_dependent_layer_id_delta_minus1[i][j] indicates the difference between layer_id[i] and the layer identifier of the jth scalable layer that the scalable layer with layer identifier equal to layer_id[i] is directly dependent on. The layer identifier of the directly dependent scalable layer is equal to (layer_id[i] – directly_dependent_layer_id_delta_minus1 – 1).

layer_dependency_info_src_layer_id_delta_minus1[i] specifies that the layer with layer identifier equal to layer_id[i] has the same layer dependency information as the layer with layer identifier equal to (layer_id[i] – layer_dependency_info_src_layer_id_delta_minus1[i] – 1).

G.10.2.4 Sub-picture scalable layer SEI message semantics

When present, this SEI message shall appear in the same SEI payload containing a motion-constrained slice group set SEI message and immediately succeeds the motion-constrained slice group set SEI message in decoding order. The slice group set identified by the motion-constrained slice group set SEI message is called the associated slice group set of the sub-picture layer information SEI message.

layer_id indicates the layer identifier of the scalable layer to which the coded slice NAL units in the associated slice group set belongs.

G.10.2.5 Non-required picture SEI message semantics

The information conveyed in this SEI message concerns an access unit. When present, this SEI message shall appear before any coded slice NAL unit or coded slice data partition NAL unit of the corresponding access unit.

num_info_entries_minus1 plus 1 indicates the number of the information entries following this syntax element. The value shall be in the range of 0 to 7, inclusive.

entry_dependency_id[i **]** indicates the dependency_id value of the target picture whose information of non-required pictures is described by the following syntax elements. The instances of entry_dependency_id[i] shall appear in the increasing order of their values. The quality_level value of the target picture is always zero. A non-required picture of the target picture is not required in decoding of any other pictures in the coded video sequence and having the same dependency_id value and quality_level value as the target picture.

NOTE – A picture having quality_level larger than 0 is a FGS picture whose inter-prediction reference source is always fixed. Therefore, a FGS picture's non-required pictures are the same as the picture having the same dependency_id value as the FGS picture and quality_level equal to 0.

num_non_required_pics_minus1[i **]** plus 1 indicates the number of non-required pictures signalled in the current entry for the target picture having the dependency_id value equal to entry_dependency_id[i] and the quality_level value equal to 0. The value shall be in the range of 0 to 30, inclusive.

non_required_pic_dependency_id[i **][** j **]** indicates the dependency_id value of the j-th non-required picture signalled in the current entry for the target picture having the dependency_id value equal to entry_dependency_id[i] and the quality_level value equal to 0.

non_required_pic_quality_level[i **][** j **]** indicates the quality_level value of the j-th non-required picture signalled in the current entry for the target picture having the dependency_id value equal to entry_dependency_id[i] and the quality_level value equal to 0. In addition, those pictures that have dependency_id equal to non_required_pic_dependency_id[i][j] and quality_level larger than non_required_pic_quality_level[i][j] are also non-required pictures for the same target picture.

non_required_pic_fragment_order[i **][** j **]** indicates the fragment_order value of the j-th non-required picture signalled in the current entry for the target picture having the dependency_id value equal to entry_dependency_id[i] and the quality_level value equal to 0. In addition, those pictures that have dependency_id equal to non_required_pic_dependency_id[i][j], quality_level equal to non_required_pic_quality_level[i][j] and fragment_order larger than non_required_pic_fragment_order[i][j] are also non-required pictures for the same target picture.

Besides the non-required pictures explicitly signalled in the SEI message, the following rules shall be applied to derive additional non-required pictures:

– If a picture having dependency_id equal to A is not a non-required picture for the picture having dependency_id equal to B wherein B is larger than or equal to A, then all the non-required pictures for the picture having dependency_id equal to A are also non-required pictures for the picture having dependency_id equal to B.

– If the layer desired for playback has dependency_id equal to C that is not equal to any of the signalled entry_dependency_id[i] values, the n-th entry that has the largest entry_dependency_id[i] smaller than C is searched for. The picture having dependency_id equal to C shall have the same set of non-required pictures as the picture having dependency_id equal to the entry_dependency_id[i] of the n-th entry and quality_level equal to 0. If there is no entry that has entry_dependency_id[i] smaller than C, then there are no non-required pictures in the associated access unit for the picture having dependency_id equal to C.

G.10.2.6 Quality layer information SEI message semantics

num_quality_layers specifies the number of quality layers defined for this frame.

quality_layer[i **]** specifies the value of the i-th quality layer.

delta_quality_layer_byte_offset[i **]** specifies the number of bytes that should be extracted for the i-th quality layer. For each i, delta_quality_layer_byte_offset[i] specifies the number of additional offset bytes for the current quality layer. The total byte offset quality_layer_byte_offset is calculated as

```
quality_layer_byte_offset = 0
for ( n = 0; n < i; n++ )
    quality_layer_byte_offset += delta_quality_layer_byte_offset[ n ]
```

The total byte offset quality_layer_byte_offset indicates the truncation point for the progressive refinement packet.

G.10.2.7 Scalable nesting SEI message semantics

A scalable nesting SEI message concerns an access unit. When present, this SEI message shall appear before any VCL NAL unit of the corresponding access unit. A scalable nest SEI message shall contain one and only one SEI message of

payloadType less than or equal to 21, which is referred to as the nested SEI message. The scope to which the nested SEI message applies is indicated by the syntax elements all_pictures_in_au_flag, and num_pictures, dependency_id[i] and quality_level[i], when present.

all_pictures_in_au_flag equal to 1 indicates that the nested SEI message applies to all the coded pictures of the access unit. all_pictures_in_au_flag equal to 0 indicates that the applicable scope of the nested SEI message is signalled by the the syntax elements num_pictures, dependency_id[i] and quality_level[i].

num_pictures_minus1 plus 1 indicates the number of coded pictures to which the nested SEI message applies.

dependency_id[i] and **quality_level**[i] indicate, respectively, the dependency_id and the quality_level of the i-th coded picture to which the nested SEI message applies.

sei_nesting_zero_bit is equal to 0.