**UNITED STATES PATENT AND TRADEMARK OFFICE**

---

**BEFORE THE PATENT TRIAL AND APPEAL BOARD**

---

AMAZON.COM, INC. and AMAZON.COM SERVICES LLC,
Petitioner

v.

INTERDIGITAL, INC.,
Patent Owner.

---

IPR2026-00195
U.S. Patent No. 10,250,877

---

# DECLARATION OF DR. CLIFFORD READER, PH.D.

## Declaration

I declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Date: 1/5/26. By: C. Reader,

Clifford Reader, Ph.D.



TCL-1005
TTE Technology Inc. d/b/a TCL North America v. InterDigital, Inc. IPR2026-00417
AMAZON-1005
Amazon v. InterDigital, IPR2026-00195

## Contents

I. BACKGROUND AND QUALIFICATIONS ..... 3
A. Education ..... 4
B. Work Experience ..... 4
C. Standardization ..... 5
D. Intellectual Property Rights ..... 7
E. Curriculum Vitae ..... 8
II. Legal Standards ..... 9
III. Brief Overview of the Standardization Process ..... 10
IV. The HEVC Working Documents Were Publicly Available ..... 14
V. Additional Remarks ..... 26

I, Dr. Clifford Reader, do hereby declare:

1. I am making this declaration at the request of Amazon.com, Inc. and Amazon.com Services LLC in the matter of *Amazon.com, Inc. v. Interdigital, Inc.,* IPR2026-00195.

2. I am being compensated for my work in this matter at my standard hourly rate of $850 for consulting services. My compensation in no way depends on the outcome of this proceeding or the content of my testimony.

3. In preparing this Declaration, I specifically considered the following materials, along with the exhibits that I cite throughout this declaration:

(a) Thomas Wiegand et al., *WD3: Working Draft 3 of High-Efficiency Video Coding*, JCTVC-E603 v.8 (June 27, 2011) ("Wiegand");

(b) Thomas Davies, *Resolution switching for coding efficiency and resilience*, JCTVC-F158 v.4 (July 7, 2011) ("Davies").

## I. BACKGROUND AND QUALIFICATIONS

4. I am a digital media consultant providing technical, business development and intellectual property consulting services in the areas of digital media including digital imaging, digital video, digital audio and digital speech. Applications include consumer audio and video transmission and storage, video, audio and speech compression, real-time video processing and display, digital

speech processing, image/video/audio systems architecture, and image/video/audio chip architecture. I have held this position since 2001 and have consulted for over 80 clients.

**A. Education**

5. I received my Doctoral degree in 1974 from the University of Sussex, England. My thesis was titled "Orthogonal Transform Coding of Still and Moving Pictures". The research for my thesis was performed in residence at the Image Processing Institute, University of Southern California, Los Angeles.

6. I received my B. Eng. Degree with Honors in 1970 from the University of Liverpool, England, in the field of electronics.

**B. Work Experience**

7. From 1970 to 1973 I performed my graduate research in video compression. I was one of the first to perform a type of image coding (adaptive block transform coding) and the first to apply this type of coding to video. This is described in my thesis[1] and summarized in an SPIE paper[2]. These techniques underlie the

[1] Reader C., "Orthogonal Transform Coding of Still and Moving Pictures", U. Sussex, England, 1973.

[2] Reader C., "Intraframe and Interframe Adaptive Transform Coding", SPIE Vol. 66, 1975.

audiovisual coding standards known as MPEG (Moving Picture Experts Group), H.26x, and virtually all other video compression schemes today.

8. From 1975 to 1981 I studied, designed and developed systems for military imaging systems including real-time image and video reconnaissance systems and battlefield management systems.

9. In the early 1980s I taught classes at Santa Clara University, California in digital signal processing and digital image processing.

10. From 1982 to 1989 I architected and led hardware and software engineering teams in the design of systems for real-time imaging for military, medical and earth resources applications.

11. From 1990 to 2001 I architected and led hardware and software engineering teams in the design of semiconductor chips and systems for real-time imaging in digital consumer audio/video applications, including videoconferencing (with speech coding), broadcast TV and DVD.

**C. Standardization**

12. In 1990 I became an accredited delegate to the Moving Picture Experts Group – MPEG. I, and my team contributed to the technical work for all three parts of the standard – Systems, Video and Audio – and I participated in the management of MPEG.

13. I was Head of Delegation (HoD) to MPEG for the United States in 1991-1992, and Editor in Chief of the MPEG1 standard. I personally reviewed and edited all three parts of the standard in detail, and wrote much of the informative annex for the MPEG1 standard.

14. I chaired the implementation subcommittee that analyzed MPEG1 Audio (Levels I & II aka MUSICAM; Level III aka mp3), Dolby AC3 and other proposed audio compression algorithms including legacy algorithms for complexity and cost of implementation.

15. I was a co-founder of the MPEG4 standard and chaired the subcommittee from inception for 2-1/2 years, beginning in 1993, following which I chaired the MPEG4 Requirements subcommittee for a further 2 years. These subcommittees established many of the fundamental principles of the MPEG4 standard, including object-based coding, software-based implementation, and development of the bitstream as a syntactic language.[3] MPEG4 focused on audiovisual low bitrate coding, including low bitrate speech coding.

16. I was instrumental in establishing the work on Advanced Audio Coding (AAC).

[3] Reader C, "MPEG4: Coding for Content, Interactivity, and Universal Accessibility", Optical Engineering, Vol. 35, No. 1, pp. 104-08, Jan. 1996.

17. I initiated the work in Synthetic-Natural Hybrid Coding (SNHC), and personally contributed to the work on compression of 3D graphics in the area of error resilient coding.

18. I was an invited expert to the H.264/MPEG4 Pt. 10 JVT standard development for which I contributed to and participated in multiple meetings in person.

19. I have chaired the AVS China IPR Subgroup for over 20 years and attend multiple meetings each year.

20. I have personally tracked the development of all the international standards since 1990 and have accessed all the documents related to the respective ISO and ITU committees.

**D. Intellectual Property Rights**

21. In 1993 I was hired by CableLabs to be the technical expert for establishing the MPEG Patent Pool (Now MPEGLA). In the course of creating a list of essential IP to practice the standard, I reviewed approximately 10,000 abstracts and 1,000 patents. This is summarized in a chapter of the MPEG book by Mitchell et al.[4]

[4] "MPEG Video Compression Standard", Edited by J. L. Mitchell et al., Chapter 16, "MPEG Patents", pp.357-62.

22. In 2002 I was hired by 10 companies to evaluate the standards essential patent environment for the nascent H.264 standard. I was an invited expert to the Joint Video Team (JVT) established by ISO and ITU to develop the H.264 standard (Also denoted MPEG4 Pt. 10, AVC).

23. In 2003 I was hired to evaluate the standards essential patent environment for the nascent national China AVS standards.

24. In 2013 I was hired to evaluate the standards essential patent environment for the emerging AV1 standard

25. In 2017 I was hired to evaluate the standards essential patent environment for the H.265/HEVC standard.

26. I am the Co-Director of the China AVS Patent Pool Administration. I lead the negotiations for sub-licensing of the AVS standards.

27. I have performed expert consulting and expert witness work for patent holders and defendants in patent licensing negotiations and litigation.

**E. Curriculum Vitae**

28. Additional information concerning my professional publications and presentations in the field of digital video and cases in which I have served as an expert are set forth in my current Curriculum Vitae, a copy of which may be found at Ex-1006. This Curriculum Vitae lists many publications authored or co-authored by me; and lists the cases in which I have testified via depositions and trials.

## II. Legal Standards

29. I understand that 35 U.S.C. § 102 sets forth the requirements for a reference to be considered "prior art," which can include among other things: patents and patent applications that were filed and/or published before the patent-at-issue's priority date, *see, e.g.*, 35 U.S.C. § 102(b), (e), or "printed publications" that were published before the patent-at-issue's priority date, *see, e.g.*, 35 U.S.C. § 102(a), (b).

30. I understand that for a reference to be considered a "printed publication" under § 102, it must have been disseminated and publicly accessible to at least the pertinent part of the public who would be interested in the subject matter to which the reference pertains. In other words, a POSITA exercising reasonable diligence must be able to locate it.

31. For instance, I understand that if a reference is indexed or cataloged in some way (e.g., by a library or industry group), or if research tools were provided, such that a POSITA exercising reasonable diligence could locate it, that may be sufficient to demonstrate the reference constitutes a "printed publication," but simply e-mailing a document to a select group of individuals may not. But I also understand that whether a reference constitutes a printed publication depends on the facts of each particular case. I also understand that whether a reference qualifies as a printed publication does not require showing a member of the public actually located and read the reference.

## III. Brief Overview of the Standardization Process

32. Before addressing the prior art, it is worth providing an overview of the standardization process followed by standardization organizations such as the ITU-VCEG and ISO-MPEG – two of the dominant organizations in developing video-coding standards. In my opinion, a POSITA would have understood the general procedures of the standardization process, and a POSITA interested in video coding would have been following video-coding standards developments closely, as many in the industry did to ensure any product involving video coding would be compliant with the relevant standards as soon as the standard was published.

33. For context, the ITU is a "United Nations specialized agency for information and communication technologies (ICTs)." Ex-1046. The ITU is headquartered in Geneva. *Id*. The ITU's Telecommunication Standardization Sector ("ITU-T") facilitates the development of international standards, called ITU Recommendations, that help define a global infrastructure of information and communication technologies. *See* Ex-1051. Within ITU-T is a working group called the Video Coding Experts Group ("VCEG"), which focuses on standards for compression coding of video signals.

34. MPEG is a subcommittee under a joint venture between the International Organization for Standardization ("ISO") and International Electrotechnical Commission ("IEC"), and, like the ITU VCEG, develops

"international standards for compression, decompression, processing, and coded representation of moving pictures." Ex-1052.

35. Although I am not a member of the ITU, I have been accredited to ITU as an invited expert, I am aware of the ITU's website – including as of the earliest possible priority date of the '877 patent – the accuracy of the dates on the ITU's website and the process for documents being uploaded to the ITU's website. I am also intimately familiar with the MPEG's website.

36. The ITU-VCEG's and ISO-MPEG's development of video coding standards generally comprises a two-stage process. In the first so-called competitive phase, the standards body conducts a requirements analysis to determine market need for a new standard and develops specific requirements for functionality and performance to meet those needs. The committee evaluates the evolving state of the art in relevant technologies to assess the degree to which a new standard would out-perform a current standard. The standards body develops a call for proposals that typically includes a provision of test materials that will allow the proposals to be evaluated on equal terms, test procedures that lay out what each proponent must provide and how evaluation will be conducted, as well as written submission requirements to describe the proposed coding tools and complexity/cost of implementing such tools.

37. The proposals are received, evaluated, and ideally one is selected to be the foundation of the new standard. In practice, the foundation may comprise an amalgam of several proposals.

38. The second so-called collaborative phase then begins, with the establishment of a "working draft" of the standard, plus a "test model" and "reference software," that will support a recursive development process over a succession of meetings. Although the final video coding standard will only normatively specify the coded data format and decoding process, the test model and reference software comprise an end-to-end encoding and decoding process, so that independent parties can conduct experiments that are directly comparable.

39. At each meeting, proponents can submit contributions that may comprise proposals for enhancing existing video coding tools or adding new tools. The proposals will generally include testing of the proposal by implementing it on the reference software and testing it with the test sequences. Other parties cross-check or validate each proposal by reading the description provided by the proponent and independently implementing the proposal on the reference software and testing it. This is facilitated by the uploading of such proposals before a meeting and of course open access to such documents. At the corresponding meeting, the standards body can review the proposal and test results from both the proponent and cross-checkers, and can decide whether or not to adopt the proposal into the draft standard.

If adopted, the proposed tool will be added to the working draft, test model and reference software. In this manner, the standard recursively advances toward maturity.

40. At meetings, the standards body may specify "core experiments" that describe in detail a particular problem or area of work where advancement is needed. Between meetings, the standards body may establish "ad hoc groups" – open to everyone – that may collaborate on specific developments, e.g., core experiments and exchange ideas via email.

41. At each meeting, the chairpersons develop a meeting report, published at the end of the meeting. The meeting report provides a summary and disposition of each contribution to the meeting, summaries of ad hoc group reports, results of core experiments, and administration activities, such as reminders of obligations under the IPR policy. In some cases an "AgendaWithNotes" running account of activities is created progressively during the meeting and is posted online.

42. The openness and timeliness of this process means that any interested person can track the development of the standard – even without attending the meetings. For example, semiconductor companies can have their research and product development departments port the reference software to their proprietary firmware environment after it is updated each meeting and simulate the nascent standard on their own chip development platform. In that way, the companies can

be prepared to release new chips supporting the new standard as soon after the final specification is frozen as possible.

## IV. The HEVC Working Documents Were Publicly Available

43. I have been asked to provide my knowledge of the HEVC standardization process and public availability and accessibility of documents published by the Joint Collaborative Team on Video Coding (JCT-VC) responsible for developing the HEVC standard. Although I was not a member of the JCT-VC, as an interested participant in the field, I was aware of the HEVC standardization process, and was following its developments.

44. I was actively monitoring the development of not only the HEVC standard, including each document that was published to the JCT-VC website, but also the development of other standards through meetings of ISO/MPEG, ITU/VCEG and ISO/ITU JVT committees in the years leading up to the formation of JCT-VC in 2010.

45. I have personal knowledge regarding the procedures for publishing JCT-VC documents on the Internet, including in the 2010-2011 timeframe. I can therefore confirm that Wiegand (Ex-1009), Davies (Ex-1010), and other documents published as part of the JCT-VC's standardization process (such as Misra (Ex-1013) and Demircin (Ex-1063)) would have been publicly available and accessible with reasonable diligence as of the date provided on the JCT-VC website.

46. For background, the JCT-VC was created in 2010 to develop the next-generation video codec standard intended to supplant MPEG4/H.264, which was one of the standards I personally contributed to, as I mentioned above.

47. The JCT-VC was a joint venture between the International Telecommunication Union Telecommunications Standardization Sector (ITU-T), and the International Organization for Standardization (ISO) Moving Picture Experts Group (MPEG). *See, e.g.*, Wiegand, 12 (Forward), 13 (§0.1).

48. Based on my personal experience and recollection, those in the video-coding industry were closely monitoring the developments of the JCT-VC. In the years leading up to developing the H.265/HEVC standard within JCT-VC, a process to develop "KTAs" – key technology areas – was conducted with the standards bodies to both identify promising areas for improving the performance of existing video coding tools and develop new video coding tools. Broadly speaking, the decision to launch the development of a new standard was made when it was felt that the performance of the previous generation standard could be improved by approximately 50%. Then following the launch of the standardization process itself, interested parties such as myself tracked the progress of the development of the draft standard as it advanced from meeting to meeting. This open process was greatly facilitated by public availability of the associated documents, including the draft standard and reference software. For example, a company may not have been

actively attending meetings and contributing new proposals, but was developing products in-house that would implement the new standard or provide services employing the new standard, and by using the reference software from the standards body was ready to enter the market as soon as the standard was finalized.

49. I also am aware of an article published by the Institute of Electrical and Electronics Engineers ("IEEE") – an authoritative source of technical information in many fields, including video-coding technologies – that announced the formation of JCT-VC in December 2010, and how meetings were already commencing. Ex-1037. Thus, readers of IEEE literature – of which there are many – would have known that JCT-VC's standardization work was already under way as of 2010, and would have known to search out further information on the development of HEVC.

50. Based upon my personal experience with the MPEG4/H.264 standardization process, and my knowledge of the HEVC standardization process, I recognized that members of the JCT-VC met quarterly – beginning in April 2010 – to advance the HEVC standardization efforts. *See* Ex-1036. Each meeting was assigned a "Unique Serial Letter" to track (among other things) which documents were attributable to each meeting. *See id.* In other words, Meeting No. 1 was assigned the letter "A," Meeting No. 2 was assigned the letter "B" and so on.

51. The "terms" which guided the JCT-VC were posted publicly on the ITU's website in 2010, and can be found here: https://www.itu.int/oth/T4601000001/en. (Ex-1031).

52. According to those terms, before each meeting, JCT-VC members would upload proposals for incorporation into the standard to be considered at that meeting. *See, e.g.*, Ex-1031, 2 ("Every contribution document to a meeting of the JCT should be registered in the document registry and uploaded to the electronic archive several days in advance of the meeting, to ensure that it is available for review by other participants."). These were also referred to as "input" documents. *See, e.g.*, *id.*, 2. After each meeting, a Meeting Report would be issued, and a working draft of the standard would typically be published. *Id.*, 2, 4. These were also referred to as "output" documents. *See id.*, 2.

53. According to the JCT-VC terms, "the JCT will maintain a single document registry and an electronic archive that are distinct from those of the parent bodies," but may be "linked to both the parent body web sites" (such as the ITU-T or MPEG websites), "and the parent bodies may ingest the JCT documents for their own reference and archival purposes." *Id.*

54. The JCT-VC terms also declared that "all input and output documents of the JCT will be public." *Id.*

55. The first report issued after the JCT-VC's first April 2010 meeting stated that all documents were being stored at an "FTP" (or "file transfer protocol" website: http://ftp3.itu.int/av-arch/jctvc-site/. Ex-1038, 3. This same website was used for all documents related to the second meeting. Ex-1039, 2. However, after the third meeting, the JCT-VC announced that it was migrating to "[a] new document distribution site http://phenix.it-sudparis.eu/jct/[, which] became operational shortly before the [third] meeting and was exclusively used for distribution of all documents." Ex-1040, 2.[5] The JCT-VC explained that "[t]he new site allows a more automated process of document registration and download." *Id.* Each meeting report during the relevant timeframe[6] contained a similar statement identifying where the documents were being hosted. *See* Ex-1041, 2; Ex-1042, 2; Ex.1043, 3; Ex-1044, 3.

56. Based on my personal experience and recollection, I was aware of the JCT-VC document system from inception. I personally accessed documents at the JCT-VC website beginning in 2010 and routinely downloaded proposals, working drafts, and meeting reports from that website. I accessed these documents because

[5] I also understand that the exact same website, including all JCT-VC documents and functionality is also available at a different internet address: phenix.int-evry.fr/jct/. I have seen both websites used and cited by others.

[6] The relevant timeframe is from approximately 2010 when the JCT-VC's public efforts began, until what I understand to be the earliest possible priority date of the '877 Patent: January 13, 2012. *See* Ex-1001, Cover.

of my involvement in the MPEG4/H.264 standardization process, and the development of the HEVC standard was relevant to my area of study and consultation. Documents on the JCT-VC website were available without any restrictions, such as requiring an account or password.

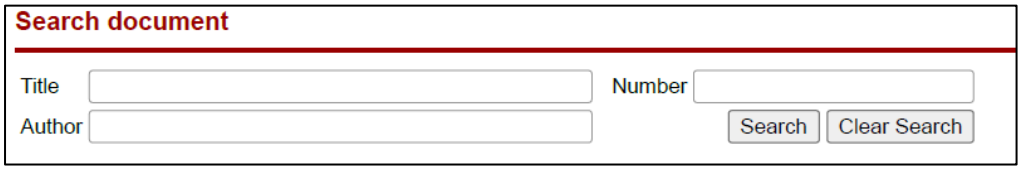
57. Leading up to the formation of JCT-VC and launch of the H.265/HEVC standardization work, I was accessing the documents of the VCEG standards body. It was there that the contributions and discussions of the KTAs that formed the basis for the new standard could be found.

58. Based on my personal experience and recollection, during the relevant timeframe, the documents on the JCT-VC website were organized by meeting. *See* Ex-1036. Once a meeting was selected from the menu, all of the input and output documents associated with that meeting were viewable, and included the document number, associated MPEG number (for the mirrored version on the MPEG website), the various upload dates (since many documents were revised and had multiple different versions), the title of the document, and the source. *See, e.g.*, Ex-1057 (Meeting 5); Ex-1060 (Meeting 6). All documents were downloadable. The website as it is presented today is consistent with my recollection of how it was presented during the relevant timeframe.

59. Moreover, I have never seen anything that would make me doubt the accuracy of the dates and timestamps of the JCT-VC documents shown on the JCT-

VC website. In fact, the third meeting report noted how "[r]egistration timestamps, initial upload timestamps, and final upload timestamps … reflect activity on the new document site." Ex-1040, 3; *see id.*, 2-3 (explaining an issue with the timeliness of certain uploads when migrating to the new website). It is my understanding that when a document is uploaded to the JCT-VC website, the "upload" date and time is automatically generated for each document.

60. Based on my personal experience and recollection, during the relevant timeframe, the JCT-VC website provided the ability to search the documents for each meeting for example by title and author.




*See, e.g.*, Ex-1057, 1 (showing search functionality by "Title," "Author," and "[Document] Number"). I understand that my recollection of the JCT-VC website's search functionality is consistent with what others who were intimately involved in the JCT-VC process have testified to. Ex-1053, ¶¶ 183, 193 (explaining search functionality has stayed the same since 2011). Those interested in following the HEVC development (such as myself) could therefore utilize the search functionality and locate documents of interest at the time by searching for words that would be

expected in the title of a relevant document, or by searching the author (e.g., if they had been publishing papers on a particular topic of interest).

61. Utilizing the search functionality would have significantly narrowed down the results displayed on the website. For instance, in my opinion, one of the best resources to understand the current state of the HEVC standard at any given time was to review the Working Draft of the standard. This could be located, for instance, by searching terms such as "working draft" or "wd" in the "Title" search bar. If one were particularly interested in evaluating proposals for the HEVC's deblocking filter, for example, one could search "deblock" to narrow the results. When searching Meeting No. 4, for instance, running a search for "deblock" reduces over 600 entries associated with this meeting to just 17 entries. *See* Ex-1055. It was thus fairly straightforward to decipher how particular proposals developed throughout the standardization process.

62. Furthermore, the meeting reports – in Microsoft Word format – each included the list of documents for that meeting, so downloading the meeting report provided a local copy for searching purposes. Specifically, each meeting report contained a section that indexed the proposals under consideration, and often provided a summary of the proposals' contents. *See, e.g.*, Ex-1038, 5–37; Ex-1039, 5–57; Ex-1040, 8–121; Ex-1041, 11-131; Ex-1042, 12-163; Ex-1043, 12-177; Ex-1044, 13-236. For this reason, even if a particular document was titled generically,

conducting a text search in the meeting report would have identified particular documents of interest that could then be located on the JCT-VC website. The proposals were also cataloged in Annex A of each meeting report, which provided the titles and authors of each document, but not a summary of their contents. *See, e.g.*, Ex-1038, 47-55; Ex-1039, 65-70; Ex-1040, 123-52; Ex-1041, 143-70; Ex-1042, 175-207; Ex-1043, 191-253; Ex-1044, 238-301.

63. Based on my personal experience and recollection, once a particular document of relevance was located, two options were available. First, under the "JCT-VC number" heading, the JCT-VC number for a given document could be selected.

| JCT-VC number | MPEG number | Created | First Upload | Last upload | Title | Source | Download |
|---|---|---|---|---|---|---|---|
| JCTVC-E603 | m20271 | 2011-03-29 19:58:12 | 2011-03-30 12:22:15 | 2011-06-27 14:32:31 | WD3: Working Draft 3 of High-Efficiency Video Coding | T. Wiegand, B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan | JCTVC-E603 |

*See, e.g.*, Ex-1057. This would cause the website to be redirected to a "Document Information" page that identified the different versions of the document that had been uploaded.

| | |
|---|---|
| **Document** | JCTVC-E603 (version 1 - date 2011-03-30 12:22:15)<br>JCTVC-E603 (version 2 - date 2011-04-06 18:49:03)<br>JCTVC-E603 (version 3 - date 2011-05-01 16:46:24)<br>JCTVC-E603 (version 4 - date 2011-05-09 19:05:39)<br>JCTVC-E603 (version 5 - date 2011-05-20 03:15:57)<br>JCTVC-E603 (version 6 - date 2011-06-13 04:23:14)<br>JCTVC-E603 (version 7 - date 2011-06-15 23:20:50)<br>JCTVC-E603 (version 8 - date 2011-06-27 14:32:31)<br>upload document |

Ex-1045. The "Document Information" page identified when each version of the document was uploaded to the JCT-VC website. *See id.* Each version of the document could then be downloaded separately as needed. *See id.* Alternatively, under the "Download" heading, the JCT-VC number for that document could be selected, which would automatically download the most recent version of the document.

| JCT-VC number | MPEG number | Created | First Upload | Last upload | Title | Source | Download |
|---|---|---|---|---|---|---|---|
| JCTVC-E603 | m20271 | 2011-03-29 19:58:12 | 2011-03-30 12:22:15 | 2011-06-27 14:32:31 | WD3: Working Draft 3 of High-Efficiency Video Coding | T. Wiegand, B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan | JCTVC-E603 |

*See, e.g.*, Ex-1057.

64. Based on my personal experience and recollection, and as confirmed by the JCT-VC's website, Wiegand (Ex-1009) and Davies (Ex-1010) (as well as other JCT-VC documents such as Misra (Ex-1013) and Demircin (Ex-1063)) were distributed and publicly available as of their 2011 upload dates displayed on the JCT-VC website. *See, e.g.*, Ex-1057 (showing first and last upload dates for documents for Meeting No. 5); Ex-1060 (same for Meeting No. 6); Ex-1045 (showing Wiegand uploaded on June 27, 2011); Ex-1050 (showing Davies uploaded on July 15, 2011). The copy of Wiegand at Ex-1009 and Davies at Ex-1010 are also true and correct copies of the documents distributed on the JCT-VC website and are substantially identical to the versions that I have in my personal archive, which I have attached as Appendix 1 (Wiegand) and Appendix 2 (Davies). Although I do not remember

exactly when I downloaded Wiegand and Davies, to the best of my recollection, it was many years ago. Once these documents were uploaded, they would have been accessible to anyone interested in finding them. Moreover, if I had been interested at the time, locating them would have been straightforward. For instance, I do specifically recall reviewing each meeting report, which chronicled the evolution of the Working Drafts (like Wiegand) and proposals under consideration throughout the development of the standard.

65. As I mentioned previously, I understand that the earliest possible priority date for the patent-at-issue here is January 2012. Ex-1001, Cover. As of that time, JCT-VC had held only seven meetings, and so I limited my search to the first seven meetings.

66. As I noted previously, although the meeting reports provide an overview of the state of the *development* of the standard and the proposals under consideration, the working drafts were what described the current state of the standard itself. Wiegand – which was the third working draft released by JCT-VC – therefore could have been located by searching each meeting for, e.g., "working draft" or "wd" on the JCT-VC website. *See, e.g.*, Ex-1059. When I performed a search for "working draft," the first results appeared in Meeting No. 3, which is when the first working draft was published. I then repeated this search for Meeting Nos. 4–7, and each produced between one to two documents. The full title of each document

provided in the results then allowed me to locate the working draft that were published after each meeting through Meeting No. 7. *See id.* In my opinion, then, locating the different working drafts at the time would have not only been straightforward, but one of the natural starting points to investigate the ongoing development of the HEVC standard.

67. From there, if one were interested in locating proposals for particular topics under consideration for subsequent drafts of the standard, searching for those would have been straightforward as well. For instance, I had mentioned as an example searching for all proposals related to "deblocking." If one were instead interested in, e.g., locating all proposals that discussed "adaptive resolution" (or switching resolutions of a video frame), one could search through the meetings using the term "resolution." The first results of this search appear in Meeting No. 3, and show just five results. Ex-1054. Repeating this search for Meeting Nos. 4–7 each also produced a limited number of results: Meeting No. 4 had eight results, Ex-1056; Meeting No. 5 had none, Ex-1058; Meeting No. 6 had five results (Davies being one of them), Ex-1061; and Meeting No. 7 had twelve, Ex-1062. The relevance of each result in the search could then be determined by examining its full title. For instance, here, many results for "resolution" had to do with features other than "adaptive resolution," and therefore could be ignored.

68. Furthermore, searching this way made it relatively easy to track developments for a particular proposal. For instance, searching "resolution" in Meeting No. 6 revealed not only Davies, but also a "Cross-check report for JCTVC-F158 [(Davies)] on resolution switching." Ex-1061. That meant someone else had investigated and checked Davies's proposal. And searching "resolution" for Meeting No. 7 showed further developments, such as that an adaptive-resolution proposal appears to have been submitted to an Ad Hoc Group ("AHG"), Ex-1062, for further consideration. Ex-1038, 2 (explaining Ad Hoc Groups). And the search also shows that as part of that AHG, others had submitted comments, Misra being one of them. Ex-1062.

69. Therefore, it was relatively easy to locate and track specific proposals on specific topics using JCT-VC's built-in search feature. And in my opinion, those interested could have located Wiegand (Ex-1009), Davies (Ex-1010), and other documents published as part of the JCT-VC's standardization process (such as Misra (Ex-1013) and Demircin (Ex-1063)) with reasonable diligence if inspired to search for their relevant topics.

## V. Additional Remarks

70. I currently hold the opinions set expressed in this declaration. But my analysis may continue, and I may acquire additional information and/or attain supplemental insights that may result in added observations.

# APPENDIX 1

**Joint Collaborative Team on Video Coding (JCT-VC)**
**of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**
5th Meeting: Geneva, CH, 16-23 March, 2011

Document: JCTVC-E603

*Title:* **WD3: Working Draft 3 of High-Efficiency Video Coding**

*Status:* Output Document of JCT-VC

*Purpose:* Working Draft of HEVC

*Author(s) or Contact(s):*

Thomas Wiegand, Email: thomas.wiegand@hhi.fraunhofer.de
Fraunhofer HHI / TU Berlin

Woo-Jin Han, Email: wjhan.han@samsung.com
Samsung Electronics

Benjamin Bross, Email: benjamin.bross@hhi.fraunhofer.de
Fraunhofer HHI

Jens-Rainer Ohm, Email: ohm@ient.rwth-aachen.de
RWTH Aachen

Gary J. Sullivan, Email: garysull@microsoft.com
Microsoft

*Source:* Editor

---

## Abstract

Working Draft 3 of High-Efficiency Video Coding.

Ed. Notes (WD3):

- Added Residual coding CABAC syntax and semantics
- Added Zig-zag scanning process
- Added CABAC Binarization processes
- Incorporated MV coding (JCTVC-E481)
- Incorporated Compression of reference indices (JCTVC-E059)
- Incorporated Zero merge candidate (JCTVC-E146)
- Incorporated Intra mode coding (JCTVC-E088/E131) (Inserted by TK 31/3/2011 with notes)
- Fixed the CABAC coefficients syntax, semantics and inverse scanning process
- Incorporated CABAC coeffs (JCTVC-E253)
- Moved the EGk binarization from the UEGk subclause in a separate subclause
- Added text representing CABAC entropy coding context initialisation
- Added text representing CABAC entropy coding context derivation.
- Mode-dependent 3- scan for intra (JCTVC-D393)
- Incorporated CABAC: Context size reduction (JCTVC-E227/E489)
- Incorporated CABAC: significance map coding simplification (JCTVC-E227/E338/E344/E494)
- Incorporated CABAC: Contexts for MVD (JCTVC-E324)
- Incorporated initial draft of CAVLC text
- CAVLC for 16x16 & 32x32 (JCTVC-E383)
- CAVLC table size reduction (JCTVC-E384)
- CAVLC for RQT (JCTVC-E404)
- CAVLC: counters (JCTVC-E143)
- CAVLC: Intra prediction mode coding in LCEC (JCTVC-D366)
- CAVLC: Inter prediction mode coding in LCEC (JCTVC- D370)
- CAVLC: 4x4 and 8x8 transform coefficient coding in LCEC (JCTVC- D374)
- Block-based ALF (JCTVC-E046/E323)
- ALF parameters to PPS (JCTVC-E045)
- Parallel deblocking (JCTVC-E496/E181/E224)
- Clipping for bi-pred averaging (JCTVC-E242)
- Reference sample padding (JCTVC-E488)
- Transformation processes are replaced by [TBD] mark (meeting note, JCTVC-E243)

- Sub-LCU-level dQP (JCTVC-E051/E220)
- Temporal layer switching and reference list management based on temporal_id (JCTVC-E279/D081/D200)
- Improved text of entropy slice (JCTVC-D070) [Ed. (WJ): the term entropy slice is replaced by lightweight slice according to proponent's text]
- Slice independent deblocking and adaptive loop filtering (JCTVC-D128)
- Fine-granularity slices (JCTVC-E483)
- PCM mode (JCTVC-E057 and JCTVC-E192)
- CAVLC: Inter pred coding (JCTVC-E381)
- CAVLC: Combined coding of inter prediction direction and reference frame index (JCTVC-D141)
- 4x4 DST (JCTVC-E125)
- Planar mode (JCTVC-E321)
- Luma-based chroma intra prediction (JCTVC-E266)
- Modification of DC predictor (JCTVC-E069)
- Bug in mapping table and the corresponding text for mostProbableIntra was fixed. (64x64 uses 3-directions, but the table was specified for 5-directions)
- Non-existing cases of Intra_DC, Intra_Planar and Intra_FromLuma are removed (due to reference sample padding, JCTVC-E488)

Ed. Notes (WD2):

- Incorporated Partial Merging according to JCTVC-D441
  - removed direct mode
  - moved merge to prediction_unit and added candidates
  - added partial merge restrictions
  - inter NxN partitioning only for smallest coding_unit
- Updated transform_tree and transform_coeff syntax
- Added transform_coeff to coding_unit syntax (Fix)
- Incorporated intra NxN partitioning only for smallest coding_unit according to JCTVC-D432
- Incorporated modified temporal motion vector predition according to JCTVC-D164
- Incorporated simplified motion vector prediction according to JCTVC-D231
  - removed median
  - removed pruning process
  - changed the selection manner of left/top predictor
- 8-tap luma interpolation filter according to JCTVC-D344
- 4-tap chroma interpolation filter according to JCTVC-D347
- Improved deblocking filter text according to JCTVC-D395
- IBDI syntax is removed
- Updated syntax and semantics
  - Two tool-enabling flags (adaptive_loop_filter_enabled_flag and cu_qp_delta_enabled_flag) are added in SPS according to software. However, low_delay_coding_enabled_flag is not added – it could be handled by more general reference management scheme. merging_enabled_flag is not added – partial merging (JCTVC-D441) was adopted thus merging cannot be turned off any more. amvp_mode[] is not added since amvp cannot be turned off any more due to absence of median predictor (JCTVC-D231). Note that software has all switches.
  - cu_qp_delta (coding unit layer), syntax and semantics are added. (JCTVC-D258)
  - collocated_from_l0 (slice header), syntax and semantics are added.
- Clean decoding refresh (CDR) (JCTVC-D234).
- Temporal motion vector memory compression (JCTVC-D072)
- Constrained intra prediction (JCTVC-D086)
- Mode-dependent intra smoothing (JCTVC-D282)
- Merging chroma intra prediction process into luma intra prediction process
- Combined reference list (JCTVC-D421)
- Chroma intra prediction mode reordering (JCTVC-D255/D278/D166)
- Adaptive loop filter text is added
- Entropy slice is added (JCTVC-D070)
- High precision bi-directional averaging (JCTVC-D321)
- Reduction of number of intra prediction modes for 64x64 blocks (JCTVC-D100)
- Misc.
  - TPE bits are reduced from 4 to 2
  - Clipping is applied to (temporally) scaled mv – revisit

Ed. Notes (WD1):

- Incorporated the decisions on high-level syntax according to JCTVC-B121
- Incorporated text from JCTVC-B205revision7
- Incorporated text from JCTVC-C319 (as found to be stable)
- Revised coding tree, coding unit and prediction unit syntaxes (coding tree syntax is newly added. needs to be confirmed)
- Initial drafting of decoding process of coding units in intra prediction mode (luma part, JCTVC-B100 and JCTVC-C042)
- Initial drafting of decoding process of coding units in inter prediction mode
- Initial drafting of scaling and transformation process
- Added text, transform 16T and 32T
- Initial drafting of deblocking process
- Improving the text, derivation process for motion vector components and reference indices
- Added text, boundary filtering strength

Open issues:

- Substantial portions of JCTVC-B205 have not been imported so far, as they require significant editorial work
- Should support for monochrome, 4:2:2 and 4:4:4 (with and w/o separate color planes) be included from the start? Currently, it has been left in the text as it doesn't seem to affect much text.
- Handling of the term "frame". We are currently leaning towards changing all occurrences of "frame" to "picture" (removed or marked all occurrences of "field")
- Large size table (de-scaling matrices) is not inserted yet.
- Slice-header syntaxes and their semantics are not completed yet. Also possible modifications that maybe necessary because of larger treeblocks (64x64) compared to macroblocks (16x16) are not yet considered.
- Text representing entropy coding named as "LCEC" are missing. The name LCEC needs to be adjusted as it propotes a property ("Low Complexity"). Currently, there is a place holder with CAVLC.
- Clipping is applied to temporally scaled motion vector in 4. Do we need this? (c.f. AVC seems not)
- SPS- and slice-level syntax items are significantly different from software in terms of both meaning and position. Needs to be unified.
- Use of bin string and bit string should be consistent.
- MD5 hash (JCTVC-E490)
- New additional loop filter (JCTVC-E049)
- ALF coefficient derivation from syntax elements
- SPS/PPS/slice-header syntax tables (with synchronization of software)
- Improve text quality by considering: strict use of terms “unit” and “block” – block = a rectangular 2D array (one component), unit = collective term for specifying information for both luma and chroma. Don’t use ther term ‘unit’ by itself – always use the term ‘unit’ with prefix – coding unit, prediction unit or transform unit.
- Syntax functions cuAddress and granularity_block_boundary (fine granularity slices, JCTVC-E483) are missing
- Do we need SPS flag for disabling PCM mode? Currently PCM mode is turned on for all cases.
- Software seems to have SPS syntax to turn on/off combined reference list (JCTVC-D421), but currently text does not.
- Both variables IntraPredMode and IntraPredModeC are used in the syntax table but the actual derivations are specified in the decoding process. Maybe it’s better to move them to the semantics section.
- Binarization of intra_chroma_pred_mode reflecting codeword switching and luma-based chroma intra prediction (JCTVC-E266) is missing.
- Clarification on the use of intra_chroma_pred_mode and IntraPredModeC is needed. The former specifies the syntax item to indicate how to determine the chroma intra prediction mode (IntraPredModeC) as 0 (to Intra_FromLuma), 1 (to Intra_Vertical), 2 (to Intra_Horizontal), 3 (Intra_DC or Intra_Planar) or 4 (re-use luma mode). The latter specifies chroma intra prediction mode, which is actually mapped to the specific prediction process.

**Deleted:** 8.4.2.1.7

**Deleted:** <#>Luma-based chroma intra prediction (JCTVC-E266)¶

**Deleted:** <#>Planar mode (JCTVC-E321)¶

**Deleted:** <#>4x4 DST (JCTVC-E125)¶

**Deleted:** <#>Modification of DC predictor (JCTVC-E069)¶

**Deleted:** <#>CAVLC: Inter pred coding (JCTVC-E381)¶
<#>CAVLC: Combined coding of inter prediction direction and reference frame index (JCTVC-D141)¶

**Deleted:** <#>entropy slice - improved text (JCTVC- D070)¶

**Deleted:** <#>slice independent deblocking and adaptive loop filtering (JCTVC- D128)¶

**Deleted:** <#>PCM mode (JCTVC-E057 and JCTVC-E192)¶

**Deleted:** Add CU, PU and TU definitions and replace treeblock with LCUs

**Deleted:** <#>CAVLC: Combined coding of inter prediction direction and reference frame index (JCTVC-D141) – corresponding software implementation seems to have more things (check of neighbouring reference indices) which is not described in JCTVC-D141, thus the relevent text was not included yet.¶

**CONTENTS**


*Page*

Abstract ..... i

0 Introduction ..... 12
0.1 Prologue ..... 12
0.2 Purpose ..... 12
0.3 Applications ..... 12
0.4 Publication and versions of this Specification ..... 12
0.5 Profiles and levels ..... 12
0.6 Overview of the design characteristics ..... 13
0.7 How to read this Specification ..... 13

1 Scope ..... 14

2 Normative references ..... 14

3 Definitions ..... 15

4 Abbreviations ..... 20

5 Conventions ..... 21
5.1 Arithmetic operators ..... 21
5.2 Logical operators ..... 21
5.3 Relational operators ..... 21
5.4 Bit-wise operators ..... 22
5.5 Assignment operators ..... 22
5.6 Range notation ..... 22
5.7 Mathematical functions ..... 22
5.8 Order of operation precedence ..... 23
5.9 Variables, syntax elements, and tables ..... 24
5.10 Text description of logical operations ..... 25
5.11 Processes ..... 26

6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships ..... 26
6.1 Bitstream formats ..... 26
6.2 Source, decoded, and output picture formats ..... 26
6.3 Spatial subdivision of pictures and slices ..... 28
6.4 Inverse scanning processes and derivation processes for neighbours ..... 29
6.4.1 Inverse treeblock scanning process ..... 29
6.5 Zig-zag scanning array initialisation process ..... 30
6.6 Scanning order array initialisation process ..... 31

7 Syntax and semantics ..... 31
7.1 Method of specifying syntax in tabular form ..... 31
7.2 Specification of syntax functions, categories, and descriptors ..... 32
7.3 Syntax in tabular form ..... 34
7.3.1 NAL unit syntax ..... 34
7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax ..... 35
7.3.2.1 Sequence parameter set RBSP syntax ..... 35
7.3.2.2 Picture parameter set RBSP syntax ..... 36
7.3.2.3 Supplemental enhancement information RBSP syntax ..... 36
7.3.2.4 Access unit delimiter RBSP syntax ..... 37
7.3.2.5 Filler data RBSP syntax ..... 37
7.3.2.6 Slice layer RBSP syntax ..... 37
7.3.2.7 RBSP slice trailing bits syntax ..... 38
7.3.2.8 RBSP trailing bits syntax ..... 38
7.3.3 Slice header syntax ..... 39
7.3.3.1 Reference picture list modification syntax ..... 40
7.3.3.2 Reference picture lists combination syntax ..... 40
7.3.3.3 Decoded reference picture marking syntax ..... 41
7.3.3.4 Adaptive loop filter parameter syntax ..... 41


Deleted: 11
Deleted: 11
Deleted: 11
Deleted: 11
Deleted: 11
Deleted: 11
Deleted: 12
Deleted: 12
Deleted: 13
Deleted: 13
Deleted: 13
Deleted: 18
Deleted: 19
Deleted: 19
Deleted: 19
Deleted: 20
Deleted: 20
Deleted: 20
Deleted: 20
Deleted: 20
Deleted: 21
Deleted: 22
Deleted: 23
Deleted: 24
Deleted: 24
Deleted: 24
Deleted: 24
Deleted: 26
Deleted: 27
Deleted: 27
Deleted: 28
Deleted: 28
Deleted: 29
Deleted: 29
Deleted: 30
Deleted: 32
Deleted: 32
Deleted: 33
Deleted: 33
Deleted: 33
Deleted: 34
Deleted: 34
Deleted: 34
Deleted: 35
Deleted: 35
Deleted: 35
Deleted: 36
Deleted: 37
Deleted: 37
Deleted: 38
Deleted: 39

7.3.4 Slice data syntax ..... 44
7.3.5 Coding tree syntax ..... 45
7.3.6 Coding unit syntax ..... 46
7.3.7 Prediction unit syntax ..... 47
7.3.8 Transform tree syntax ..... 49
7.3.9 Transform coefficient syntax ..... 51
7.3.10 Residual coding CABAC syntax ..... 52
7.3.11 Residual coding CAVLC syntax ..... 52
7.3.12 Level and sign CAVLC syntax ..... 53
7.4 Semantics ..... 54
7.4.1 NAL unit semantics ..... 54
7.4.1.1 Encapsulation of an SODB within an RBSP (informative) ..... 56
7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences ..... 57
7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics ..... 61
7.4.2.1 Sequence parameter set RBSP semantics ..... 61
7.4.2.2 Picture parameter set RBSP semantics ..... 63
7.4.2.3 Supplemental enhancement information RBSP semantics ..... 64
7.4.2.4 Access unit delimiter RBSP semantics ..... 64
7.4.2.5 Filler data RBSP semantics ..... 64
7.4.2.6 Slice layer RBSP semantics ..... 65
7.4.2.7 RBSP slice trailing bits semantics ..... 65
7.4.2.8 RBSP trailing bits semantics ..... 65
7.4.3 Slice header semantics ..... 65
7.4.3.1 Reference picture list modification semantics ..... 68
7.4.3.2 Reference picture lists combination semantics ..... 68
7.4.3.3 Decoded reference picture marking semantics ..... 69
7.4.3.4 Adaptive loop filter parameter semantics ..... 71
7.4.4 Slice data semantics ..... 73
7.4.5 Coding tree semantics ..... 73
7.4.6 Coding unit semantics ..... 74
7.4.7 Prediction unit semantics ..... 75
7.4.8 Transform tree semantics ..... 78
7.4.9 Transform coefficient semantics ..... 80
7.4.10 Residual coding CABAC semantics ..... 81
7.4.11 Residual coding CAVLC semantics ..... 82
8 Decoding process ..... 83
8.1 NAL unit decoding process ..... 83
8.2 Slice decoding process ..... 84
8.2.1 Decoding process for picture order count ..... 84
8.2.1.1 Decoding process for picture order count type 0 ..... 84
8.2.1.2 Decoding process for picture order count type 1 ..... 84
8.2.1.3 Decoding process for picture order count type 2 ..... 84
8.2.2 Decoding process for reference picture lists construction ..... 84
8.2.2.1 Decoding process for picture numbers ..... 84
8.2.2.2 Initialisation process for reference picture lists ..... 85
8.2.2.3 Modification process for reference picture lists ..... 86
8.2.2.4 Mapping process for reference picture lists combination in B slices ..... 88
8.2.3 Decoded reference picture marking process ..... 89
8.2.3.1 Sequence of operations for decoded reference picture marking process ..... 89
8.2.3.2 Decoding process for gaps in frame_num ..... 90
8.2.3.3 Sliding window decoded reference picture marking process ..... 90
8.2.3.4 Adaptive memory control decoded reference picture marking process ..... 90
8.3 Decoding process for coding units coded in intra prediction mode ..... 91
8.3.1 Derivation process for luma intra prediction mode ..... **Error! Bookmark not defined.**
8.3.2 Derivation process for chroma intra prediction mode ..... 95
8.3.3 Decoding process for intra blocks ..... 96
8.3.3.1 Intra sample prediction ..... 97
8.4 Decoding process for coding units coded in inter prediction mode ..... 105
8.4.1 Inter prediction process ..... 105
8.4.2 Decoding process for prediction units in inter prediction mode ..... 107
8.4.2.1 Derivation process for motion vector components and reference indices ..... 108
8.4.2.2 Decoding process for inter prediction samples ..... 119



**Deleted:** 40
**Deleted:** 41
**Deleted:** 41
**Deleted:** 43
**Deleted:** 45
**Deleted:** 47
**Deleted:** 48
**Deleted:** 48
**Deleted:** 49
**Deleted:** 50
**Deleted:** 50
**Deleted:** 52
**Deleted:** 53
**Deleted:** 57
**Deleted:** 57
**Deleted:** 58
**Deleted:** 59
**Deleted:** 59
**Deleted:** 59
**Deleted:** 59
**Deleted:** 59
**Deleted:** 60
**Deleted:** 60
**Deleted:** 62
**Deleted:** 63
**Deleted:** 63
**Deleted:** 66
**Deleted:** 67
**Deleted:** 67
**Deleted:** 68
**Deleted:** 69
**Deleted:** 71
**Deleted:** 72
**Deleted:** 73
**Deleted:** 74
**Deleted:** 75
**Deleted:** 76
**Deleted:** 76
**Deleted:** 76
**Deleted:** 76
**Deleted:** 76
**Deleted:** 76
**Deleted:** 76
**Deleted:** 77
**Deleted:** 77
**Deleted:** 78
**Deleted:** 80
**Deleted:** 81
**Deleted:** 81
**Deleted:** 82
**Deleted:** 82
**Deleted:** 82
**Deleted:** 83
**Deleted:** 84
**Deleted:** 86
**Deleted:** 86
**Deleted:** 87
**Deleted:** 93
**Deleted:** 93
**Deleted:** 95
**Deleted:** 96
**Deleted:** 107

8.4.3 Decoding process for the residual signal of coding units coded in inter prediction mode ..... 126
8.4.3.1 Decoding process for luma residual blocks ..... 126
8.4.3.2 Decoding process for chroma residual blocks ..... 127
8.5 Scaling, transformation and array construction process prior to deblocking filter process ..... 128
8.5.1 Scaling and transformation process ..... 128
8.5.2 Inverse scanning process for transform coefficients ..... 129
8.5.3 Scaling process for transform coefficients ..... 129
8.5.4 Transformation process for scaled transform coefficients ..... 130
8.5.4.1 Transformation process for 4 samples ..... 131
8.5.4.2 Transformation process for 8 samples ..... 131
8.5.4.3 Transformation process for 16 samples ..... 131
8.5.4.4 Transformation process for 32 samples ..... 132
8.5.5 Array construction process ..... 132
8.6 In-loop filter process ..... 132
8.6.1 Deblocking filter process ..... 132
8.6.1.1 Derivation process of transform unit boundary ..... 133
8.6.1.2 Derivation process of prediction unit boundary ..... 134
8.6.1.3 Derivation process of boundary filtering strength ..... 134
8.6.1.4 Filtering process for coding unit ..... 135
8.6.2 Adaptive loop filter process ..... 143
8.6.2.1 Derivation process for filter coefficients ..... 145
8.6.2.2 Derivation process for filter index array for luma samples ..... 146
8.6.2.3 Filtering process for luma samples ..... 147
8.6.2.4 Filtering process for chroma samples ..... 148
9 Parsing process ..... 149
9.1 Parsing process for Exp-Golomb codes ..... 149
9.1.1 Mapping process for signed Exp-Golomb codes ..... 150
9.2 CAVLC parsing process for slice data ..... 151
9.2.1 Parsing process for VLC codes ..... 151
9.2.2 Initialisation process ..... 153
9.2.2.1 Counter initialisation for codeword index adaptive mapping ..... 153
9.2.2.2 Initialisation process for lastPosVlcNumIndex ..... 154
9.2.2.3 Initialisation process for lastPosTable ..... 154
9.2.2.4 Initialisation process for splitPredPartModeTable ..... 154
9.2.2.5 Initialisation process for intraModeTable ..... 154
9.2.2.6 Initialisation process for cbpSplitTransTable ..... 155
9.2.3 Codeword index mapping update process ..... 155
9.2.4 Parsing process for transform coefficient syntax elements ..... 156
9.2.4.1 Parsing process for last_pos_level_one ..... 156
9.2.4.2 Parsing process for level_minus2_and_sign ..... 157
9.2.4.3 Parsing process for run_level_one ..... 157
9.2.4.4 Parsing process for level ..... 161
9.2.4.5 Parsing process for cu_split_pred_part_mode ..... 161
9.2.4.6 Parsing process for rem_intra_luma_pred_mode ..... 162
9.2.4.7 Parsing process for cbp_and_split_transform ..... 162
9.3 CABAC parsing process for slice data ..... 163
9.3.1 Initialisation process ..... 164
9.3.1.1 Initialisation process for context variables ..... 164
9.3.1.2 Initialisation process for the arithmetic decoding engine ..... 175
9.3.2 Binarization process ..... 176
9.3.2.1 Unary (U) binarization process ..... 181
9.3.2.2 Truncated unary (TU) binarization process ..... 181
9.3.2.3 Truncated Rice (TR) binarization process ..... 181
9.3.2.4 k-th order Exp-Golomb (EGk) binarization process ..... 182
9.3.2.5 Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process ..... 182
9.3.2.6 Fixed-length (FL) binarization process ..... 183
9.3.2.7 Binarization process for cu_qp_delta ..... 183
9.3.2.8 Binarization process for pred_type ..... 183
9.3.2.9 Binarization process for rem_intra_luma_pred_mode ..... 183
9.3.2.10 Binarization process for coeff_abs_level_minus3 ..... 184
9.3.3 Decoding process flow ..... 184
9.3.3.1 Derivation process for ctxIdx ..... 185



Deleted: 114
Deleted: 114
Deleted: 115
Deleted: 116
Deleted: 116
Deleted: 117
Deleted: 117
Deleted: 118
Deleted: 118
Deleted: 118
Deleted: 118
Deleted: 121
Deleted: 128
Deleted: 128
Deleted: 128
Deleted: 129
Deleted: 129
Deleted: 130
Deleted: 131
Deleted: 136
Deleted: 136
Deleted: 137
Deleted: 138
Deleted: 139
Deleted: 139
Deleted: 140
Deleted: 141
Deleted: 142
Deleted: 142
Deleted: 144
Deleted: 144
Deleted: 145
Deleted: 145
Deleted: 145
Deleted: 145
Deleted: 146
Deleted: 146
Deleted: 147
Deleted: 147
Deleted: 148
Deleted: 148
Deleted: 152
Deleted: 152
Deleted: 153
Deleted: 153
Deleted: 154
Deleted: 155
Deleted: 155
Deleted: 165
Deleted: 166
Deleted: 169
Deleted: 170
Deleted: 170
Deleted: 170
Deleted: 171
Deleted: 171
Deleted: 171
Deleted: 171
Deleted: 172
Deleted: 172
Deleted: 173
Deleted: 173

9.3.3.2 Arithmetic decoding process ..... 194
9.3.4 Arithmetic encoding process (informative) ..... 200
9.3.4.1 Initialisation process for the arithmetic encoding engine (informative) ..... 200
9.3.4.2 Encoding process for a binary decision (informative) ..... 201
9.3.4.3 Renormalization process in the arithmetic encoding engine (informative) ..... 202
9.3.4.4 Bypass encoding process for binary decisions (informative) ..... 204
9.3.4.5 Encoding process for a binary decision before termination (informative) ..... 205
9.3.4.6 Byte stuffing process (informative) ..... 207
Annex A Profiles and levels ..... 208
A.1 Requirements on video decoder capability ..... 208
A.2 Profiles ..... 208
A.2.1 ABC profile ..... 208
A.3 Levels ..... 208
Annex B Byte stream format ..... 210
B.1 Byte stream NAL unit syntax and semantics ..... 210
B.1.1 Byte stream NAL unit syntax ..... 210
B.1.2 Byte stream NAL unit semantics ..... 210
B.2 Byte stream NAL unit decoding process ..... 211
B.3 Decoder byte-alignment recovery (informative) ..... 211
Annex C Hypothetical reference decoder ..... 212
C.1 Operation of coded picture buffer (CPB) ..... 213
C.1.1 Timing of bitstream arrival ..... 213
C.1.2 Timing of coded picture removal ..... 213
C.2 Operation of the decoded picture buffer (DPB) ..... 213
C.3 Bitstream conformance ..... 213
C.4 Decoder conformance ..... 213
Annex D Supplemental enhancement information ..... 214
D.1 SEI payload syntax ..... 214
D.2 SEI payload semantics ..... 214
Annex E Video usability information ..... 215
E.1 VUI syntax ..... 215
E.1.1 VUI parameters syntax ..... 215
E.2 VUI semantics ..... 215

## LIST OF FIGURES

Figure 6-2 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture [Ed. Re-draw figure] ..... 25
Figure 6-3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture [Ed. Re-draw figure] ..... 25
Figure 6-4 – A picture with 11 by 9 treeblocks that is partitioned into two slices ..... 29
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 to 31, inclusive ..... 60
Figure 8-2 – Intra prediction angle definition (informative) ..... 102
Figure 8-3 – Spatial neighbours that can be used as merging candidates (informative) illustrates the position of the spatial neighbours A, B, C and D relative to the current prediction unit. ..... 111
Figure 8-4 – Spatial neighbours, that can be used to derive reference indices for temporal merging candidates, (informative) illustrates the position of the spatial neighbours A, B, C, D and E relative to the current prediction unit. ..... 113
Figure 8-5 – Spatial motion vector neighbours ..... 115
Figure 8-6 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation ..... 121
Figure 8-8 Mapping between geometric position and luma adaptive loop filter index according to alfTap (informative) ..... 143

Deleted: 182
Deleted: 189
Deleted: 189
Deleted: 190
Deleted: 191
Deleted: 193
Deleted: 194
Deleted: 196
Deleted: 197
Deleted: 197
Deleted: 197
Deleted: 197
Deleted: 197
Deleted: 199
Deleted: 199
Deleted: 199
Deleted: 199
Deleted: 200
Deleted: 200
Deleted: 201
Deleted: 202
Deleted: 202
Deleted: 202
Deleted: 202
Deleted: 202
Deleted: 202
Deleted: 203
Deleted: 203
Deleted: 203
Deleted: 204
Deleted: 204
Deleted: 204
Deleted: 204
Deleted: 26
Deleted: 26
Deleted: 27
Deleted: 56
Deleted: 92
Deleted: 99
Deleted: 101
Deleted: 103
Deleted: 109
Deleted: 139

Figure 9-1 – Overview of the arithmetic decoding process for a single bin (informative) ..... 194
Figure 9-2 – Flowchart for decoding a decision ..... 196
Figure 9-3 – Flowchart of renormalization ..... 198
Figure 9-4 – Flowchart of bypass decoding process ..... 199
Figure 9-5 – Flowchart of decoding a decision before termination ..... 200
Figure 9-6 – Flowchart for encoding a decision ..... 202
Figure 9-7 – Flowchart of renormalization in the encoder ..... 203
Figure 9-8 – Flowchart of PutBit(B) ..... 204
Figure 9-9 – Flowchart of encoding bypass ..... 205
Figure 9-10 – Flowchart of encoding a decision before termination ..... 206
Figure 9-11 – Flowchart of flushing at termination ..... 207
Figure C-9-12 – Structure of byte streams and NAL unit streams for HRD conformance checks ..... 212
Figure C-9-13 – HRD buffer model ..... 213

Deleted: 183
Deleted: 185
Deleted: 187
Deleted: 188
Deleted: 189
Deleted: 191
Deleted: 192
Deleted: 193
Deleted: 194
Deleted: 195
Deleted: 196
Deleted: 201
Deleted: 202

## LIST OF TABLES

Table 5-1 – Operation precedence from highest (at top of table) to lowest (at bottom of table) ..... 24
Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag ..... 27
Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes ..... 55
Table 7-2 – Meaning of primary_pic_type ..... 64
Table 7-3 – Name association to slice_type ..... 65
Table 7-4 – modification_of_pic_nums_idc operations for modification of reference picture lists ..... 68
Table 7-5 – Interpretation of adaptive_ref_pic_marking_mode_flag ..... 70
Table 7-6 – Memory management control operation (memory_management_control_operation) values ..... 71
Table 7-7 – Specification of alf_chroma_idc ..... 73
Table 7-8 – Specification of cu_split_pred_part_mode when log2CUSize is greater than Log2MinCUSize ..... 74
Table 7-9 – Specification of cu_split_pred_part_mode when log2CUSize is equal to Log2MinCUSize ..... 74
Table 7-10 - Name association to prediction mode and partitioning type ..... 75
Table 7-11 – Number of bins and number of mode for rem_intra_luma_pred_mode associated to PuSize ..... 76
Table 7-12 – Name association to inter prediction mode ..... 77
Table 7-13 – Specification of cbp_and_split_transform ..... 80
Table 7-14 – Specification of ScanType[ log2TrafoSize – 2 ][ IntraPredMode ] ..... 81
Table 7-15 – Specification of ScanTypeC[ log2TrafoSize – 2 ][ IntraPredModeC ] ..... 81
Table 8-1 – Specification of intraPredModeNum ..... 93
Table 8-2 – Specification of mapIntraPredMode5 and mapIntraPredMode9 ..... 93
Table 8-3 – Specification of IntraPredModeC according to the values of intra_chroma_pred_mode and IntraPredMode[ xB ][ yB ] ..... 96
Table 8-4 – Specification of intraPredMode and associated names ..... 98
Table 8-5 – Specification of intraFilterType[ nS ][ IntraPredMode ] for various prediction unit sizes ..... 99
Table 8-6 – Specification of intraPredOrder ..... 101
Table 8-7 – Specification of intraPredAngle ..... 102

Deleted: 23
Deleted: 26
Deleted: 52
Deleted: 60
Deleted: 61
Deleted: 64
Deleted: 65
Deleted: 66
Deleted: 68
Deleted: 69
Deleted: 69
Deleted: 70
Deleted: 70
Deleted: 71
Deleted: 73
Deleted: 74
Deleted: 74
Deleted: 85
Deleted: 86
Deleted: 87
Deleted: 89
Deleted: 90
Deleted: 92
Deleted: 93

Table 8-8 – Specification of invAngle ..... 102
Table 8-9 – Assignment of the luma prediction sample predSampleLX$_L$[ $x_L$, $y_L$ ] ..... 112
Table 8-10 – Assignment of the chroma prediction sample predSampleLX$_C$[ $x_C$, $y_C$ ] for ( X, Y ) being replaced by ( 1, b ), ( 2, c ), ( 3, d ), ( 4, e ), ( 5, f ), ( 6, g ), and ( 7, h ), respectively ..... 114
Table 8-11 – Derivation of threshold variables β and $t_C$ from input Q ..... 136
Table 8-12 – Specification of horPos[ i ] according to alfTap for adaptive loop filter process of luma samples ..... 139
Table 8-13 – Specification of verPos[ i ] according to alfTap for adaptive loop filter process of luma samples ..... 139
Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative) ..... 141
Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative) ..... 142
Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v) ..... 143
Table 9-4 – codeNum and codeword with vlcNum equal to 15 ..... 145
Table 9-5 – Specification of initial values of lastPosTable[tableNum][codeNum] ..... 146
Table 9-6 – Specification of intraModeTable[k][codeNum] ..... 146
Table 9-7 – Specification of initial values of cbpSplitTransTable[flagPattern-8][n][codeNum] ..... 147
Table 9-8 – Specification of lastPosVlcNumTable[blockType][vlcNumIdx] ..... 149
Table 9-9 – Derivation of vlcNum from tableIdx and maxRunIdx ..... 150
Table 9-10 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is equal to 4 ..... 151
Table 9-11 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is greater than 4 ..... 151
Table 9-12 – Derivation of runOfZeros from maxRun and codeNum ..... 151
Table 9-13 – Derivation of levelThreshold from vlcIdx ..... 153
Table 9-14 – Specification of cbpVlcNumTable[cbpVlcNumIdx] ..... 155
Table 9-15 – Derivation of cMax from flagPattern and n ..... 155
Table 9-16 – Specification of cbpSplitTransTable and counterNum associated ..... 155
Table 9-17 – Association of ctxIdx and syntax elements for each slice type in the initialisation process ..... 157
Table 9-18 – Values of variables m and n for alf_cu_flag ctxIdx ..... 157
Table 9-19 – Values of variables m and n for split_coding_unit_flag ctxIdx ..... 158
Table 9-20 – Values of variables m and n for skip_flag ctxIdx ..... 158
Table 9-21 – Values of variables m and n for cu_qp_delta ctxIdx ..... 158
Table 9-22 – Values of variables m and n for pred_type ..... 158
Table 9-23 – Values of variable m and n for prev_intra_luma_pred_flag ctxIdx ..... 158
Table 9-24 – Values of variable m and n for rem_intra_luma_pred_mode ctxIdx ..... 158
Table 9-25 – Values of variable m and n for intra_chroma_pred_mode ctxIdx ..... 159
Table 9-26 – Values of variable m and n for merge_flag ctxIdx ..... 159
Table 9-27 – Values of variable m and n for merge_idx ctxIdx ..... 159
Table 9-28 – Values of variable m and n for inter_pred_flag ctxIdx ..... 159
Table 9-29 – Values of variable m and n for ref_idx_lc, ref_idx_l0, ref_idx_l1 ctxIdx ..... 159
Table 9-30 – Values of variable m and n for mvd_lc, mvd_l0, mvd_l1 ctxIdx ..... 159
Table 9-31 – Values of variable m and n for mvp_idx_lc, mvp_idx_l0, mvp_idx_l1 ctxIdx ..... 160
Table 9-32 – Values of variable m and n for no_residual_data_flag ctxIdx ..... 160
Table 9-33 – Values of variable m and n for split_transform_flag ctxIdx ..... 160

Deleted: 93
Deleted: 112
Deleted: 114
Deleted: 136
Deleted: 139
Deleted: 139
Deleted: 141
Deleted: 142
Deleted: 143
Deleted: 145
Deleted: 146
Deleted: 146
Deleted: 147
Deleted: 149
Deleted: 150
Deleted: 151
Deleted: 151
Deleted: 151
Deleted: 153
Deleted: 155
Deleted: 155
Deleted: 155
Deleted: 157
Deleted: 157
Deleted: 158
Deleted: 158
Deleted: 158
Deleted: 158
Deleted: 158
Deleted: 158
Deleted: 159
Deleted: 159
Deleted: 159
Deleted: 159
Deleted: 159
Deleted: 159
Deleted: 160
Deleted: 160
Deleted: 160

Table 9-34 – Values of variable m and n for cbf_luma ctxIdx ..... 169
Table 9-35 – Values of variable m and n for cbf_cb ctxIdx ..... 169
Table 9-36 – Values of variable m and n for cbf_cr ctxIdx ..... 169
Table 9-37 – Values of variable m and n for last_significant_coefficient_x ctxIdx ..... 170
Table 9-38 – Values of variable m and n for last_significant_coefficient_y ctxIdx ..... 171
Table 9-39 – Values of variable m and n for significant_coeff_flag ctxIdx (A) ..... 172
Table 9-40 – Values of variable m and n for significant_coeff_flag ctxIdx (B) ..... 172
Table 9-41 – Values of variable m and n for significant_coeff_flag ctxIdx (C) ..... 173
Table 9-42 – Values of variable m and n for coeff_abs_level_greater1_flag ctxIdx ..... 174
Table 9-43 – Values of variable m and n for coeff_abs_level_greater2_flag ctxIdx ..... 175
Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset ..... 177
Table 9-45 – Bin string of the unary binarization (informative) ..... 181
Table 9-46 – Binarization for pred_type ..... 183
Table 9-47 – Binarization for rem_intra_luma_pred_mode ..... 184
Table 9-48 – Specifcation of cTRMax depending on cRiceParam ..... 184
Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values ..... 186
Table 9-50 – Specifcation of ctxIdxInc using left and above syntax elements ..... 190
Table 9-51 – Specifcation of lastCtx depending on binIdx ..... 191
Table 9-52 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx ..... 196
Table 9-53 – State transition table ..... 197
Table A-9-54 – Level limits [Ed. (TW) Kept for convenience] ..... 209

Deleted: 160
Deleted: 160
Deleted: 160
Deleted: 161
Deleted: 162
Deleted: 163
Deleted: 163
Deleted: 164
Deleted: 165
Deleted: 166
Deleted: 168
Deleted: 171
Deleted: 173
Deleted: 173
Deleted: 174
Deleted: 176
Deleted: 179
Deleted: 180
Deleted: 186
Deleted: 187
Deleted: 199

**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.

**Rec. / IS**

# High-Efficiency Video Coding

## 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 Collaborative Team on Video Coding (JCT-VC) in 2010 for development of a new Recommendation | International Standard.

### 0.2 Purpose

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

[Ed. TW: revise the following] 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.

[Ed. TW: revise the following] 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.

### 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.

[Ed. TW: revise the following] The coded representation specified in the syntax is designed to enable a high compression capability for a desired image or video quality. 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 coding or arithmetic coding.

### 0.7 How to read this Specification

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

**0.8 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 6.6 (Scanning order array initialisation process**

Deleted: 7

Input to this process is a block size blkSize.

Output of this process is the array ScanOrder[ scanIdx ][ pos ][ comp ]. The array index scanIdx equal to 0 specifies a zig-zag scan as specified in subclause **Error! Reference source not found.**, scanIdx equal to 1 specifies a horizontal scan and scanIdx equal to 2 specifies a vertical scan. The array index pos specifies the scan position ranging from 0 to ( blkSize * blkSize ) − 1. The array index comp equal to 0 specifies the horizontal component and the array index comp equal to 1 specifies the vertical component. The array ScanOrder is derived as follows.

```
        ScanOrder[0] = ZigZag

        i = 0
        y = 0
        while( y < blkSize ) {
            x = 0
            while( x < blkSize ) {
                ScanOrder[ 1 ][ i ][ 0 ] = x
                ScanOrder[ 1 ][ i ][ 1 ] = y
                x++
                i++
            }
            y++
        }

        i = 0
        x = 0
        while( x < blkSize ) {
            y = 0
            while( y < blkSize ) {
                ScanOrder[ 2 ][ i ][ 0 ] = x
                ScanOrder[ 2 ][ i ][ 1 ] = y
                y++
                i++
            }
            x++
        }
```

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 7.3.10, 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 7.4.10, 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.

Deleted: Syntax and semantics

Deleted: 7.4

Deleted: 8

Annex A specifies profiles 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. 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 High-Efficiency 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. [Ed. TW: revise the following]

- 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 10527:2007, *Colorimetric Observers*.

## 3 Definitions

[Ed. (TW) adpated definitions so far. Needs more work including turning them into 1 sentence each.]

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

**3.1** **access unit**: A set of *NAL units* that are consecutive in *decoding order* and contain exactly one *primary coded picture*. In addition to the *primary coded picture* one *auxiliary coded picture*, or other *NAL units* not containing *slices* of a *primary coded picture*. The decoding of an access unit always results in a *decoded picture*.

**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** **B slice**: A *slice* that may be decoded using *intra prediction* or *inter prediction* using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.

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

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

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

**3.9** **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*.

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

**3.11** **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*.

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

**3.13** **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*.

**3.14** **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.

**3.15** **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.

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

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

**3.18** **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.

**3.19** **clean decoding refresh (CDR) access unit**: An *access unit* in which the *primary coded picture* is a *CDR picture*.

**3.20** **clean decoding refresh (CDR) picture**: A *coded picture* containing only *slices* with *I slice types* that causes the *decoding process* to mark all *reference pictures* except the current CDR picture as "unused for reference" immediately before the decoding of the first picture following the current CDR with an *output order* greater than the current CDR picture. All *coded pictures* that follow a CDR picture in *output order* can be decoded without *inter prediction* from any *picture* that precedes the CDR picture in *output order*. All *coded pictures* with *output order* smaller than the current CDR are not affected by the deferred marking process.

**3.21** **coded picture**: A *coded representation* of a *picture*.

**3.22** **coded picture buffer (CPB)**: A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex 0.

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

**3.24** **coded slice NAL unit**: A *NAL unit* containing a *slice*.

**3.25** **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*.

**3.26** **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that make up a *picture* in monochrome format.

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

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

**3.29** **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*.

**3.30** **decoded picture buffer (DPB)**: A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex 0.

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

**3.32** **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*.

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

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

**3.35** **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.*

**3.36** **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*.

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

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

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

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

**3.41** **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.

**3.42** **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*.

**3.43** **I slice**: A *slice* that is decoded using *intra prediction* only.

**3.44** **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.

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

**3.46** **instantaneous decoding refresh (IDR) picture**: A *coded picture* for which the variable IdrPicFlag is equal to 1. An IDR picture causes the *decoding process* to mark all *reference pictures* as "unused for reference"

immediately after the decoding of the IDR picture. All *coded pictures* that follow an IDR picture in *decoding order* can be decoded without *inter prediction* from any *picture* that precedes the IDR picture in *decoding order*. The first *picture* of each *coded video sequence* in *decoding order* is an IDR picture.

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

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

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

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

**3.51** **intra slice**: See *I slice*.

**3.52** **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*.

**3.53** **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 *treeblock* layers.

**3.54** **leaf**: A terminating node of a tree that is a root node of a tree of depth 0.

**3.55** **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*.

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

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

**3.58** **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.

**3.59** **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.

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

**3.61** **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*.

**3.62** **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.

**3.63** **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*.

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

**3.65** **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*.

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

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

**3.68** **P slice**: A *slice* that may be decoded using *intra prediction* or *inter prediction* using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.

**3.69** **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*.

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

**3.71** **picture**: [Ed. (TW) define]

**3.72** **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.*

**3.73** **picture order count**: [Ed. (TW) re-check] A variable that is associated with each *coded picture* and has a value that is non-decreasing with increasing *picture* position in *output order* relative to the first output *picture* of the previous *IDR picture* in *decoding order* or relative to the first output *picture* of the previous *picture*, in *decoding order*, that contains a *memory management control operation* that marks all *reference pictures* as "unused for reference".

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

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

**3.76** **predictive slice**: See *P slice*.

**3.77** **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.

**3.78** **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 *treeblocks* of the *picture.* The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures.

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

**3.80** **quadtree**: A *tree* in which a parent node can be split into four child nodes. A child node may become parent node for another plit into four child nodes.

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

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

**3.83** **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.

**3.84** **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.

**3.85** **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.*

**3.86** **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*.

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

**3.88** **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*.

**3.89** **reference picture list**: A list of *reference pictures* that is used for *uni-prediction* of a *P* or *B slice.* For the *decoding process* of a *P slice,* there is one reference picture list. For the *decoding process* of a *B slice*, there are two reference picture lists (list 0 and list 1) and a reference picture lists combination. [Ed. (TW) the latter may not be true anymore]

**3.90** **reference picture list 0**: A *reference picture list* used for *inter prediction* of a *P* or *B slice*. All *inter prediction* used for *P slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *bi-prediction* for a *B slice*, with the other being *reference picture list*. [Ed. (TW) the latter may not be true anymore]

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

**3.92** **reference picture lists combination**: A reference picture list used for uni-prediction of a B slice. Reference picture lists combination is derived from the entries of the reference picture lists 0 and reference picture list 1.

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

**3.94** **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.

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

**3.96** **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 *picture*. Sample aspect ratio is expressed as *h*:*v*, where *h* is horizontal width and *v* is vertical height (in arbitrary units of spatial distance).

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

**3.98** **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.*

**3.99** **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.

**3.100** **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.

**3.101** **slice**: An integer number of *treeblocks* ordered consecutively in the *raster scan*. For the *primary coded picture*, the division of each *picture* into slices is a *partitioning*. The *treeblock addresses* are derived from the first *treeblock address* in a slice (as represented in the *slice header*).

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

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

**3.104** **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*.

**3.105** **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.

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

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

**3.108** **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*.

**3.109** **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.

**3.110** **tree**: A tree is a finite ste of nodes with a unique root node. A terminating node is called a *leaf*.

**3.111** **treeblock**: A NxN *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a NxN *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* into treeblocks is a *partitioning*.

**3.112** **treeblock address**: A treeblock address is the index of a *treeblock* in a *treeblock raster scan* of the *picture* starting with zero for the top-left *treeblock* in a *picture*.

**3.113** **treeblock location**: The two-dimensional coordinates of a *treeblock* in a *picture* denoted by ( x, y ). For the top left *treeblock* of the *picture* ( x, y ) is equal to ( 0, 0 ). x is incremented by 1 for each *treeblock* column from left to right. The value of y is incremented by 1 for each *treeblock* row from top to bottom.

**3.114** **treeblock partition**: A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *treeblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *treeblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.

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

**3.116** **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.

**3.117** **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.

**3.118** **VCL NAL unit**: A collective term for *coded slice NAL units*.

**3.119** **zig-zag scan**: A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest.

## 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 |
| CDR | Clean Decoding Refresh |
| 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 |
| MSB | Most Significant Bit |
| NAL | Network Abstraction Layer |
| RBSP | Raw Byte Sequence Payload |
| SEI | Supplemental Enhancement Information |
| SODB | String Of Data Bits |
| TB | Treeblock |

| | |
|---|---|
| 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, including matrix 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. |

## 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.
- ^ Bit-wise "exclusive 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:

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

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

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

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

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

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

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

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

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

Median( x, y, z ) = x + y + z − Min( x, Min( y, z ) ) − Max( x, Max( y, z ) ) (5-10)

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

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

Round( x ) = Sign( x ) * Floor( Abs( x ) + 0.5 ) (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 Order of operation precedence

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

– operations of a higher precedence are evaluated before any operation of a lower precedence,

– operations of the same precedence are evaluated sequentially from left to right.

Table 5-1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

Deleted: Table 5-1

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

**Table 5-1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)**

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

Deleted: 5-1

Deleted: 5-1

## 5.9 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 without 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 that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 0 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.7) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column

(horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[ x, y ] or as $s_{yx}$.

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 (counted from the most to the least significant bit) equal to 1.

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

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

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

### 5.10 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

### 5.11 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.

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

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

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

## 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 pictures that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of pictures in decoding order.

The source and decoded pictures are each comprised of one or more sample arrays:

- Luma (Y) only (monochrome).
- Luma and two chroma (YCbCr or YCgCo).
- Green, Blue and Red (GBR, also known as RGB).
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

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 variables SubWidthC, and SubHeightC are specified in Table 6-1, depending on the chroma format sampling structure, which is specified through chroma_format_idc and separate_colour_plane_flag. 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 and separate_colour_plane_flag**

| chroma_format_idc | separate_colour_plane_flag | Chroma Format | SubWidthC | SubHeightC |
|---|---|---|---|---|
| 0 | 0 | monochrome | - | - |
| 1 | 0 | 4:2:0 | 2 | 2 |
| 2 | 0 | 4:2:2 | 2 | 1 |
| 3 | 0 | 4:4:4 | 1 | 1 |
| 3 | 1 | 4:4:4 | - | - |

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, depending on the value of separate_colour_plane_flag, the following applies.

– If separate_colour_plane_flag is equal to 0, each of the two chroma arrays has the same height and width as the luma array.

– Otherwise (separate_colour_plane_flag is equal to 1), the three colour planes are separately processed as monochrome sampled pictures.

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 14, 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 pictures are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

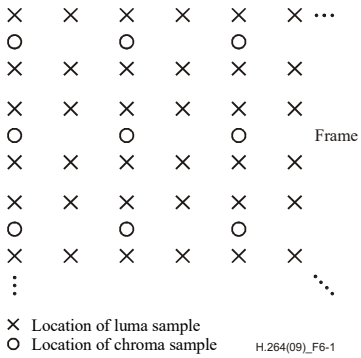



**Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture [Ed. Re-draw figure]**

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 picture are as shown in Figure 6-2.

Deleted: Table 6-1

Deleted: Table 6-1

Deleted: 6-1

Deleted: 6-1

Deleted: Figure 6-1

Deleted: 6-2

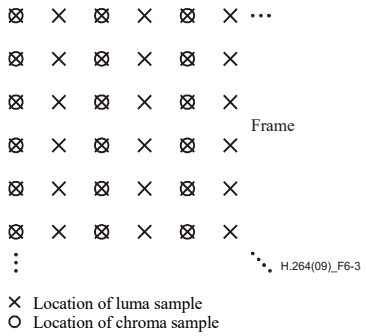



**Figure 6-2 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture [Ed. Re-draw figure]**

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

Deleted: 6-3

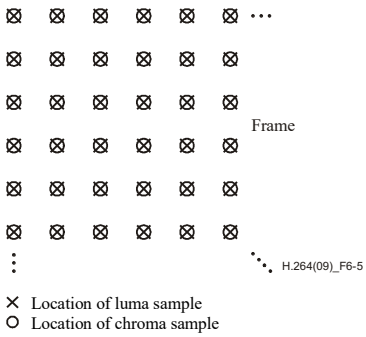



**Figure 6-3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture [Ed. Re-draw figure]**

The samples are processed in units of treeblocks. The luma array for each treeblock in samples in both width and height is

$$TbSize = 64 \quad (6\text{-}1)$$
$$Log2TbSize = 6 \quad (6\text{-}2)$$

The variables TbWidthC and TbHeightC, which specify the width and height, respectively, of the chroma arrays for each treeblock, are derived as follows.

– If chroma_format_idc is equal to 0 (monochrome) or separate_colour_plane_flag is equal to 1, TbWidthC and TbHeightC are both equal to 0.

– Otherwise, TbWidthC and TbHeightC are derived as

$$TbWidthC = TbSize / SubWidthC \quad (6\text{-}3)$$
$$TbHeightC = TbSize / SubHeightC \quad (6\text{-}4)$$

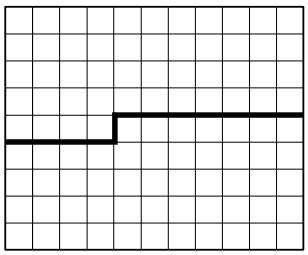
## 6.3 Spatial subdivision of pictures and slices

This subclause specifies how a picture is partitioned into slices and treeblocks. Pictures are divided into slices. A slice is a sequence of treeblocks.

Each treeblock is comprised of one Ntb x Ntb luma array and, when the chroma sampling format is not equal to 4:0:0 and separate_colour_plane_flag is equal to 0, two corresponding chroma sample arrays. When separate_colour_plane_flag is equal to 1, each treeblock is comprised of one Ntb x Ntb luma or chroma sample array. For example, a picture may be divided into two slices as shown in Figure 6-4.

When a picture is coded using three separate colour planes (separate_colour_plane_flag is equal to 1), a slice contains only treeblocks of one colour component being identified by the corresponding value of colour_plane_id, and each colour component array of a picture consists of slices having the same colour_plane_id value. Coded slices with different values of colour_plane_id within an access unit can be interleaved with each other under the constraint that for each value of colour_plane_id, the coded slice NAL units with that value colour_plane_id shall be in the order of increasing treeblock address for the first treeblock of each coded slice NAL unit.

NOTE – When separate_colour_plane_flag is equal to 0, each treeblock of a picture is contained in exactly one slice. When separate_colour_plane_flag is equal to 1, each treeblock of a colour component is contained in exactly one slice (i.e., information for each treeblock of a picture is present in exactly three slices and these three slices have different values of colour_plane_id).



H.264(09)_F6-7

**Figure 6-4 – A picture with 11 by 9 treeblocks that is partitioned into two slices**

Each treeblock is assigned a partition signalling to identify the block sizes for intra or inter prediction and for transform coding. The partitioning is a recursive quadtree partitioning. The root of the quadtree is associated with the treeblock. The quadtree is split until a leaf is reached, which is referred to as the coding node. The coding node is the root node of two trees, the prediction tree and the transform tree.

The prediction tree specifies the position and size of prediction blocks. The prediction tree and associated prediction data are referred to as prediction unit.

The transform tree specifies the position and size of transform blocks. The transform tree and associated transform data are referred to as transform unit.

The splitting information for luma and chroma is identical for the prediction tree and may or may not be identical for the transform tree.

The coding node and the associated prediction and transform units form together a coding unit.

[Ed.: (WJ) block = a rectangular 2D array (one component), unit = collective term for specifying information for both luma and chroma. Don't use ther term 'unit' by itself – always use the term 'unit' with prefix – coding unit, prediction unit or transform unit.]

## 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 treeblock scanning process

Input to this process is a treeblock address tbAddr.

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

The inverse treeblock scanning process is specified as follows.

$$x = \text{InverseRasterScan}( \text{tbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0 ) \quad (6\text{-}5)$$

Deleted: Figure 6-4

Deleted: [Ed. (TW) What is a "unit"?]¶

y = InverseRasterScan( tbAddr, 16, 16, PicWidthInSamples$_L$, 1 ) (6-6)

[Ed. (TW): The text found in JCVTVC-B205 that relates to the rest of this clause requires substantial editorial and formatting work. It cannot be imported as is.]

### 6.4.2 Derivation process for the availability of treeblock addresses

Input to this process is a treeblock address tbAddr.

Output of this process is the availability of the treeblock tbAddr.

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

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

- tbAddr < 0
- tbAddr > CurrTbAddr
- the treeblock with address tbAddr belongs to a different slice than the treeblock with address CurrTbAddr and the lightweight_slice_flag of the slice containing the treeblock with address CurrTbAddr is equal to 0.

[Ed. (WJ): do we need to separate the use of slice and lightweight slice? E.g. Is lightweight slice a kind of slice or sub-slice? We may need the concrete definitions]

## 6.5 Zig-zag scanning array initialisation process

Input to this process is a block size blkSize.

Output of this process is the array ZigZag[ pos ][ comp ]. The array index pos specify the scan position ranging from 0 to ( blkSize * blkSize ) − 1. The array index comp equal to 0 specifies the horizontal component and the array index comp equal to 1 specifies the vertical component. The array ZagZag is derived as follows.

```
ZigZag[ 0 ][ 0 ] = 0
ZigZag[ 0 ][ 1 ] = 0
i = 1
x = 1
y = 0
stopLoop = ( blkSize == 1 ) ? true : false
while( !stopLoop ) {
    while( x >= 0 ) {
        if (x >= 0 && x < blkSize && y >= 0 && y < blkSize ) {
            ZigZag[ i ][ 0 ] = x
            ZigZag[ i ][ 1 ] = y
            i++
        }
        x− −
        y++
    }
    x = 0
    while( y >= 0 ) {
        if (x >= 0 && x < blkSize && y >= 0 && y < blkSize ) {
            ZigZag[ i ][ 0 ] = x
            ZigZag[ i ][ 1 ] = y
            i++
        }
        x++
        y− −
    }
    y = 0
    if ( i >= blkSize * blkSize )
        stopLoop = true
}
```

### 6.6 Scanning order array initialisation process

Input to this process is a block size blkSize.

Output of this process is the array ScanOrder[ scanIdx ][ pos ][ comp ]. The array index scanIdx equal to 0 specifies a zig-zag scan as specified in subclause **Error! Reference source not found.**, scanIdx equal to 1 specifies a horizontal scan and scanIdx equal to 2 specifies a vertical scan. The array index pos specifies the scan position ranging from 0 to ( blkSize * blkSize ) − 1. The array index comp equal to 0 specifies the horizontal component and the array index comp equal to 1 specifies the vertical component. The array ScanOrder is derived as follows.

Deleted: 6.5

```
ScanOrder[0] = ZigZag

i = 0
y = 0
while( y < blkSize ) {
    x = 0
    while( x < blkSize ) {
        ScanOrder[ 1 ][ i ][ 0 ] = x
        ScanOrder[ 1 ][ i ][ 1 ] = y
        x++
        i++
    }
    y++
}

i = 0
x = 0
while( x < blkSize ) {
    y = 0
    while( y < blkSize ) {
        ScanOrder[ 2 ][ i ][ 0 ] = x
        ScanOrder[ 2 ][ i ][ 1 ] = y
        y++
        i++
    }
    x++
}
```

## 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 no more data in the RBSP, the return value of more_rbsp_data( ) is equal to FALSE.
- Otherwise, the RBSP data is searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp_stop_one_bit) of the rbsp_trailing_bits( ) syntax structure, the following applies.
  - If there is more data in an RBSP before the rbsp_trailing_bits( ) syntax structure, 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.

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

**Deleted:** 9.2

**Deleted:** 9.1

**Deleted:** 9.1

**Deleted:** 9.1

**Deleted:** 9.1

## 7.3 Syntax in tabular form

### 7.3.1 NAL unit syntax

| nal_unit( NumBytesInNALunit ) { | **Descriptor** |
|---|---|
| **forbidden_zero_bit** | f(1) |
| **nal_ref_idc** | u(2) |
| **nal_unit_type** | u(5) |
| NumBytesInRBSP = 0 | |
| nalUnitHeaderBytes = 1 | |
| if( nal_unit_type == 1 \|\| nal_unit_type == 4 \|\| nal_unit_type == 5 ) { | |
| **temporal_id** | u(3) |
| **output_flag** | u(1) |
| **reserved_one_4bits** | u(4) |
| nalUnitHeaderBytes += 1 | |
| } | |
| for( i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++ ) { | |
| if( i + 2 < NumBytesInNALunit && next_bits( 24 ) == 0x000003 ) { | |
| **rbsp_byte[** NumBytesInRBSP++ **]** | b(8) |
| **rbsp_byte[** NumBytesInRBSP++ **]** | b(8) |
| i += 2 | |
| **emulation_prevention_three_byte** /* equal to 0x03 */ | f(8) |
| } else | |
| **rbsp_byte[** NumBytesInRBSP++ **]** | 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( ) { | **Descriptor** |
|---|---|
| **profile_idc** | u(8) |
| **reserved_zero_8bits** /* equal to 0 */ | u(8) |
| **level_idc** | u(8) |
| **seq_parameter_set_id** | ue(v) |
| **max_temporal_layers_minus1** | u(3) |
| **pic_width_in_luma_samples** | u(16) |
| **pic_height_in_luma_samples** | u(16) |
| **bit_depth_luma_minus8** | ue(v) |
| **bit_depth_chroma_minus8** | ue(v) |
| **pcm_bit_depth_luma_minus1** | u(4) |
| **pcm_bit_depth_chroma_minus1** | u(4) |
| **log2_max_frame_num_minus4** | ue(v) |
| **pic_order_cnt_type** | ue(v) |
| if( pic_order_cnt_type == 0 ) | |
| **log2_max_pic_order_cnt_lsb_minus4** | ue(v) |
| else if( pic_order_cnt_type == 1 ) { | |
| **delta_pic_order_always_zero_flag** | u(1) |
| **offset_for_non_ref_pic** | se(v) |
| **num_ref_frames_in_pic_order_cnt_cycle** | ue(v) |
| for( i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++ ) | |
| **offset_for_ref_frame[ i ]** | se(v) |
| } | |
| **max_num_ref_frames** | ue(v) |
| **gaps_in_frame_num_value_allowed_flag** | u(1) |
| **log2_min_coding_block_size_minus3** | ue(v) |
| **log2_diff_max_min_coding_block_size** | ue(v) |
| **log2_min_transform_block_size_minus2** | ue(v) |
| **log2_diff_max_min_transform_block_size** | ue(v) |
| **log2_min_pcm_coding_block_size_minus3** | ue(v) |
| **max_transform_hierarchy_depth_inter** | ue(v) |
| **max_transform_hierarchy_depth_intra** | ue(v) |
| **chroma_pred_from_luma_enabled_flag** | u(1) |
| **loop_filter_across_slice_flag** | u(1) |
| **sample_adaptive_offset_enabled_flag** | u(1) |
| **adaptive_loop_filter_enabled_flag** | u(1) |
| **pcm_loop_filter_disable_flag** | u(1) |
| **cu_qp_delta_enabled_flag** | u(1) |
| **temporal_id_nesting_flag** | u(1) |
| rbsp_trailing_bits( ) | |
| } | |

#### 7.3.2.2 Picture parameter set RBSP syntax

| pic_parameter_set_rbsp( ) { | **Descriptor** |
|---|---|
| **pic_parameter_set_id** | ue(v) |
| **seq_parameter_set_id** | ue(v) |
| **entropy_coding_mode_flag** | u(1) |
| **num_temporal_layer_switching_point_flags** | ue(v) |
| for( i = 0; i < num_temporal_layer_switching_point_flags; i++ ) | |
| **temporal_layer_switching_point_flag**[ i ] | u(1) |
| **num_ref_idx_l0_default_active_minus1** | ue(v) |
| **num_ref_idx_l1_default_active_minus1** | ue(v) |
| **pic_init_qp_minus26** /* relative to 26 */ | se(v) |
| **constrained_intra_pred_flag** | u(1) |
| **slice_granularity** | u(2) |
| **shared_pps_info_enabled_flag** | u(1) |
| if( shared_pps_info_enabled_flag ) | |
| if( adaptive_loop_filter_enabled_flag ) | |
| alf_param( ) | |
| if( cu_qp_delta_enabled_flag ) | |
| **max_cu_qp_delta_depth** | u(4) |
| rbsp_trailing_bits( ) | |
| } | |

#### 7.3.2.3 Supplemental enhancement information RBSP syntax

| sei_rbsp( ) { | **Descriptor** |
|---|---|
| do | |
| sei_message( ) | |
| while( more_rbsp_data( ) ) | |
| rbsp_trailing_bits( ) | |
| } | |

#### 7.3.2.3.1 Supplemental enhancement information message syntax

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

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

#### 7.3.2.4 Access unit delimiter RBSP syntax

| access_unit_delimiter_rbsp( ) { | Descriptor |
|---|---|
| **primary_pic_type** | u(3) |
| rbsp_trailing_bits( ) | |
| } | |

#### 7.3.2.5 Filler data RBSP syntax

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

#### 7.3.2.6 Slice layer RBSP syntax

| slice_layer_rbsp( ) { | Descriptor |
|---|---|
| slice_header( ) | |
| slice_data( ) | |
| rbsp_slice_trailing_bits( ) | |
| } | |

#### 7.3.2.7 RBSP slice trailing bits syntax

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

#### 7.3.2.8 RBSP trailing bits syntax

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

### 7.3.3 Slice header syntax

| slice_header( ) { | Descriptor |
|---|---|
| **lightweight_slice_flag** | u(1) |
| if( !lightweight_slice_flag ) { | |
| **slice_type** | ue(v) |
| **pic_parameter_set_id** | ue(v) |
| **frame_num** | u(v) |
| if( IdrPicFlag ) | |
| **idr_pic_id** | ue(v) |
| if( pic_order_cnt_type == 0 ) | |
| **pic_order_cnt_lsb /*** | u(v) |
| if( slice_type == P \|\| slice_type == B ) { | |
| **num_ref_idx_active_override_flag** | u(1) |
| if( num_ref_idx_active_override_flag ) { | |
| **num_ref_idx_l0_active_minus1** | ue(v) |
| if( slice_type == B ) | |
| **num_ref_idx_l1_active_minus1** | ue(v) |
| } | |
| } | |
| ref_pic_list_modification( ) | |
| ref_pic_list_combination( ) | |
| if( nal_ref_idc != 0 ) | |
| dec_ref_pic_marking( ) | |
| } | |
| if( entropy_coding_mode_flag && slice_type != I) | |
| **cabac_init_idc** | ue(v) |
| **first_slice_in_pic_flag** | u(1) |
| if( first_slice_in_pic_flag == 0 ) | |
| **slice_address** | u(v) |
| if( !lightweight_slice_flag ) { | |
| **slice_qp_delta** | se(v) |
| if( sample_adaptive_offset_enabled_flag ) | |
| sao_param() | |
| if( deblocking_filter_control_present_flag ) { | |
| **disable_deblocking_filter_idc** | |
| if( disable_deblocking_filter_idc != 1 ) { | |
| **slice_alpha_c0_offset_div2** | |
| **slice_beta_offset_div2** | |
| } | |

Deleted: first_tb_in_slice ... [1]

Deleted: entropy_slice_flag

Deleted: entropy

Deleted:

Deleted:

Formatted: Font: Bold

Formatted: Font: Bold

| | |
|---|---|
| } | |
| if( slice_type = = B ) | |
| **collocated_from_l0_flag** | u(1) |
| if( adaptive_loop_filter_enabled_flag ) { | |
| if( !shared_pps_info_enabled_flag ) | |
| alf_param( ) | |
| alf_cu_control_param( ) | |
| } | |
| } | |
| } | |

Deleted: else

Deleted: →→if( entropy_coding_mode_flag && slice_type [... [2]]

#### 7.3.3.1 Reference picture list modification syntax

| ref_pic_list_modification( ) { | **Descriptor** |
|---|---|
| if( slice_type % 5 != 2 && slice_type % 5 != 4 ) { | |
| **ref_pic_list_modification_flag_l0** | u(1) |
| if( ref_pic_list_modification_flag_l0 ) | |
| do { | |
| **modification_of_pic_nums_idc** | ue(v) |
| if( modification_of_pic_nums_idc = = 0 \|\| modification_of_pic_nums_idc = = 1 ) | |
| **abs_diff_pic_num_minus1** | ue(v) |
| else if( modification_of_pic_nums_idc = = 2 ) | |
| **long_term_pic_num** | ue(v) |
| } while( modification_of_pic_nums_idc != 3 ) | |
| } | |
| if( slice_type % 5 = = 1 ) { | |
| **ref_pic_list_modification_flag_l1** | u(1) |
| if( ref_pic_list_modification_flag_l1 ) | |
| do { | |
| **modification_of_pic_nums_idc** | ue(v) |
| if( modification_of_pic_nums_idc = = 0 \|\| modification_of_pic_nums_idc = = 1 ) | |
| **abs_diff_pic_num_minus1** | ue(v) |
| else if( modification_of_pic_nums_idc = = 2 ) | |
| **long_term_pic_num** | ue(v) |
| } while( modification_of_pic_nums_idc != 3 ) | |
| } | |
| } | |

#### 7.3.3.2 Reference picture lists combination syntax

| ref_pic_list_combination( ) { | C | Descriptor |
|---|---|---|
| if( slice_type % 5 = = 1 ) { // b slice | | |
| **ref_pic_list_combination_flag** | 2 | u(1) |
| if( ref_pic_list_combination_flag ) { | | |
| **num_ref_idx lc_active_minus1** | 2 | ue(v) |
| **ref_pic_list_modification_flag_lc** | 2 | u(1) |
| if( ref_pic_list_modification_flag_lc ) | | |
| for ( i =0; i <= num_ref_idx_lc_active_minus1; i++ ) { | | |
| **pic_from_list_0_flag** | 2 | u(1) |
| **ref_idx_list_curr** | 2 | ue(v) |
| } | | |
| } | | |
| } | | |
| } | | |

#### 7.3.3.3 Decoded reference picture marking syntax

| dec_ref_pic_marking( ) { | Descriptor |
|---|---|
| if( IdrPicFlag ) { | |
| **no_output_of_prior_pics_flag** | u(1) |
| **long_term_reference_flag** | u(1) |
| } else { | |
| **adaptive_ref_pic_marking_mode_flag** | u(1) |
| if( adaptive_ref_pic_marking_mode_flag ) | |
| do { | |
| **memory_management_control_operation** | ue(v) |
| if( memory_management_control_operation = = 1 \|\| memory_management_control_operation = = 3 ) | |
| **difference_of_pic_nums_minus1** | ue(v) |
| if(memory_management_control_operation = = 2 ) | |
| **long_term_pic_num** | ue(v) |
| if( memory_management_control_operation = = 3 \|\| memory_management_control_operation = = 6 ) | |
| **long_term_frame_idx** | ue(v) |
| if( memory_management_control_operation = = 4 ) | |
| **max_long_term_frame_idx_plus1** | ue(v) |
| } while( memory_management_control_operation != 0 ) | |
| } | |
| } | |

#### 7.3.3.4 Sample adaptive offset parameter syntax

| sao_param( ) { | C | Descriptor |
|---|---|---|
| **sample_adaptive_offset_flag** | 2 | u(1) |
| if ( sample_adaptive_offset_flag ) { | | |
| sao_split_param( 0, 0, 0 ) | | |
| sao_offset_param( 0, 0, 0 ) | | |
| } | | |
| } | | |

| sao_split_param( x0, y0, saoDepth ) { | C | Descriptor |
|---|---|---|
| if( saoDepth < SaoMaxDepth ) | | |
| **sao_split_flag**[ saoDepth ][ x0 ][ y0 ] | 2 | u(1) \| ae(v) |
| if( sao_split_flag[ saoDepth ][ x0 ][ y0 ] ) { | | |
| sao_split_param( x0 + 0, y0 + 0, saoDepth + 1 ) | | |
| sao_split_param( x0 + 1, y0 + 0, saoDepth + 1 ) | | |
| sao_split_param( x0 + 0, y0 + 1, saoDepth + 1 ) | | |
| sao_split_param( x0 + 1, y0 + 1, saoDepth + 1 ) | | |
| } | | |
| } | | |

[Ed.: (WJ) is it better to assign separate section?]

| sao_offset_param( x0, y0, saoDepth ) { | C | Descriptor |
|---|---|---|
| if( sao_split_flag[ saoDepth ][ x0 ][ y0 ] ) { | | |
| sao_offset_param( x0 + 0, y0 + 0, saoDepth + 1 ) | | |
| sao_offset_param( x0 + 1, y0 + 0, saoDepth + 1 ) | | |
| sao_offset_param( x0 + 0, y0 + 1, saoDepth + 1 ) | | |
| sao_offset_param( x0 + 1, y0 + 1, saoDepth + 1 ) | | |
| } else { | | |
| **sao_type_idx**[ saoDepth ][ x0 ][ y0 ] | 2 | ue(v) \| ae(v) |
| if( sao_type_idx[ saoDepth ][ x0 ][ y0 ] != 0 ) | | |
| for( i = 0; i < NumSaoClass; i++ ) | | |
| **sao_offset**[ saoDepth ][ x0 ][ y0 ][ i ] | 2 | se(v) \| ae(v) |
| } | | |
| } | | |

[Ed.: (WJ) is it better to assign separate section?]

### 7.3.3.5 Adaptive loop filter parameter syntax

| alf_param() { | C | Descriptor |
|---|---|---|
| **adaptive_loop_filter_flag** | 2 | u(1) |
| if ( adaptive_loop_filter_flag ) { | | |
| **alf_region_adaptation_flag** | 2 | u(1) |
| **alf_length_luma_minus_5_div2** | 2 | ue(v) |
| **alf_no_filters_minus1** | 2 | ue(v) |
| if (alf_no_filters_minus1 == 1) | | |
| **alf_start_second_filter** | 2 | ue(v) |
| else if (alf_no_filters_minus1 > 1) { | | |
| for (i=1; i< 16; i++) | | |
| **alf_filter_pattern[i]** | 2 | u(1) |
| } | | |
| if (AlfNumFilters > 1) | | |
| **alf_pred_method** | 2 | u(1) |
| **alf_min_kstart_minus1** | 2 | ue(v) |
| for (i=0; i < AlfMaxDepth; i++) | | |
| **alf_golomb_index_bit[i]** | 2 | u(1) |
| for (i=0; i< AlfNumFilters; i++) | | |
| for (j=0; j< AlfCodedLengthLuma; j++) | | |
| **alf_coeff_luma**[i][j] | | ge(v) |
| **alf_chroma_idc** | 2 | ue(v) |
| if ( alf_chroma_idc ) { | | |
| **alf_length_chroma_minus_5_div2** | 2 | ue(v) |
| for( i = 0; i< AlfCodedLengthChroma; i++ ) | | |
| **alf_coeff_chroma**[i] | | se(v) |
| } | | |
| } | | |
| } | | |

Deleted: 0

### 7.3.3.6 Adaptive loop filter coding unit control parameter syntax

| alf_cu_control_param() { | C | Descriptor |
|---|---|---|
| if( adaptive_loop_filter_flag ) { | | |
| **alf_cu_control_flag** | 2 | u(1) |
| if( alf_cu_control_flag ) { | | |
| **alf_cu_control_max_depth** | 2 | ue(v) |
| **alf_length_cu_control_info** | 2 | se(v) |
| for( i = 0; i < NumAlfCuFlag; i++ ) | | |
| **alf_cu_flag**[ i ] | 2 | u(1) \| ae(v) |
| } | | |
| } | | |
| } | | |

Deleted: →→→if( alf_cu_control_max_depth ) ... [3]

Deleted: →

Deleted: u

### 7.3.4 Slice data syntax

| slice_data( ) { | Descriptor |
|---|---|
| CurrTbAddr = LCUAddress | |
| moreDataFlag = 1 | |
| if( adaptive_loop_filter_flag && alf_cu_control_flag ) | |
| AlfCuFlagIdx = -1 | |
| do { | |
| xCU = HorLumaLocation( CurrTbAddr ) | |
| yCU = VerLumaLocation( CurrTbAddr ) | |
| moreDataFlag = coding_tree( xCU, yCU, Log2TbSize ) | |
| CurrTbAddr = NextTbAddress( CurrTbAddr ) | |
| } while( moreDataFlag ) | |
| } | |

**Deleted:** first_tb_in_slice

**Deleted:** →→if( !entropy_coding_mode_flag ) ... [4]

### 7.3.5 Coding tree syntax

| coding_tree( x0, y0, log2CUSize ) { | Descriptor |
|---|---|
| if( x0 + ( 1 << log2CUSize ) <= PicWidthInSamples$_L$ && y0 + ( 1 << log2CUSize ) <= PicHeightInSamples$_L$ && cuAddress( x0, y0 ) >= SliceAddress ) { | |
| if( !entropy_coding_mode_flag && slice_type != I ) | |
| **cu_split_pred_part_mode**[ x0 ][ y0 ] | ce(v) |
| else if( log2CUSize > Log2MinCUSize ) | |
| **split_coding_unit_flag[** x0 **][** y0 **]** | u(1) \| ae(v) |
| } | |
| if( adaptive_loop_filter_flag && alf_cu_control_flag ) { | |
| cuDepth = Log2MaxCUSize – log2CUSize | |
| if( cuDepth <= alf_cu_control_max_depth ) | |
| if( cuDepth == alf_cu_control_max_depth \|\| | |
| split_coding_unit_flag[ x0 ][ y0 ] == 0 ) | |
| AlfCuFlagIdx++ | |
| } | |
| if( split_coding_unit_flag[ x0 ][ y0 ] ) { | |
| if( cu_qp_depta_enabled_flag && log2CUSize == log2MinCUDQPSize ) | |
| IsCuQpDeltaCoded = 0 | |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) | |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) | |
| if( cuAddress( x1, y0 ) > SliceAddress ) | |
| moreDataFlag = coding_tree( x0, y0, log2CUSize – 1 ) | |
| if( cuAddress( x0, y1 ) > SliceAddress && moreDataFlag && x1 < PicWidthInSamples$_L$ ) | |
| moreDataFlag = coding_tree( x1, y0, log2CUSize − 1 ) | |
| if( cuAddress( x1, y1 ) > SliceAddress && moreDataFlag && y1 < PicHeightInSamples$_L$ ) | |
| moreDataFlag = coding_tree( x0, y1, log2CUSize − 1 ) | |
| if( moreDataFlag && x1 < PicWidthInSamples$_L$ && y1 < PicHeightInSamples$_L$ ) | |
| moreDataFlag = coding_tree( x1, y1, log2CUSize − 1 ) | |
| } else { | |
| if(adaptive_loop_filter_flag && alf_cu_control_flag ) | |
| AlfCuFlag[ x0 ][ y0 ] = alf_cu_flag[ AlfCuFlagIdx ] | |
| coding_unit( x0, y0, log2CUSize ) | |
| if( !entropy_coding_mode_flag ) | |
| moreDataFlag = more_rbsp_data( ) | |
| else { | |
| if( granularity_block_boundary( x0, y0, log2CUSize ) ) { | |
| **end_of_slice_flag** | ae(v) |
| moreDataFlag = !end_of_slice_flag | |
| } else | |
| moreDataFlag = 1 | |
| } | |
| } | |
| return moreDataFlag | |
| } | |

Formatted: Highlight

Formatted: Highlight

Formatted: Highlight

Formatted: Highlight

Formatted: (Asian) Korean, (Other) English (US)

Formatted: Font: Bold

[Ed. (WJ): cuAddress( x, y ) returns the address of the smallest coding unit at (x, y), expressed in global decoding order smallest coding unit coordinates. granularity_block_boundary( x0, y0, log2CUSize ) returns true when the coding unit specified by ( x0, y0, log2CUSize ) is the last coding unit in the slice in global decoding order. These functions are not defined yet. They will be added later.]

### 7.3.6 Coding unit syntax

| coding_unit( x0, y0, log2CUSize ) { | **Descriptor** |
|---|---|
| if( entropy_coding_mode_flag && slice_type != I ) | |
| **skip_flag[** x0 **][** y0 **]** | u(1) \| ae(v) |
| if( skip_flag[ x0 ][ y0 ] ) | |
| prediction_unit( x0, y0, log2CUSize, log2CUSize, 0 , 0 ) | |
| else { | |
| if( !entropy_coding_mode_flag ) { | |
| if( slice_type == I && log2CUSize == Log2MinCUSize ) | |
| **intra_part_mode** | u(1) |
| } else if( slice_type != I \|\| log2CUSize == Log2MinCUSize ) | |
| **pred_type** | u(v) \| ae(v) |
| x1 = x0 + ( ( 1 << log2CUSize ) >> 1 ) | |
| y1 = y0 + ( ( 1 << log2CUSize ) >> 1 ) | |
| if( PartMode == PART_2Nx2N ) { | |
| prediction_unit( x0, y0, log2CUSize, log2CUSize, 0 , 0 ) | |
| } else if( PartMode == PART_2NxN ) { | |
| prediction_unit( x0, y0, log2CUSize, log2CUSize – 1, 0 , log2CUSize > Log2MinCUSize ) | |
| prediction_unit( x0, y1, log2CUSize, log2CUSize – 1, 1 , 0 ) | |
| } else if( PartMode == PART_Nx2N ) { | |
| prediction_unit( x0, y0, log2CUSize - 1, log2CUSize, 0 , log2CUSize > Log2MinCUSize ) | |
| prediction_unit( x1, y0, log2CUSize - 1, log2CUSize, 1 , 0 ) | |
| } else { /* PART_NxN */ | |
| prediction_unit( x0, y0, log2CUSize – 1, log2CUSize – 1, 0 , 0 ) | |
| prediction_unit( x1, y0, log2CUSize – 1, log2CUSize – 1, 1 , 0 ) | |
| prediction_unit( x0, y1, log2CUSize – 1, log2CUSize – 1, 2 , 0 ) | |
| prediction_unit( x1, y1, log2CUSize – 1, log2CUSize – 1, 3 , 0 ) | |
| } | |
| } | |
| if( !pcm_flag ) { | |
| transform_tree( x0, y0, log2CUSize, 0, 0 ) | |
| transform_coeff( x0, y0, log2CUSize, 0, 0 ) | |
| transform_coeff( x0, y0, log2CUSize, 0, 1 ) | |
| transform_coeff( x0, y0, log2CUSize, 0, 2 ) | |
| } | |
| } | |

### 7.3.7 Prediction unit syntax

| prediction_unit( x0, y0, log2PUWidth, log2PUHeight, PartIdx , InferredMergeFlag ) { | **Descriptor** |
|---|---|
| if( skip_flag[ x0 ][ y0 ] ) { | |
| if( NumMergeCand > 1 ) | |
| **merge_idx[** x0 **][** y0 **]** | ue(v) \| ae(v) |
| } else if( PredMode == MODE_INTRA ) { | |
| if( PartMode == PART_2Nx2N && log2PUWidth >= Log2IPCMCUSize ) | |
| **pcm_flag** | u(1) \| ae(v) |
| if( pcm_flag ) { | |
| while( !byte_aligned( ) ) | |
| **pcm_alignment_zero_bit** | u(v) |
| for( i = 0; i < 1 << ( log2CUSize << 1 ); i++ ) | |
| **pcm_sample_luma[** i **]** | u(v) |
| for( i = 0; i < ( 1 << ( log2CUSize << 1 ) ) >> 1; i++ ) | |
| **pcm_sample_chroma[** i **]** | u(v) |
| } else { | |
| **prev_intra_luma_pred_flag[** x0 **][** y0 **]** | u(1) \| ae(v) |
| if( prev_intra_luma_pred_flag[ x0 ][ y0 ] ) | |
| if( NumMPMCand > 1 ) | |
| **mpm_idx[** x0 **][** y0 **]** | u(1) \| ae(v) |
| else | |
| **rem_intra_luma_pred_mode[** x0 **][** y0 **]** | ce(v) \| ae(v) |
| if( IntraPredMode[ x0 ][ y0 ] == 2 ) | |
| **planar_flag_luma**[ x0 ][ y0 ] | u(1) \| ae(v) |
| **intra_chroma_pred_mode**[ x0 ][ y0 ] | ue(v) \| ae(v) |
| SignaledAsChromaDC = ( chroma_pred_from_luma_enabled_flag ? intra_chroma_pred_mode[ x0 ][ y0 ] == 3 : intra_chroma_pred_mode[ x0 ][ y0 ] == 2 ) | |
| if( IntraPredMode[ x0 ][ y0 ] != 2 && IntraPredMode[ x0 ][ y0 ]!=34 && SignaledAsChromaDC ) | |
| **planar_flag_chroma**[ x0 ][ y0 ] | u(1) \| ae(v) |
| } | |
| } else { /* MODE_INTER */ | |
| if( !InferredMergeFlag ) | |
| if( entropy_coding_mode_flag \|\| PartMode != PART_2Nx2N ) | |
| **merge_flag[** x0 **][** y0 **]** | u(1) \| ae(v) |
| if( merge_flag[ x0 ][ y0 ] && NumMergeCand > 1 ) { | |
| **merge_idx[** x0 **][** y0 **]** | ue(v) \| ae(v) |
| } else { | |
| if( slice_type == B ) { | |
| if(!entropy_coding_mode_flag && use_combined_inter_pred_ref( x0, y0 ) ) { | |
| **combined_inter_pred_ref_idx** | ue(v) |
| if( combined_inter_pred_ref_idx == MaxPredRef ) | |
| **inter_pred_flag[** x0 **][** y0 **]** | ue(v) |
| } else | |
| **inter_pred_flag[** x0 **][** y0 **]** | ue(v) \| ae(v) |

Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: Highlight
Formatted: Font: Bold
Formatted: English (US)
Formatted: English (US)
Formatted: Highlight
Formatted: Highlight
Formatted: English (US)
Formatted: Font: Bold
Formatted: English (US)
Formatted: English (US)
Formatted: Font: Bold
Formatted: Font: Bold
Formatted: English (US)

| | |
|---|---|
| } | |
| if( inter_pred_flag[ x0 ][ y0 ] == Pred_LC ) { | |
| if( num_ref_idx_lc_active_minus1 > 0 ) { | |
| if( !entropy_coding_mode_flag && use_combined_inter_pred_ref( x0, y0 ) ) { | |
| if( combined_inter_pred_ref_idx == MaxPredRef ) | |
| **ref_idx_lc_minus4**[ x0 ][ y0 ] | ue(v) |
| } else | |
| **ref_idx_lc[** x0 **][** y0 **]** | ae(v) |
| } | |
| **mvd_lc[** x0 **][** y0 **][** 0 **]** | se(v) \| ae(v) |
| **mvd_lc[** x0 **][** y0 **][** 1 **]** | se(v) \| ae(v) |
| if( NumMVPCand( LcToLx ) > 1 ) | |
| **mvp_idx_lc[ x0 ][ y0 ]** | ue(v) \| ae(v) |
| } | |
| else { /* Pred_L0 or Pred_BI */ | |
| if( num_ref_idx_l0_active_minus1 > 0 ) { | |
| if( !entropy_coding_mode_flag && use_combined_inter_pred_ref( x0, y0 ) ) { | |
| if( combined_inter_pred_ref_idx == MaxPredRef ) | |
| **ref_idx_l0_minusX**[ x0 ][ y0 ] | ue(v) |
| } else | |
| **ref_idx_l0_minusX**[ x0 ][ y0 ] | ue(v) \| ae(v) |
| } | |
| **mvd_l0[** x0 **][** y0 **][** 0 **]** | se(v) \| ae(v) |
| **mvd_l0[** x0 **][** y0 **][** 1 **]** | se(v) \| ae(v) |
| if( NumMVPCand( L0 ) > 1 ) | |
| **mvp_idx_l0[ x0 ][ y0 ]** | ue(v) \| ae(v) |
| } | |
| if( inter_pred_flag[ x0 ][ y0 ] == Pred_BI ) { | |
| if( num_ref_idx_l1_active_minus1 > 0 ) { | |
| if( !entropy_coding_mode_flag && use_combined_inter_pred_ref( x0, y0 ) ) { | |
| if( combined_inter_pred_ref_idx == MaxPredRef ) | |
| **ref_idx_l1_minusX**[ x0 ][ y0 ] | ue(v) |
| } else | |
| **ref_idx_l1[** x0 **][** y0 **]** | ue(v) \| ae(v) |
| } | |
| **mvd_l1[** x0 **][** y0 **][** 0 **]** | se(v) \| ae(v) |
| **mvd_l1[** x0 **][** y0 **][** 1 **]** | se(v) \| ae(v) |
| if( NumMVPCand( L1 ) > 1 ) | |
| **mvp_idx_l1[ x0 ][ y0 ]** | ue(v) \| ae(v) |
| } | |
| } | |
| } | |
| } | |

**Formatted:** Font: Bold

**Formatted:** English (US)

**Deleted:** ue(v) |

**Deleted:** →→→→→**ref_idx_l0[ x0 ][ y0 ]** ... [5]

**Formatted:** Font: Bold

**Formatted:** Font: Bold

**Formatted:** Font: Bold

**Formatted:** English (US)

[Ed: (BB) NumMergeCand needs to be defined. Decoding processes of neighboring blocks have to be finished to derive NumMergeCand. Same applies for NumMVPCand (TK) and NumMPMCand ]

[(TK) Note that the CAVLC syntax may be a little different for the prev_intra_luma_pred_flag[ x0 ][ y0 ] is incorporated into the coding of rem_intra_luma_pred_mode[ x0 ][ y0 ]. It should be decided where the syntax tables for CAVLC (LCEC) should diverge from that of the CABAC. ]

[(TK) Note that in CAVLC decoding process, there is also a set of ranking tables and a set of code tables. The selection of these tables are conditional on the NumMPMCand. These tables are available in JCTVC-E131 and should be added to the section for the description of the decoding process for the CAVLC (LCEC) if and when that section becomes available]

### 7.3.8 Transform tree syntax

| transform_tree( x0, y0, log2TrafoSize, trafoDepth, blkIdx ) { | **Descriptor** |
|---|---|
| if( entropy_coding_mode_flag && trafoDepth == 0 && IntraSplitFlag == 0) { | |
| if( PredMode != MODE_INTRA ) | |
| **no_residual_data_flag** | u(1) \| ae(v) |
| residualDataPresentFlag = !no_residual_data_flag | |
| } else { | |
| residualDataPresentFlag = TRUE | |
| } | |
| if( residualDataPresentFlag) { | |
| intraSplitFlag = ( IntraSplitFlag && trafoDepth == 0 ? 1 : 0 ) | |
| maxDepth = ( PredMode == MODE_INTRA ? max_transform_hierarchy_depth_intra + IntraSplitFlag : max_transform_hierarchy_depth_inter ) | |
| if( !entropy_coding_mode_flag ) { | |
| xB = x0 − ( x0 & ( ( 1 << log2TrafoSize ) − 1 ) ) | |
| yB = y0 − ( y0 & ( ( 1 << log2TrafoSize ) − 1 ) ) | |
| if( trafoDepth != 0 && !cbf_luma[ xB ][ yB ][ trafoDepth − 1 ] ) { | |
| if( cbf_cb[ xB ][ yB ][ trafoDepth − 1 ] && log2TrafoSize > Log2MinTrafoSize ) | |
| **cbf_cb**[ x0 ][ y0 ] [ trafoDepth ] | u(1) |
| if( cbf_cr[ xB ][ yB ][ trafoDepth − 1 ] && log2TrafoSize > Log2MinTrafoSize ) | |
| **cbf_cr**[ x0 ][ y0 ] [ trafoDepth ] | u(1) |
| if(log2TrafoSize <= Log2MaxTrafoSize && !intraSplitFlag && log2TrafoSize > Log2MinTrafoSize && trafoDepth < maxDepth ) | |
| **split_transform_flag**[ x0 ][ y0 ] [ trafoDepth ] | u(1) |
| } else | |
| **cbp_and_split_transform** | vlc(n,v) |
| } | |
| if( log2TrafoSize <= Log2MaxTrafoSize && log2TrafoSize > Log2MinTrafoSize && trafoDepth < maxDepth && !intraSplitFlag && entropy_coding_mode_flag ) | |
| **split_transform_flag**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| if( PredMode != MODE_INTRA && log2TrafoSize <= Log2MaxTrafoSize && entropy_coding_mode_flag ) { | |
| firstChromaCbf = ( log2TrafoSize == Log2MaxTrafoSize \|\| trafoDepth == 0 ? 1 : 0 ) | |
| if( firstChromaCbf \|\| log2TrafoSize > Log2MinTrafoSize ) { | |
| xBase = x0 − ( x0 & ( ( 1 << log2TrafoSize ) − 1 ) ) | |
| yBase = y0 − ( y0 & ( ( 1 << log2TrafoSize ) − 1 ) ) | |
| if( firstChromaCbf \|\| cbf_cb[ xBase ][ yBase ][ trafoDepth − 1 ] ) | |
| **cbf_cb**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| if( firstChromaCbf \|\| cbf_cr[ xBase ][ yBase ][ trafoDepth − 1 ] ) | |
| **cbf_cr**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| } | |
| } | |
| if( split_transform_flag[ x0 ][ y0 ][ trafoDepth ] ) { | |
| x1 = x0 + ( ( 1 << log2TrafoSize ) >> 1 ) | |
| y1 = y0 + ( ( 1 << log2TrafoSize ) >> 1 ) | |

| | |
|---|---|
| transform_tree( x0, y0, log2TrafoSize − 1, trafoDepth + 1, 0 ) | |
| transform_tree( x1, y0, log2TrafoSize − 1, trafoDepth + 1, 1 ) | |
| transform_tree( x0, y1, log2TrafoSize − 1, trafoDepth + 1, 2 ) | |
| transform_tree( x1, y1, log2TrafoSize − 1, trafoDepth + 1, 3 ) | |
| } else if( entropy_coding_mode_flag ){ | |
| if( PredMode == MODE_INTRA \|\| trafoDepth != 0 \|\|<br>cbf_cb[ x0 ][ y0 ][ trafoDepth ] \|\|<br>cbf_cr[ x0 ][ y0 ][ trafoDepth ] ) | |
| **cbf_luma**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| if( PredMode == MODE_INTRA ) | |
| if( log2TrafoSize > Log2MinTrafoSize ) { | |
| **cbf_cb**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| **cbf_cr**[ x0 ][ y0 ][ trafoDepth ] | u(1) \| ae(v) |
| } else if( blkIdx == 0 ) { | |
| **cbf_cb**[ x0 ][ y0 ][ trafoDepth − 1 ] | u(1) \| ae(v) |
| **cbf_cr**[ x0 ][ y0 ][ trafoDepth − 1 ] | u(1) \| ae(v) |
| } | |
| } | |
| } | |
| } | |

### 7.3.9 Transform coefficient syntax

| transform_coeff( x0, y0, log2TrafoSize, trafoDepth, cIdx ) { | **Descriptor** |
|---|---|
| if( ( ( cIdx = = 0 && cbf_luma[ x0 ][ y0 ][ trafoDepth ] ) \|\| ( cIdx = = 1 && cbf_cb[ x0 ][ y0 ][ trafoDepth ] ) \|\| ( cIdx = = 2 && cbf_cr[ x0 ][ y0 ][ trafoDepth ] ) ) ) { | |
| if( cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded ) { | |
| **cu_qp_delta** | se(v) \| ae(v) |
| IsCuQpDeltaCoded = 1 | |
| } | |
| if( split_transform_flag[ x0 ][ y0 ][ trafoDepth ] ) { | |
| x1 = x0 + ( ( 1 << log2TrafoSize ) >> 1 ) | |
| y1 = y0 + ( ( 1 << log2TrafoSize ) >> 1 ) | |
| transform_coeff( x0, y0, log2TrafoSize − 1, trafoDepth + 1, cIdx ) | |
| transform_coeff( x1, y0, log2TrafoSize − 1, trafoDepth + 1, cIdx ) | |
| transform_coeff( x0, y1, log2TrafoSize − 1, trafoDepth + 1, cIdx ) | |
| transform_coeff( x1, y1, log2TrafoSize − 1, trafoDepth + 1, cIdx ) | |
| } else { | |
| if ( PredMode = = MODE_INTRA ) { | |
| if (cIdx = = 0) | |
| scanIdx = ScanType[ log2TrafoSize -2][ IntraPredMode ] | |
| else | |
| scanIdx = ScanTypeC[ log2TrafoSize -2][ IntraPredModeC ] | |
| } else { | |
| scanIdx = 0 | |
| } | |
| if ( entropy_coding_mode_flag ) | |
| residual_coding_cabac( x0, y0, log2TrafoSize, trafoDepth, scanIdx, cIdx ) | |
| else if ( !entropy_coding_mode_flag ) | |
| residual_coding_cavlc( x0, y0, log2TrafoSize, trafoDepth, scanIdx, cIdx ) | |
| } | |
| } | |
| } | |

### 7.3.10 Residual coding CABAC syntax

| residual_coding_cabac( x0, y0, log2TrafoSize, trafoDepth, scanIdx, cIdx ) { | **Descriptor** |
|---|---|
| **last_significant_coeff_x** | ae(v) |
| **last_significant_coeff_y** | ae(v) |
| n = 0 | |
| xC = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ n ][ 0 ] | |
| yC = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ n ][ 1 ] | |
| while( ( xC != last_significant_coeff_x ) \|\| ( yC != last_significant_coeff_y ) ) { | |
| **significant_coeff_flag[** xC **][** yC **]** | ae(v) |
| n++ | |
| xC = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ n ][ 0 ] | |
| yC = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ n ][ 1 ] | |
| } | |
| numSubsets = max( 1, ( 1 << ( log2TrafoSize << 1 ) ) >> ( cIdx > 0 ? 6 : 4 )) | |
| for( i = 0; i < numSubsets; i++ ) { | |
| offset = i << 4 | |
| xS = ScanOrder[ log2TrafoSize − 2 ][ 0 ][ i ][ 0 ] << 2 | |
| yS = ScanOrder[ log2TrafoSize − 2 ][ 0 ][ i ][ 1 ] << 2 | |
| for( n = 0; n < 16; n++ ) { | |
| xOffset = n – (n >> 2 ) << 2 | |
| yOffset = n >> 2 | |
| if( significant_coeff_flag[ xS + xOffset ][ yS + yOffset ] ) | |
| **coeff_abs_level_greater1_flag[** n **]** | ae(v) |
| } | |
| for( n = 0; n < 16; n++ ) { | |
| if( coeff_abs_level_greater1_flag[ n ] ) { | |
| **coeff_abs_level_greater2_flag[** n **]** | ae(v) |
| if( coeff_abs_level_greater2_flag[ n ] ) | |
| **coeff_abs_level_minus3[** n **]** | ae(v) |
| } | |
| } | |
| for( n = 0; n < 16; n++ ) { | |
| xOffset = n – (n >> 2 ) << 2 | |
| yOffset = n >> 2 | |
| if( significant_coeff_flag[ xS + xOffset ][ yS + yOffset ] ) { | |
| **coeff_sign_flag[** n **]** | ae(v) |
| transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n + offset ] = ( coeff_abs_level_minus3[ n ] + 3 ) * ( 1 − 2 * coeff_sign_flag[ n ] ) | |
| } else | |
| transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n + offset ] = 0 | |
| } | |
| } | |
| } | |

### 7.3.11 Residual coding CAVLC syntax

| residual_coding_cavlc( x0, y0, log2TrafoSize, trafoDepth, cIdx ) { | **Descriptor** |
|---|---|
| n = ( 1 << ( log2TrafoSize << 1 ) ) >> ( cIdx > 0 ? 2 : 0 ) | |
| **last_pos_level_one** | ce(v) |
| lastPos = last_pos_level_one % n | |
| levelGreaterThanOneFlag = ( last_pos_level_one > n ) | |
| level_and_sign( levelGreaterThanOneFlag, transCoeffLevel, x0, y0, trafoDepth, cIdx, lastPos) | |
| runModeFlag = TRUE | |
| n = lastPos – 1 | |
| sumBigCoeff = 0 | |
| switchThres = ( PredMode == MODE_INTRA && cIdx == 0)? 0 : 49 | |
| while (runModeFlag && n >= 0) { | |
| **run_level_one** | ce(v) |
| runOfZeros = run_level_one % n | |
| levelGreaterThanOneFlag = (run_level_one > n) | |
| trOne = (trOne = = 0 \| \| levelGreaterThanOneFlag) ? 0 : Max(4, trOne + 1) | |
| n = n – runOfZeros | |
| if( n >= 0 ) { | |
| level_and_sign( levelGreaterThanOneFlag, transCoeffLevel, x0, y0, trafoDepth, cIdx, n) | |
| if ( levelGreaterThanOneFlag ) { | |
| sumBigCoeff = sumBigCoeff + level | |
| if ( log2TrafoSize==2 \| \| n > switchThres \| \| sumBigCoeff > 2 ) | |
| runModeFlag = FALSE | |
| } | |
| } | |
| n-- | |
| } | |
| while ( n >= 0 ) { | |
| **level** | ce(v) |
| if ( level > 0 ) { | |
| **sign_flag** | u(1) |
| sign = 1 – 2 * sign_flag | |
| transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n ] = level * sign | |
| } | |
| n-- | |
| } | |
| } | |

### 7.3.12 Level and sign CAVLC syntax

| level_and_sign( levelGreaterThanOneFlag, transCoeffLevel, x0, y0, trafoDepth, cIdx, n ) { | **Descriptor** |
|---|---|
| if (levelGreaterThanOneFlag) { | |
| **level_minus2_and_sign** | ce(v) |
| level = ((level_minus2_and_sign >> 1) + 2 | |
| sign = 1 - 2*(level_minus2_and_sign & 1) | |
| transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n ] = level * sign | |
| } | |
| else { | |
| **sign_flag** | u(1) |
| transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n ] = 1 – 2 *(sign_flag) | |
| } | |
| } | |

## 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, a picture parameter set or a slice of a reference picture.

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, nal_ref_idc equal to 0 for a NAL unit containing a slice indicates that the slice is part of a non-reference picture.

nal_ref_idc shall not be equal to 0 for sequence parameter set or picture parameter set NAL units. When nal_ref_idc is equal to 0 for one NAL unit with nal_unit_type equal to 1 or 4 of a particular picture, it shall be equal to 0 for all NAL units with nal_unit_type equal to 1 or 4 of the picture.

nal_ref_idc shall not be equal to 0 for 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.

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. Since different applications might use NAL unit types 0 and 24..31 for different purposes, particular care must be exercised in the design of encoders that generate NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive, and in the design of decoders that interpret the content of NAL units with nal_unit_type equal to 0 or in the range of 24 to 31, inclusive.

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.

**Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes**

Deleted: 7-1

Deleted: 7-1

| nal_unit_type | Content of NAL unit and RBSP syntax structure | NAL unit type class |
|---|---|---|
| 0 | Unspecified | non-VCL |
| 1 | Coded slice of a non-IDR and non-CDR picture<br>slice_layer_rbsp( ) | VCL |
| 2-3 | Reserved | n/a |
| 4 | Coded slice of a CDR picture<br>slice_layer_rbsp( ) | VCL |
| 5 | Coded slice of an IDR picture<br>slice_layer_rbsp( ) | VCL |
| 6 | Supplemental enhancement information (SEI)<br>sei_rbsp( ) | non-VCL |
| 7 | Sequence parameter set<br>seq_parameter_set_rbsp( ) | non-VCL |
| 8 | Picture parameter set<br>pic_parameter_set_rbsp( ) | non-VCL |
| 9 | Access unit delimiter<br>access_unit_delimiter_rbsp( ) | non-VCL |
| 10-11 | Reserved | n/a |
| 12 | Filler data<br>filler_data_rbsp( ) | non-VCL |
| 13-23 | Reserved | n/a |
| 24..31 | Unspecified | non-VCL |

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. The variable IdrPicFlag is specified as

$$IdrPicFlag = ( nal\_unit\_type == 5 ) ? 1 : 0 \quad (7\text{-}1)$$

When the value of nal_unit_type is equal to 5 for a NAL unit containing a slice of a particular picture, the picture shall not contain NAL units with nal_unit_type equal to 1 or 4. For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, such a picture is referred to as an IDR picture.

**temporal_id** specifies a temporal identifier for the NAL unit. The value of temporal_id shall be the same for all NAL units of an access unit. When an access unit contains any NAL unit with nal_unit_type equal to 5, temporal_id shall be equal to 0.

**output_flag** affects the decoded picture output and removal processes as specified in Annex C.

[Ed. KS: There is no output definition on Annex C yet]

**reserved_one_4bits** shall be equal to 1. Other values of reserved_one_4bits may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of reserved_one_4bits.

**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.

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.

Deleted: Table 7-1

NOTE 6 – 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 7 – 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,

– 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:

1. 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 last zero byte of a byte-aligned three-byte sequence 0x000000 in the RBSP (which is replaced by the four-byte sequence 0x00000300) is taken into account when searching the RBSP data for the next occurrence of byte-aligned bits with the binary patterns specified above.

2. The resulting sequence of bytes is then prefixed as follows.
   – If nal_unit_type is not equal to 14 or 20, the sequence of bytes is prefixed with the first byte of the NAL unit containing the syntax elements forbidden_zero_bit, nal_ref_idc, and nal_unit_type, where nal_unit_type indicates the type of RBSP data structure the NAL unit contains.
   – Otherwise (nal_unit_type is equal to 14 or 20), the sequence of bytes is prefixed with the first four bytes of the NAL unit, where the first byte contains the syntax elements forbidden_zero_bit, nal_ref_idc, and nal_unit_type and the following three bytes contain the syntax structure nal_unit_header_svc_extension( ). The syntax element nal_unit_type in the first byte indicates the presence of the syntax structure nal_unit_header_svc_extension( ) in the following three bytes and the type of RBSP data structure the NAL unit contains.

The process specified above 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,
– 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 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

This subclause specifies the activation process of picture and sequence parameter sets for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

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 for a coded picture shall have the same content as that of the active picture parameter set RBSP for the coded picture unless it follows the last VCL NAL unit of the 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 for a coded video sequence shall have the same content as that of the active sequence parameter set RBSP for the coded video sequence unless it follows the last access unit of the 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 0), 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. For coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, 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

This subclause specifies the order of NAL units and coded pictures and association to access unit for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

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 (when present),

– 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.

– 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.

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 20 to 31, inclusive, is shown in Figure 7-1. [Ed. (TW):adjust text and figure]

Deleted: Figure 7-1

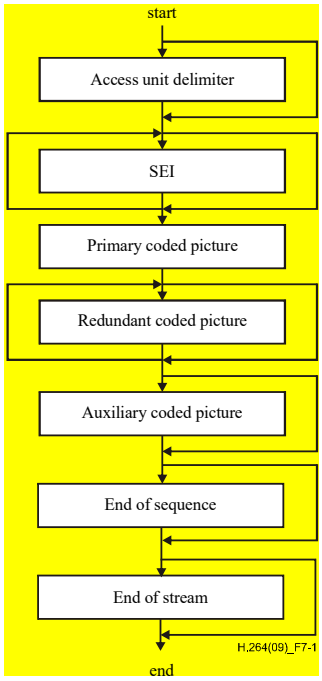



**Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive**

#### 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 for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

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.

– 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.

– IdrPicFlag differs in value.

– IdrPicFlag is equal to 1 for both and idr_pic_id differs in value.

#### 7.4.1.2.5 Order of VCL NAL units and association to coded pictures

[Ed. (TW): insert text]

### 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 coded video sequence conforms.

**reserved_zero_8bits** shall be equal to 0. Decoders shall ignore the value of reserved_zero_8bits.

**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.

**max_temporal_layers_minus1** + 1 specifies the maximum number of temporal layers present in the sequence. The value of max_temporal_layers_minus1 shall be in the range of 0 to 7, inclusive.

**pic_width_in_luma_samples** specifies the width of each decoded picture in luma samples. The value of pic_width_in_luma_samples shall be in the range of 0 to $2^{16}$-1, inclusive.

**pic_height_in_luma_samples** specifies the height of each decoded picture in luma samples. The value of pic_height_in_luma_samples shall be in the range of 0 to $2^{16}$-1, inclusive.

**bit_depth_luma_minus8** + 8 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

$$BitDepth_Y = 8 + bit\_depth\_luma\_minus8 \quad (7\text{-}2)$$
$$QpBdOffset_Y = 6 * bit\_depth\_luma\_minus8 \quad (7\text{-}3)$$

bit_depth_luma_minus8 shall be in the range of 0 to 6, inclusive.

**bit_depth_chroma_minus8** + 8 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

$$BitDepth_C = 8 + bit\_depth\_chroma\_minus8 \quad (7\text{-}4)$$
$$QpBdOffset_C = 6 * bit\_depth\_chroma\_minus8 \quad (7\text{-}5)$$

bit_depth_chroma_minus8 shall be in the range of 0 to 6, inclusive.

**pcm_sample_bit_depth_luma_minus1** + 1 specifies the number of bits used to represent each of PCM sample values of luma component. The value of pcm_sample_bit_depth_luma_minus1 + 1 shall be smaller than or equal to the value of $BitDepth_Y$.

$$PCMBitDepth_Y = 1 + pcm\_sample\_bit\_depth\_luma\_minus1 \quad (7\text{-}6)$$

**pcm_sample_bit_depth_chroma_minus1** + 1 specifies the number of bits used to represent each of PCM sample values of chroma components. The value of pcm_sample_bit_depth_chroma_minus1 + 1 shall be smaller than or equal to the value of $BitDepth_C$.

$$PCMBitDepth_C = 1 + pcm\_sample\_bit\_depth\_chroma\_minus1 \quad (7\text{-}7)$$

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

$$MaxFrameNum = 2^{(\ log2\_max\_frame\_num\_minus4 + 4\ )} \quad (7\text{-}8)$$

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 XX). 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 an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture,

**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 follows:

$$MaxPicOrderCntLsb = 2^{(\ log2\_max\_pic\_order\_cnt\_lsb\_minus4 + 4\ )} \quad (7\text{-}9)$$



Formatted: Font: Bold
Formatted: Subscript
Deleted: 4
Formatted: Subscript
Formatted: Indent: Left: 0.5", Space Before: 9.65 pt, After: 12 pt, Tab stops: 3.37", Centered + 6.74", Right
Formatted: Font: Bold
Formatted: Subscript
Deleted: 4
Deleted: ¶
Deleted: 6
Deleted: 7

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 subclause XX. The value of offset_for_non_ref_pic shall be in the range of $-2^{31}+1$ 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 XX. 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 XX. The value of offset_for_ref_frame[ i ] shall be in the range of $-2^{31}+1$ to $2^{31}-1$, inclusive.

When pic_order_cnt_type is equal to 1, the variable ExpectedDeltaPerPicOrderCntCycle is derived by

```
ExpectedDeltaPerPicOrderCntCycle = 0
for( i = 0;  i < num_ref_frames_in_pic_order_cnt_cycle;  i++ )
    ExpectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]        (7-10)
```

Deleted: 8

**max_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. max_num_ref_frames also determines the size of the sliding window operation. The value of max_num_ref_frames shall be in the range of 0 to MaxDpbFrames, inclusive.

**gaps_in_frame_num_value_allowed_flag** specifies the allowed values of frame_num and the decoding process in case of an inferred gap between values of frame_num.

**log2_min_coding_block_size_minus3** specifies the minimum size of a coding block.

The variable Log2MinCUSize is set equal to log2_min_coding_block_size_minus3 + 3.

**log2_diff_max_min_coding_block_size** specifies the difference between the maximum and minimum coding block size.

The variable Log2MaxCUSize is set equal to log2_min_coding_block_size_minus 3 + 3 + log2_diff_max_min_coding_block_size.

**log2_min_transform_block_size_minus2** specifies the minimum size of a transform block.

The variable Log2MinTrafoSize is set equal to log2_min_transform_block_size_minus2 + 2.

**log2_diff_max_min_transform_block_size** specifies the difference between the maximum and minimum transform size.

The variable Log2MaxTrafoSize is set equal to log2_min_transform_block_size_minus 2 + 2 + log2_diff_max_min_transform_block_size.

The bitstream shall not contain data that result in Log2MaxTrafoSize greater than Log2MaxCUSize.

**log2_min_pcm_coding_block_size_minus3** + 3 specifies the minimum size of I_PCM coding blocks.

Formatted: Font: Bold

The variable Log2MinIPCMCUSize is set equal to log2_min_pcm_coding_block_size_minus3 + 3. The variable Log2MinIPCMCUSize should be equal or less than Log2MaxCUSize.

**max_transform_hierarchy_depth_intra** specifies the maximum hierarchy depth for transform blocks of coding blocks coded in intra prediction mode.

**max_transform_hierarchy_depth_inter** specifies the maximum hierarchy depth for transform units of coding units coded in inter prediction mode.

**chroma_pred_from_luma_enabled_flag** equal to 1 specifies the intra chroma prediction process using the reconstructed luma block is applied according to the intra chroma prediction mode.

Formatted: Font: Bold

**loop_filter_across_slice_flag** equal to 1 specifies the in-loop filtering operations are performed across slice boundary; otherwise, the in-loop operations are slice-independent and not across slice boundary. The in-loop filtering operations include deblocking filter, sample adaptive offset, and adaptive loop filter.

Formatted: Font: Bold

**sample_adaptive_offset_enabled_flag** equal to 1 specifies the sample adaptive offset process is applied to the reconstruced picture after the deblocking filter process.

**adaptive_loop_filter_enabled_flag** equal to 1 specifies the adaptive loop filter process is applied to the reconstructed picture after the deblocking filter process.

**pcm_loop_filter_disable_flag** specifies whether the loop filter process is disabled on reconstructed pixels of I_PCM blocks. If the pcm_loop_filter_disable_flag value is equal to 1, both deblocking and adaptive loop filter processes on the reconstructed pixels of I_PCM blocks are disabled; otherwise the pcm_loop_filter_disable_flag value is equal to 0, both deblocking and adaptive loop filter processes on the reconstructed pixels of I_PCM blocks are not disabled.

[Ed. (WJ): select one expression – enabled_flag or disable_flag]

Formatted: Font: Bold

**cu_qp_delta_enabled_flag** equal to 1 specifies cu_qp_delta presents in coding unit layer to further modify the value of $QP_Y$.

**temporal_id_nesting_flag** specifies whether inter prediction is additionally restricted for the coded video sequence.

Dependent on temporal_id_nesting_flag, the following applies.

– If temporal_id_nesting_flag is equal to 0, additional constraints may not be obeyed.

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

  – For each access unit auA with temporal_id equal to tIdA, an access unit auB with temporal_id equal to tIdB and tIdB less than or equal to tIdA shall not be referenced by inter prediction when there exists an access unit auC with temporal_id equal to tIdC and tIdC less than tIdB, which follows the access unit auB and precedes the access unit auA in decoding order.

NOTE – The syntax element temporal_id_nesting_flag is used to indicate that temporal up-switching, i.e., switching from decoding of up to a specific temporal_id tIdN to decoding up to a temporal_id tIdM > tIdN, is always possible.

#### 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.

– Otherwise (entropy_coding_mode_flag is equal to 1), the method specified by the right descriptor in the syntax table is applied.

**num_temporal_layer_switching_point_flags** specifies how many temporal switching point flags are present. If temporal_id_nesting_flag is equal to 1, num_temporal_layer_switching_point_flags shall be equal to 0.

**temporal_layer_switching_point_flag**[ i ] specifies if the current access point is a temporal switching point allowing the decoding of higher temporal id layers following this access unit. If temporal_id_nesting_flag is equal to 1, temporal_layer_switching_point_flag[ i ] shall be inferred to be equal to 1. If temporal_id_nesting_flag is equal to 0 and num_temporal_layer_switching_point_flags is less than i, temporal_layer_switching_point_flag[ i ] shall be inferred to be equal to 0 .

NOTE – When starting to decode a higher temporal layer i, availability of required reference pictures can be guaranteed immediately following an IDR, or a picture with the temporal_id value j lower than i and temporal_switching_flag[ j ] equal to 1.

**num_ref_idx_l0_default_active_minus1** specifies how num_ref_idx_l0_active_minus1 is inferred for P and B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l0_default_active_minus1 shall be in the range of 0 to 31, inclusive.

**num_ref_idx_l1_default_active_minus1** specifies how num_ref_idx_l1_active_minus1 is inferred for B slices with num_ref_idx_active_override_flag equal to 0. The value of num_ref_idx_l1_default_active_minus1 shall be in the range of 0 to 31, 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 cu_qp_delta is decoded at the coding unit layer. The value of pic_init_qp_minus26 shall be in the range of $-(26 + QpBdOffset_Y)$ to +25, inclusive.

**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 macroblock types.

**slice_granularity** indicates the slice granularity within a picture. The value of slice_granularity shall not be larger than Min( Log2MaxCUSize – 4, log2_diff_max_min_coding_block_size ). The variable SliceGranularity is set to the value of ( slice_granularity << 1 ).

**shared_pps_info_enabled_flag** specifies the shared information in picture parameter set RBSP shall be used for the referred slices. If shared_pps_info_enabled_flag is equal to 1, the alf_param() in picture parameter set RBSP shall be applied for the referred slices; otherwise, the alf_param() in slice header(s) shall be applied.

**max_cu_qp_delta_depth** specifies the maximum allowed depth that is used for specifying $QP_Y$ values for coding unit. The value of max_cu_qp_delta_depth shall be in the range of 0 to 15, inclusive.

The variable log2MinCUDQPSize specifying the minimum coding unit size that can further modifies the value of $QP_Y$ as follows:

log2MinCUDQPSize = Log2MaxCUSize - max_cu_qp_delta_depth (7-11)

#### 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 RBSP 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 RBSP bytes in the SEI payload.

NOTE – The NAL unit byte sequence containing the SEI message might include one or more emulation prevention bytes (represented by emulation_prevention_three_byte syntax elements). Since the payload size of an SEI message is specified in RBSP bytes, the quantity of emulation prevention bytes is not included in the size payloadSize of an 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 payload 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 7-2 for the given value of primary_pic_type.

**Table 7-2 – Meaning of primary_pic_type**

| primary_pic_type | slice_type values that may be present in the primary coded picture |
|---|---|
| 0 | 2 |
| 1 | 0, 2 |
| 2 | 0, 1, 2 |

#### 7.4.2.5 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.



Formatted: Font: Bold

Deleted: 7

Formatted: Heading 5

Deleted: Table 7-2

Deleted: 7-2

Deleted: 7-2

#### 7.4.2.6 Slice layer RBSP semantics

The slice layer RBSP consists of a slice header and slice data.

[Ed. (TW): insert text]

#### 7.4.2.7 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).

#### 7.4.2.8 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[ 0 ], delta_pic_order_cnt[ 1 ], and slice_group_change_cycle shall be the same in all slice headers of a coded picture.

**lightweight_slice_flag** equal to 1 specifies that the value of slice header syntax elements not present shall be inferred to be equal to the value of slice header syntax elements in a proceeding slice, where a proceeding slice is defined as the slice containing treeblock with location (LCUAddress – 1). lightweight_slice_flag shall be equal to 0 when LCUAddress equal to 0.

**slice_type** specifies the coding type of the slice according to Table 7-3.

**Table 7-3 – Name association to slice_type**

| slice_type | Name of slice_type |
|---|---|
| 0 | P (P slice) |
| 1 | B (B slice) |
| 2 | I (I slice) |

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2.

When max_num_ref_frames is equal to 0, slice_type shall be equal to 2.

**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 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

**Deleted: first_tb_in_slice** specifies the address of the first treeblock in the slice.¶
The first treeblock address of the slice is derived as follows.¶
–→ TBD.¶

**Deleted: entropy**

**Deleted:** equal to 1 specifies the slice is an entropy slice. The entropy slice denotes that the entropy decoding process shall use an alternative definition of neighbor availability than the reconstruction process whereas entropy_slice_flag equal to 0 denotes that the entropy decoding and reconstruction processes shall use the same definition of neighbor availability. [Ed.: (WJ) needs to be clarified more]equal to 1 specifies that the value of slice header syntax

**Deleted:** first_tb_in_slice

**Deleted:** first_tb_in_slice

**Deleted:** [Ed.: (WJ) do we need to code lightweight_slice_flag when first_tb_in_slice is equal to 0?]…

**Deleted:** Table 7-3

**Deleted: 7-3**

**Deleted: 7-3**

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.

– 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 both of the following conditions are true.

  – the current picture and the preceding reference picture belong to consecutive access units in decoding order

  – 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 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 )
    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.

**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 length 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.

**num_ref_idx_active_override_flag** equal to 1 specifies that the syntax element num_ref_idx_l0_active_minus1 is present for P and B slices and that the syntax element num_ref_idx_l1_active_minus1 is present for B slices. num_ref_idx_active_override_flag equal to 0 specifies that the syntax elements num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1 are not present.

When the current slice is a P or B slice and field_pic_flag is equal to 0 and the value of num_ref_idx_l0_default_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_default_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.

When the current slice is a P or B slice and num_ref_idx_l0_active_minus1 is not present, num_ref_idx_l0_active_minus1 shall be inferred to be equal to num_ref_idx_l0_default_active_minus1.

The range of num_ref_idx_l0_active_minus1 is specified as follows.p

– 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** specifies the maximum reference index for reference picture list 1 that shall be used to decode the slice.

When the current slice is a B slice and num_ref_idx_l1_active_minus1 is not present, num_ref_idx_l1_active_minus1 shall be inferred to be equal to num_ref_idx_l1_default_active_minus1.

The range of num_ref_idx_l1_active_minus1 is constrained as specified in the semantics for 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.

**first_slice_in_pic_flag** indicates whether the slice is the first slice of the picture. If first_slice_in_pic_flag is equal to 1, the variables SliceAddress and LCUAddress are both set to 0 and the decoding starts with the first LCU in the picture.

**slice_address** specifies the address in slice granularity resolution in which the slice starts and shall be represented by ( Ceil( Log2( NumLCUsInPicture ) ) + SliceGranularity ) bits in the bitstream where NumLCUsInPicture is the number of LCUs in a picture.

The variable LCUAddress is set to ( slice_address >> SliceGranularity ) and represents the LCU part of the slice address in raster scan order. The variable GranularityAddress is set to ( slice_address - ( LCUAddress << SliceGranularity ) ) and represents the sub-LCU part of the slice address expressed in z-scan order.

The variable SliceAddress is then set to ( LCUAddress << ( log2_diff_max_min_coding_block_size << 1 ) ) + ( GranularityAddress << ( ( log2_diff_max_min_coding_block_size << 1 ) – SliceGranularity ) and the slice decoding starts with the largest coding unit possible at the slice starting coordinate.

**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 cu_qp_delta in the coding unit layer. The initial $QP_Y$ quantisation parameter for the slice is computed as

$$SliceQP_Y = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta \qquad (7\text{-}12)$$

The value of slice_qp_delta shall be limited such that $SliceQP_Y$ is in the range of $-QpBdOffset_Y$ to +51, inclusive.

**collocated_from_l0_flag** equal to 1 specifies the picture that contains the co-located partition shall be derived from list 0, otherwise the picture shall be derived from list 1.

Formatted: Font: Bold

Formatted: Font: Bold

Deleted: 9

### 7.4.3.1 Reference picture list modification semantics

The syntax elements modification_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_modification_flag_l0** equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 0. ref_pic_list_modification_flag_l0 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l0 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_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 8.2.2.2 is equal to "no reference picture", ref_pic_list_modification_flag_l0 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList0[ num_ref_idx_l0_active_minus1 ] in the modified list produced as specified in subclause 8.2.2.3 is not equal to "no reference picture".

**ref_pic_list_modification_flag_l1** equal to 1 specifies that the syntax element modification_of_pic_nums_idc is present for specifying reference picture list 1. ref_pic_list_modification_flag_l1 equal to 0 specifies that this syntax element is not present.

When ref_pic_list_modification_flag_l1 is equal to 1, the number of times that modification_of_pic_nums_idc is not equal to 3 following ref_pic_list_modification_flag_l1 shall not exceed num_ref_idx_l1_active_minus1 + 1.

When decoding a slice with slice_type equal to 1 or 6 and RefPicList1[ num_ref_idx_l1_active_minus1 ] in the initial reference picture list produced as specified in subclause 8.2.2.2 is equal to "no reference picture", ref_pic_list_modification_flag_l1 shall be equal to 1 and modification_of_pic_nums_idc shall not be equal to 3 until RefPicList1[ num_ref_idx_l1_active_minus1 ] in the modified list produced as specified in subclause 8.2.2.3 is not equal to "no reference picture".

**modification_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 modification_of_pic_nums_idc are specified in Table 7-4. The value of the first modification_of_pic_nums_idc that follows immediately after ref_pic_list_modification_flag_l0 or ref_pic_list_modification_flag_l1 shall not be equal to 3.

Deleted: Table 7-4

**Table 7-4 – modification_of_pic_nums_idc operations for modification of reference picture lists**

Deleted: 7-4

Deleted: 7-4

| modification_of_pic_nums_idc | modification 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 modification 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 8.2.2.3.1.

[Ed. (TW): clarify the following paragraph]

**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 Reference picture lists combination semantics

**ref_pic_list_combination_flag** equal to 1 indicates that the reference picture list 0 and the reference picture list 1 are combined to be an additional reference picture lists combination used for the prediction units being uni-directional predicted. This flag equal to 0 indicates that the reference picture list 0 and reference picture list 1 are identical thus

reference picture list 0 is used as the reference picture lists combination. The reference picture lists combination is set to be empty at the start of the loop defined in this table.

**num_ref_idx lc_active_minus1**+1 specifies the number of reference pictures selected from reference picture list 0 or reference picture list 1 in the reference picture lists combination.

**ref_pic_list_modification_flag_lc** equal to 1 specifies that the syntax element pic_from_list_0_flag and ref_idx_list_curr are present for specifying the mapping for the entries of the reference picture lists combination to the entries of reference picture list 0 and reference picture list 1.

**ref_pic_list_modification_flag_lc** equal to 0 specifies that these syntax elements are not present. The reference picture lists combination is initialized as specified in subsclause X.X.X.X.

**pic_from_list_0_flag** indicates the current reference picture added into the reference picture lists combination is from reference picture list 0 or reference picture list 1. When this flag is equal to 1, the picture is from the reference picture list 0, and the CurrRefPicList is reference picture list 0; when this flag is equal to 0, the picture is from the reference picture list 1, and the CurrRefPicList is reference picture list 1;

**ref_idx_list_curr** indicates the reference index of the picture in the CurrRefPicList to be appended at the end of the reference picture lists combination.

#### 7.4.3.3 Decoded reference picture marking semantics

[Ed. (TW): clarify the following paragraphs wrt frame/field/picture]

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 content of the decoded reference picture marking syntax structure shall be the same in all slice headers of the primary 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 is 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 is 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. 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 picture, no_output_of_prior_pics_flag equal to 1 may (but should not) 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 picture 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 max_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 7-5, adaptive_ref_pic_marking_mode_flag shall be equal to 1 when the number of pictures that are currently marked as "used for long-term reference" is equal to Max( max_num_ref_frames, 1 ).

**Deleted:** Table 7-5

**Table 7-5 – 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 7-6. 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.

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 picture 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 2 – 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".

Deleted: 7-5

Deleted: 7-5

Deleted: Table 7-6

**Table 7-6 – 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 |

**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 shall be one of the set of picture numbers assigned to reference pictures.

**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 shall be one of the set of long-term picture numbers assigned to reference pictures.

**long_term_frame_idx** is used (with memory_management_control_operation equal to 3 or 6) to assign a long-term picture 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 picture 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 max_num_ref_frames, inclusive.

#### 7.4.3.4 Sample adaptive offset parameter semantics

**sample_adaptive_offset_flag** specifies whether sample adaptive offset applies or not for the current slice.

**sao_split_flag**[ saoDepth ][ x0 ][ y0 ] specifies whethere a region is split into four sub regions with half horizontal and vertical number of LCU. When sao_split_flag[ saoDepth ][ x0 ][ y0 ] is not present, it shall be inferred to be equal to 0.

The maximum allowed depth for sample adaptive offset process SaoMaxDepth is derived as follows:

SaoMaxDepth = Min( 4, Min( Floor( Log2( PicWidthInLCUs ) ), Floor( Log2( PicHeightInLCUs ) ) ) ) (7-13)

[Ed.: (WJ) PicWidthInLCUs and PicHeightInLCUs should be defined elsewhere. Note: SAO assumes that PicWidthInLCUs = Round( $PicWidthInSamples_L$ ÷ ( 1 << Log2MaxCUSize ) )

**sao_type_idx**[ saoDepth ][ x0 ][ y0 ] indicates the offset type for the region specified by saoDepth, x0 and y0.

Deleted: 7-6
Deleted: 7-6
Deleted: 10

**sao_offset**[ saoDepth ][ x0 ][ y0 ][ i ] indicates the offset value of i-th category for the region specified by saoDepth, x0 and y0.

[Ed.: (WJ) x0 and y0 are not sample position here, but indices from 0 to 3 for 4-split sub region. Maybe better to use sample position for the consistency to other parts of the text]

The number of categories, NumSaoClass, is specified in Table 7-7.

**Table 7-7 – Specification of NumSaoClass**

| sao_type_idx[ saoDepth ][ x0 ][ y0 ] | NumSaoCategory | Edge type (informative) |
|---|---|---|
| 0 | 0 | Not applied |
| 1 | 4 | 1D 0-degree edge |
| 2 | 4 | 1D 90-degree edge |
| 3 | 4 | 1D 135-degree edge |
| 4 | 4 | 1D 45-degree edge |
| 5 | 16 | Central band |
| 6 | 16 | Side band |

#### 7.4.3.5 Adaptive loop filter parameter semantics

**adaptive_loop_filter_flag** specifies whether adaptive loop filter applies or not for the current slice.

**alf_region_adaptation_flag** specifies the filter adaptation method shall be applied for the current slice. If alf_region_adaptation_flag is equal to 1, the region-based filter adaptation shall be applied, otherwise, the block-based directional-activity filter adaptation shall be applied.

**alf_length_luma_minus5_div2** specifies the filter length in horizontal direction for the luma component alfTap used in the adaptive loop filter process as specified by

$$alfTap = ( alf\_length\_luma\_minus5\_div2 << 1 ) + 5 \quad (7\text{-}14)$$

The number of coded luma filter coefficients AlfCodedLengthLuma and the number of luma filter coefficients AlfLengthLuma are derived as follows:

– If alfTap is equal to 9, [Ed.: (WJ) truncated diamond shape]

$$AlfCodedLengthLuma = ( ( alfTap * alfTap ) >> 2 ) + 1 \quad (7\text{-}15)$$

$$AlfLengthLuma = ( ( alfTap * alfTap ) >> 1 ) \quad (7\text{-}16)$$

– Otherwise (alfTap is less than 9), [Ed.: (WJ) diamond shape]

$$AlfCodedLengthLuma = ( ( alfTap * alfTap ) >> 2 ) + 2 \quad (7\text{-}17)$$

$$AlfLengthLuma = ( ( alfTap * alfTap ) >> 1 ) + 2 \quad (7\text{-}18)$$

**alf_no_filters_minus1** plus 1 specifies the number of filter sets for the current slice.

**alf_start_second_filter** specifies the variance index of luma samples where the second filter is applied, when alf_no_filters_minus1 is equal to 1.

**alf_filter_pattern**[ i ] specifies the filter index array corresponding to i-th variance index of luma samples, when alf_no_filters_minus1 is greater than 1. The number of filter sets AlfNumFilters is derived as follows:

– If alf_no_filters_minus1 is less than 2,

$$AlfNumFilters = alf\_no\_filters\_minus1 + 1 \quad (7\text{-}19)$$

– Otherwise (alf_no_filters_minus1 is greater than 2)

$$AlfNumFilters = \Sigma_i\ alf\_filter\_pattern[ i ] \text{ with } i = 0..15 \quad (7\text{-}20)$$

Deleted: 7-7
Deleted: 7-7
Deleted: 11
Deleted: 11
Deleted: 12
Deleted: 13
Deleted: 14
Deleted: 15
Deleted: 16

**alf_pred_method** specifies whether the filter coefficients are coded in a predictive way.

**alf_min_kstart_minus1** plus 1 specifies the minimum order k of k-th order exponential golomb code for the luma filter coefficients for the adaptive loop filter.

**alf_golomb_index_bit** specifies the difference in order k of k-th order exponential golomb code for the different groups of the luma filter coefficients. Note that there are several groups of the luma filter coefficients where each group may have different order k.

**alf_coeff_luma**[ i ][ j ] specifies the j-th filter coefficient of i-th filter for the adaptive loop filtering process for luma samples.

**alf_chroma_idc** specifies which chroma components are to be filtered.

**Table 7-8 – Specification of alf_chroma_idc**

| alf_chroma_idc | chroma component to be filtered |
|---|---|
| 0 | None |
| 1 | Cr |
| 2 | Cb |
| 3 | Cb and Cr |

**alf_length_chroma_minus5_div2** specifies the filter length in horizontal direction for the luma component alfTapC used in the adaptive loop filter process as specified by

alfTapC = ( alf_length_chroma_minus5_div2 << 1 ) + 5 (7-21)

The number of coded chroma filter coefficients AlfCodedLengthChroma and the number of chroma filter coefficients AlfLengthChroma are derived as follows:

AlfCodedLengthChroma = ( ( alfTapC * alfTapC – 1 ) >> 1 ) + 2 (7-22)

AlfLengthChroma = ( alfTapC * alfTapC ) + 1 (7-23)

**alf_coeff_chroma**[ i ] specifies the i-th filter coefficient for the adaptive loop filtering process for chroma samples.

#### 7.4.3.6 Adaptive loop filter coding unit control parameter semantics

**alf_cu_control_flag** specifies whether the adaptive loop filter process for luma component shall be applied adaptively according to the coding unit. If alf_cu_control_flag is equal to 1, the filtering process shall be applied only when alf_cu_flag is equal to 1, otherwise, the filtering process shall be applied to all luma samples in the current slice.

**alf_cu_control_max_depth** specifies the maximum split depth from treeblock for transmitting the flag indicating the application of the adaptive loop filter process.

**alf_length_cu_control_info** specifies the information of the number of alf_cu_flag succeeding this information. When the variable specifying the number of alf_cu_flag shall be computed as

NumAlfCuFlag = alf_length_cu_control_info + NumTBsInPicture (7-24)

**alf_cu_flag**[i] specifies the information whether a coding unit is filtered when the adaptive loop filter is applied.

#### 7.4.4 Slice data semantics

**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.

#### 7.4.5 Coding tree semantics

**split_coding_unit_flag**[ x0 ][ y0 ] specifies whether a coding unit is split into coding units with half horizontal and vertical size. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When split_coding_unit_flag[ x0 ][ y0 ] is not present, the following applies:



Deleted: 7-8

Deleted: 7-8

Deleted: 17

Deleted: 18

Deleted: 19

Deleted: alf_length_cu_control_info shall present unless alf_cu_control_max_depth is 0, otherwise, it shall be inferred to be equal to 0. When The

Deleted: The

Deleted: 20

Deleted: [Ed.: (WJ) NumTBsInPicture should be dependent on multiple slice structure]¶

– If log2CUSize is greater than Log2MinCUSize, the value of split_coding_unit_flag[ x0 ][ y0 ] is inferred to be equal to 1.

– Otherwise (log2CUSize is equal to Log2MinCUSize), the value of split_coding_unit_flag[ x0 ][ y0 ] is inferred to be equal to 0.

**cu_split_pred_part_mode**[ x0 ][ y0 ] specifies split_coding_unit_flag and when the coding unit is not split the skip_flag[ x0 ][ y0 ], PredMode and PartMode of a coding unit. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the coding unit relative to the top-left luma sample of the picture.

**Table 7-9 – Specification of cu_split_pred_part_mode when log2CUSize is greater than Log2MinCUSize**

| cu_split_pred_part_mode | split_coding_unit_flag | merge_flag | PredMode | PartMode |
|---|---|---|---|---|
| 0 | 1 | - | - | - |
| 1 | 0 | - | MODE_SKIP | PART_2Nx2N |
| 2 | 0 | 1 | MODE_INTER | PART_2Nx2N |
| 3 | 0 | 0 | MODE_INTER | PART_2Nx2N |
| 4 | 0 | - | MODE_INTER | PART_2NxN |
| 5 | 0 | - | MODE_INTER | PART_Nx2N |
| 6 | 0 | - | MODE_INTRA | PART_2Nx2N |

**Table 7-10 – Specification of cu_split_pred_part_mode when log2CUSize is equal to Log2MinCUSize**

| cu_split_pred_part_mode | split_coding_unit_flag | merge_flag | PredMode | PartMode |
|---|---|---|---|---|
| 0 | 0 | - | MODE_SKIP | PART_2Nx2N |
| 1 | 0 | 1 | MODE_INTER | PART_2Nx2N |
| 2 | 0 | 0 | MODE_INTER | PART_2Nx2N |
| 3 | 0 | - | MODE_INTER | PART_2NxN |
| 4 | 0 | - | MODE_INTER | PART_Nx2N |
| 5 (escape symbol) | 0 | - | MODE_INTRA | PART_2Nx2N |
| | | | MODE_INTRA | PART_NxN |
| | | | MODE_INTER | PARN_NXN |

### 7.4.6 Coding unit semantics

**skip_flag**[ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, no more syntax elements except the motion vector predictor indices are parsed after skip_flag[ x0 ][ y0 ]. skip_flag[ x0 ][ y0 ] equal to 0 specifies that the coding unit is not skipped. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When skip_flag[ x0 ][ y0 ] is not present, it shall be inferred to be equal to 0.

**intra_part_mode** specifies whether the PartMode for the current coding unit is PART_2Nx2N or PART_NxN, when the slice type is I and log2CUSize is equal to Log2MinCUSize. If intra_part_mode is equal to 1, the PartMode is PART_2Nx2N. Otherwise if intra_part_mode is equal to 0, the PartMode is PART_NxN.



Deleted: 7-9
Deleted: 7-9
Deleted: 7-10
Deleted: 7-10
Deleted: 1

**pred_type** specifies prediction mode and partitioning type of the current coding unit. The semantics of pred_type depend on the slice type. The variables PredMode, PartMode and IntraSplitFlag are derived from the value of pred_type as defined in Table 7-11. The value of pred_type shall not be equal to 3 or 5 when slice_type is equal to P or B and log2CUSize is greater than Log2MinCUSize.

When pred_type is not present, the variables PredMode and PartMode are derived as follows.

– If slice_type is equal to I and log2CUSize is greater than Log2MinCUSize,

-PredMode is inferred to be equal to MODE_INTRA

-PartMode is inferred to be equal to PART_2Nx2N

-IntraSplitFlag is inferred to be equal to 0

– Otherwise (slice_type is equal to P or B), if skip_flag[ x0 ][ y0 ] is equal to 1,

-PredMode is inferred to be equal to MODE_SKIP

-PartMode is inferred to be equal to PART_2Nx2N

**pcm_flag** specifies whether the associated coding unit with PART_2Nx2N is coded by I_PCM: If the pcm_flag is equal to 1, the associated coding unit with PART_2Nx2N is coded by I_PCM. When the pcm_flag is not present, it shall be infered to be equal to 0.

**pcm_alignment_zero_bit** is a bit equal to 0.

**pcm_sample_luma**[ i ] represents a coded luma sample value in the raster scan within the coding unit. The number of bits used to represent each of these samples is PCMBitDepth$_Y$.

**pcm_sample_chroma**[ i ] represents a coded chroma sample value in the raster scan within the coding unit. The first half of the values represent coded Cb samples and the remaining half of the values represent coded Cr samples. The number of bits used to represent each of these samples is PCMBitDepth$_C$.

**Table 7-11 - Name association to prediction mode and partitioning type**

| slice_type | pred_type | IntraSplitFlag | PredMode | PartMode |
|---|---|---|---|---|
| I | 0 | 0 | MODE_INTRA | PART_2Nx2N |
| | 1 | 1 | MODE_INTRA | PART_NxN |
| P or B | 0 | - | MODE_INTER | PART_2Nx2N |
| | 1 | - | MODE_INTER | PART_2NxN |
| | 2 | - | MODE_INTER | PART_Nx2N |
| | 3 | - | MODE_INTER | PART_NxN |
| | 4 | 0 | MODE_INTRA | PART_2Nx2N |
| | 5 | 1 | MODE_INTRA | PART_NxN |
| | inferred | - | MODE_SKIP | PART_2Nx2N |

### 7.4.7 Prediction unit semantics

**mvp_idx_l0**[ x0 ][ y0 ] specifies the motion vector predictor index of list 0 where x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When mvp_idx_l0[ x0 ][ y0 ] is not present, it shall be inferred to be equal to 0.

**mvp_idx_l1**[ x0 ][ y0 ] has the same semantics as mvp_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

**mvp_idx_lc**[ x0 ][ y0 ] has the same semantics as mvp_idx_l0, with l0 and list 0 replaced by lc and list combination, respectively.

**prev_intra_luma_pred_flag**[ x0 ][ y0 ], **mpm_idx[** x0 **][** y0 **]** and **rem_intra_luma_pred_mode**[ x0 ][ y0 ] specify the intra prediction mode for luma samples. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. When prev_intra_luma_pred_flag[ x0 ][ y0 ] is equal to 1, the intra prediction mode is inferred from a neighbouring intra-predicted prediction unit according to subclause 8.3.1.

When **mpm_idx[** x0 **][** y0 **]** is not present, it shall be inferred to have a value of 0.



Deleted: Table 7-8

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Subscript

Formatted: Font: Bold

Formatted: No bullets or numbering, Tab stops: Not at 0.2"

Formatted: Subscript

Deleted: 7-11

Deleted: 7-11

Deleted: 8.3.1

The variable IntraLumaModeBins specifies the number of bins indicating the rem_intra_luma_pred_mode[ x0 ][ y0 ]. IntraLumaModeBins depends on the size of the prediction unit and limits the number of available prediction modes as given by Table 7-12.

**Table 7-12 – Number of bins and number of mode for rem_intra_luma_pred_mode associated to PuSize**

| PuSize | IntraLumaModeBins | Number of Modes |
|---|---|---|
| PU_4x4 | 4 | 17 |
| PU_8x8 | 5 | 34 |
| PU_16x16 | 5 | 34 |
| PU_32x32 | 5 | 34 |
| PU_64x64 | 1 | 3 |

**planar_flag_luma**[ x0 ][ y0 ] specifies whether the luma block shall be derived by the planar prediction when IntraPredMode[ x0 ][ y0 ] is equal to 2. If planar_flag_luma[ x0 ][ y0 ] is not present, planar_flag_luma[ x0 ][ y0 ] shall be inferred to be equal to 0.

**intra_chroma_pred_mode**[ x0 ][ y0 ] specifies the intra prediction mode for chroma samples. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

**planar_flag_chroma**[ x0 ][ y0 ] specifies whether the chroma block shall be derived by the planar prediction when IntraPredModeC is equal to 2.

If planar_flag_chroma[ x0 ][ y0 ] is not present (SignaledAsChromaDC is not equal to 1 or IntraPredMode[ x0 ][ y0 ] is equal to Intra_DC or IntraPredMode[ x0 ][ y0 ] is equal to Intra_Planar), planar_flag_chroma[ x0 ][ y0 ] shall be inferred as follows:

- If SignaledAsChromaDC is not equal to 1, planar_flag_chroma[ x0 ][ y0 ] shall be inferred to be equal to 0.
- Otherwise, if IntraPredMode[ x0 ][ y0 ] is equal to Intra_DC or Intra_Planar, the following applies.
  - If chroma_pred_from_luma_enabled_flag is equal to 1, the following applies.
    - If intra_chroma_pred_mode[ x0 ][ y0 ] is equal to 3, planar_flag_chroma[ x0 ][ y0 ] is set equal to 1 – planar_flag_luma[ x0 ][ y0 ]. [Ed. (WJ): possible conflict with codeword switch]
    - Otherwise (intra_chroma_pred_mode[ x0 ][ y0 ] is equal to 4), planar_flag_chroma[ x0 ][ y0 ] is set equal to planar_flag_luma[ x0 ][ y0 ].
  - Otherwise (chroma_pred_from_luma_enabled_flag is equal to 0), the following applies.
    - If intra_chroma_pred_mode[ x0 ][ y0 ] is equal to 2, planar_flag_chroma[ x0 ][ y0 ] is set equal to 1 – planar_flag_luma[ x0 ][ y0 ]. [Ed. (WJ): possible conflict with codeword switch]
    - Otherwise (intra_chroma_pred_mode[ x0 ][ y0 ] is equal to 3), planar_flag_chroma[ x0 ][ y0 ] is set equal to planar_flag_luma[ x0 ][ y0 ].

[Ed. (WJ): suggested to be noted from proponents – under the specific value of intra_chroma_pred_mode, the planar_flag_chroma may be inferred based on planar_flag_luma. The binarization of intra_chroma_pred_mode should take account of this property and the codeword switching as listed in the open issue]

**merge_flag**[ x0 ][ y0 ] specifies whether the inter prediction parameters for the current prediction unit are inferred from a neighbouring inter-predicted partition. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When merge_flag[ x0 ][ y0 ] is not present (InferredMergeFlag is equal to 1), it is inferred to be equal to 1.

**merge_idx**[ x0 ][ y0 ] specifies the merging candidate index of the merging candidate list where x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When merge_idx[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.



Deleted: Table 7-11
Deleted: 7-12
Deleted: 7-12
Deleted: PU_2x2 ... [6]
Deleted: PU_128x128 ... [7]
Formatted: Font: Bold
Deleted: s
Formatted: Font: Bold
Formatted: Font:
Formatted: Font:
Formatted ... [8]
Formatted ... [10]
Formatted ... [9]
Formatted ... [11]
Formatted ... [12]
Formatted ... [13]
Formatted ... [14]
Formatted ... [15]
Formatted ... [16]
Formatted ... [17]
Formatted ... [18]
Formatted ... [19]
Formatted ... [20]
Formatted ... [21]
Formatted ... [22]
Formatted ... [23]
Formatted ... [24]
Formatted ... [25]
Formatted ... [26]
Formatted ... [27]
Formatted ... [28]
Formatted ... [29]
Formatted ... [30]
Formatted ... [31]
Formatted ... [32]
Formatted ... [33]
Formatted ... [34]
Formatted ... [35]
Formatted ... [36]
Formatted ... [37]
Formatted ... [38]
Formatted ... [39]
Formatted ... [40]
Formatted ... [41]
Formatted ... [42]
Formatted ... [43]
Formatted ... [44]
Formatted ... [45]
Formatted ... [46]
Formatted ... [47]
Deleted: <#>¶

**combined_inter_pred_ref_idx** specifies the values of inter_pred_flag and ref_idx_lX with X being 0, 1 or c. When it is not present, it shall be inferred to be equal to 0. [Ed. (WJ): needs to be improved]

The variables NumPredRefLC, NumPredRefL0 and NumPredRefL1 are specified as

NumPredRefLC = Min( 4, num_ref_idx_lc_active_minus1 + 1 ) (7-25)
NumPredRefL0 = Min( 2, num_ref_idx_l0_active_minus1 + 1 ) (7-26)
NumPredRefL1 = Min( 2, num_ref_idx_l1_active_minus1 + 1 ) (7-27)

The variable MaxPredRef is specified as

– If num_ref_idx_lc_active_minus1 is greater than 0, the following applies.

MaxPredRef = NumPredRefLC + NumPredRefL0 * NumPredRefL1 (7-28)

– Otherwise (combined list is not used), the following applies.

MaxPredRef = NumPredRefL0 + NumPredRefL0 * NumPredRefL1 (7-29)

The function use_combined_inter_pred_ref( x, y ) is specified as follows: [Ed. (WJ): move to syntax function section?]

use_combined_inter_pred_ref ( x, y ) = check_inter_pred_ref_neighbour( x-1, y ) &&
check_inter_pred_ref_neighbour( x, y-1 ) &&
check_inter_pred_ref_neighbour( x-1, y-1 )

The function check_inter_pred_ref_neighbour( x, y ) is specified as follows: [Ed. (WJ): move to syntax function section?]

if x or y is outside current slice, check_inter_pred_ref_neighbour( x, y ) = true
else if ( num_ref_idx_lc_active_minus1 <= 0 && ref_idx_l0[ x ][ y ] < 2 && ref_idx_l1[ x ][ y ] < 2 )
check_inter_pred_ref_neighbour( x, y ) = true
else if ( num_ref_idx_lc_active_minus1 > 0 && inter_pred_flag[ x ][ y ] != PredLC &&
ref_idx_l0[ x ][ y ] < 2 && ref_idx_l1[ x ][ y ] < 2 )
check_inter_pred_ref_neighbour( x, y ) = true
else if ( num_ref_idx_lc_active_minus1 > 0 && inter_pred_flag[ x ][ y ] == PredLC && ref_idx_lc[ x ][ y ] < 4 )
check_inter_pred_ref_neighbour( x, y ) = true
else
check_inter_pred_ref_neighbour( x, y ) = false

**inter_pred_flag**[ x0 ][ y0 ] specifies whether uni-prediction, or bi-prediction is used for the current prediction unit according to Table 7-13. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

**Table 7-13 – Name association to inter prediction mode**

| slice_type | inter_pred_flag | Name of inter_pred_flag |
|---|---|---|
| P | inferred | Pred_L0 |
| B | 0 | Pred_LC |
| | 1 | Pred_BI |

When entropy_coding_mode_flag is equal to 1 and inter_pred_flag[ x0 ][ y0 ] is not present, it shall be inferred to be equal to Pred_L0 when slice_type is equal to P and Pred_BI when slice_type is equal to B.

When entropy_coding_mode_flag is equal to 0, inter_pred_flag[ x0 ][ y0 ] is not present and combined_inter_pred_ref_idx is present, it shall be inferred as follows:



Formatted: Font: Bold
Deleted: 18
Deleted: 18
Deleted: 18
Formatted: Normal, Indent: Left: 0", Tab stops: 0.2", Left + Not at 0.59" + 0.79" + 0.98"
Deleted: 18
Deleted: 18
Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: Normal, Indent: Left: 0", Tab stops: 0.3", Left + Not at 0.59" + 0.79" + 0.98"
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: English (UK)
Formatted: Tab stops: 0.3", Left
Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
Formatted: Normal, Indent: Left: 0", Tab stops: 0.3", Left + Not at 0.59" + 0.79" + 0.98"
Deleted: ¶
Deleted: Table 7-11
Deleted: 7-13
Deleted: 7-13

– If num_ref_idx_lc_active_minus1 is greater than 0 and combined_inter_pred_ref_idx < NumPredRefLC, inter_pred_flag[ x0 ][ y0 ] is set equal to Pred_LC,

– Otherwise, if num_ref_idx_lc_active_minus1 is equal to 0 and combined_inter_pred_ref_idx < NumPredRefL0, inter_pred_flag[ x0 ][ y0 ] is set equal to Pred_L0,

– Otherwise, inter_pred_flag[ x0 ][ y0 ] is set equal to Pred_BI,

**ref_idx_l0**[ x0 ][ y0 ] specifies the list 0 reference picture index for the current prediction unit. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture.

When ref_idx_l0[ x0 ][ y0 ] is not present and combined_inter_pred_ref_idx is present, ref_idx_l0[ x0 ][ y0 ] shall be inferred to be equal to ( ( combined_inter_pred_ref_idx – NumPredRefLC ) / NumPredRefL1 ).

**ref_idx_l0_minusX**[ x0 ][ y0 ] + X specifies the value of ref_idx_l0[ x0 ][ y0 ] when combined_inter_pred_ref_idx is present and it is equal to MaxPredRef.

The constant X is derived as follows:

– If inter_pred_flag[ x0 ][ y0 ] is Pred_L0, X is set equal to 2,

– Otherwise, X is set equal to 0.

**ref_idx_l1**[ x0 ][ y0 ] has the same semantics as ref_idx_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

When ref_idx_l1[ x0 ][ y0 ] is not present and combined_inter_pred_ref_idx is present, ref_idx_l1[ x0 ][ y0 ] shall be inferred to be equal to ( ( combined_inter_pred_ref_idx – NumPredRefLC ) % NumPredRefL1 ).

**ref_idx_l1_minusX**[ x0 ][ y0 ] + X specifies the value of ref_idx_l1[ x0 ][ y0 ] when combined_inter_pred_ref_idx is present and it is equal to MaxPredRef.

The constant X is derived as follows:

– If ref_idx_l0[ x0 ][ y0 ] is less than 2, X is set equal to 2,

– Otherwise, X is set equal to 0,

**ref_idx_lc**[ x0 ][ y0 ] has the same semantics as ref_idx_l0, with l0 and list 0 replaced by lc and list combination, respectively.

When ref_idx_lc[ x0 ][ y0 ] is not present and combined_inter_pred_ref_idx is present, ref_idx_lc[ x0 ][ y0 ] shall be inferred to be equal to combined_inter_pred_ref_idx.

**ref_idx_lc_minus4**[ x0 ][ y0 ] + 4 specifies the value of ref_idx_lc [ x0 ][ y0 ] when combined_inter_pred_ref_idx is present and it is equal to MaxPredRef,

**mvd_l0**[ x0 ][ y0 ][ compIdx ], specifies the difference between a list 0 vector component to be used and its prediction. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered prediction block relative to the top-left luma sample of the picture. The horizontal motion vector component difference is assigned compIdx = 0 and the vertical motion vector component is assigned compIdx = 1.

When any of the two components is not present, the inferred value is 0.

**mvd_l1**[ x0 ][ y0 ][ compIdx ] has the same semantics as mvd_l0, with l0 and list 0 replaced by l1 and list 1, respectively.

**mvd_lc**[ x0 ][ y0 ][ compIdx ] has the same semantics as mvd_l0, with l0 and list 0 replaced by lc and list combination, respectively.

### 7.4.8 Transform tree semantics

**no_residual_data_flag** equal to 1 specifies that no residual data are present for the current coding unit. no_residual_data_flag equal to 0 specifies that residual data are present for the current coding unit.

When no_residual_data_flag is not present, its value shall be inferred to be equal to 0.

**split_transform_flag**[ x0 ][ y0 ][ trafoDepth ] specifies whether a block is split into blocks with half horizontal and vertical size for the purpose of transform coding. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units.

When split_transform_flag[ x0 ][ y0 ][ trafoDepth ] is not present, it is inferred as follows:



Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27"

Formatted: Font:

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27"

Deleted: ¶

Formatted: English (UK)

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27"

Formatted: Font: English (UK)

Deleted: ¶

Formatted: Font: Bold

Formatted: English (US)

– If log2TrafoSize is greater than Log2MaxTrafoSize or intraSplitFlag is equal to 1, the value of split_transform_flag[ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 1.

– Otherwise (log2TrafoSize is less than or equal to Log2MaxTrafoSize and intraSplitFlag is equal to 0), the value of split_transform_flag[ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 0.

**cbf_luma**[ x0 ][ y0 ][ trafoDepth ] equal to 1 specifies that the luma transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units.

When cbf_luma[ x0 ][ y0 ][ trafoDepth ] is not present, it is inferred to be equal to 1.

**cbf_cb**[ x0 ][ y0 ][ trafoDepth ] equal to 1 specifies that the Cb transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units.

When cbf_cb[ x0 ][ y0 ][ trafoDepth ] is not present and PredMode is not equal to MODE_INTRA, the value of cbf_cb[ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 0.

**cbf_cr**[ x0 ][ y0 ][ trafoDepth ] equal to 1 specifies that the Cr transform block contains one or more transform coefficient levels not equal to 0. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units.

When cbf_cr[ x0 ][ y0 ][ trafoDepth ] is not present and PredMode is not equal to MODE_INTRA, the value of cbf_cr[ x0 ][ y0 ][ trafoDepth ] is inferred to be equal to 0.

**cbp_and_split_transform** contains a number of coded block flags and/or a split_transform_flag. Depending on a 4-bit value flagPattern defined as follows, the value of cbp_and_split_transform is defined according to Table 7-14

Deleted: Table 7-13

```
codeLuma = trafoDepth == 0 || cbf_luma[ xB ][ yB ][ trafoDepth − 1 ]
codeCb =   trafoDepth == 0 || (cbf_cb[ xB ][ yB ][ trafoDepth − 1 ] &&
           log2TrafoSize > Log2MinTrafoSize )
codeCr =   trafoDepth == 0 || (cbf_cr[ xB ][ yB ][ trafoDepth − 1 ] &&
           log2TrafoSize > Log2MinTrafoSize )                                (7-30)
codeSplitTrans = log2TrafoSize <= Log2MaxTrafoSize &&
                 log2TrafoSize > Log2MinTrafoSize &&
                 trafoDepth < maxDepth &&
                 !intraSplitFlag
flagPattern = (codeLuma << 3) + (codeCb << 2) + (codeCr << 1) + codeSplitTrans
```

Deleted: 23

The values of codeLuma, codeCb, codeCr and codeSplitTrans specify respectivley if an associated flag of cbf_luma, cbf_cb, cbf_cr, and split_transform_flag is included in the syntax element cbp_and_split_transform.

In Table 7-14, Luma, Cb, Cr and Split denote the values of cbf_luma[ x0 ][ y0 ][ trafoDepth ], cbf_cb[ x0 ][ y0 ][ trafoDepth ], cbf_cr[ x0 ][ y0 ][ trafoDepth ] and split_transform_flag[ x0 ][ y0 ][ trafoDepth ] respectively. Luma1, Luma2 and Luma3 denote respectively the values of cbf_luma[ x1 ][ y0 ][ trafoDepth ], cbf_luma[ x0 ][ y1 ][ trafoDepth ] and cbf_luma[ x1 ][ y1 ][ trafoDepth ]. The values of x1 and y1 are derived as

Deleted: Table 7-13

```
x1 = x0 + ( ( 1 << log2TrafoSize ) >> 1 )                                    (7-31)
```

Deleted: 21

```
y1 = y0 + ( ( 1 << log2TrafoSize ) >> 1 )                                    (7-32)
```

Deleted: 21

**Table 7-14 – Specification of cbp_and_split_transform**

| flagPattern | cbp_and_split_transform |
|---|---|
| 8 | (Luma<<3) + (Luma1<<2) + (Luma2<<1) + Luma3 - 1 |
| 9 | (Luma<<1) + Split |
| 10 | (Luma<<1) + Cr |
| 11 | (Luma<<2) + (Cr<<1) + Split |
| 12 | (Luma<<1) + Cb |
| 13 | (Luma<<2) + (Cb<<1) + Split |
| 14 | (Luma<<2) + (Cb<<1) + Cr |
| 15 | (Luma<<2) + ((Cb\|Cr)<<1) + Split |

### 7.4.9 Transform coefficient semantics

The transform coefficient levels are parsed into the arrays transCoeffLevel[ x0 ][ y0 ][ trafoDepth ][ cIdx ][ n ]. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered transform block relative to the top-left luma sample of the picture. The array index trafoDepth specifies the current subdivision level of a coding unit into blocks for the purpose of transform coding. trafoDepth is equal to 0 for blocks that correspond to coding units. The array index cIdx specifies an indicator for the colour component; it is equal to 0 for luma, equal to 1 for Cb, and equal to 2 for Cr. The array index n specifies the scanning position of the transform coefficient levels.

**cu_qp_delta** can change the value of $QP_Y$ for a quantization group of coding units where the quantization group of coding units is specified as follows:

– If a coding unit with the split_coding_unit_flag[ x0 ][ y0 ] equal to 0 and the log2CUSize is greater than or equal to log2MinCUDQPSize, the quantization group includes this coding unit only.

– Otherwise, if a coding unit with the split_coding_unit_flag[ x0 ][ y0 ] equal to 1 and the log2CUSize is equal to log2MinCUDQPSize, the quantization group includes all coding units split from this coding unit.

The decoded value of cu_qp_delta shall be in the range of $-( 26+ QpBdOffset_Y / 2 )$ to $+( 25+ QpBdOffset_Y / 2 )$, inclusive. cu_qp_delta shall be inferred to be equal to 0 when it is not present for any quantization group of coding units.

The value of $QP_Y$ is derived as

$$QP_Y = ( ( QP_{Y,PREV} + cu\_qp\_delta + 52 + 2*QpBdOffset_Y )\%( 52 + QpBdOffset_Y ) ) - QpBdOffset_Y \quad (7\text{-}33)$$

where $QP_{Y,PREV}$ is the luma quantization parameter, $QP_Y$, of the left neighbor quantization group of coding units in the current slice. If the left neighbor quantization group in the current slice is not available, $QP_{Y,PREV}$ is the luma quantization parameter, of the previous quantization group in decoding order in the current slice. For the first quantization group of coding units in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ at the start of each slice.

The value of $QP'_Y$ is derived as

$$QP'_Y = QP_Y + QpBdOffset_Y \quad (7\text{-}34)$$

When PredMode is equal to MODE_INTRA, different scanning orders are used. The arrays specifying the scanning order are derived as follows.

– The array ScanType[ log2TrafoSize ][ IntraPredMode ], specifying the scanning order for various transform block sizes and luma intra prediction modes, is derived as specified in Table 7-15

Deleted: 7-14
Deleted: 7-14
Deleted: 21
Deleted: 22
Deleted: − 2
Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left + Not at 0.28"
Deleted: Table 7-12

**Table 7-15 – Specification of ScanType[ log2TrafoSize ][ IntraPredMode ]**

| IntraPredMode | log2TrafoSize | | | |
|---|---|---|---|---|
| | 2 | 3 | 4 | 5 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 2 | 2 | 0 | 0 |
| 2-3 | 0 | 0 | 0 | 0 |
| 4-5 | 1 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |
| 7-8 | 2 | 2 | 0 | 0 |
| 9-10 | 0 | 0 | 0 | 0 |
| 11-12 | 1 | 1 | 0 | 0 |
| 13-14 | 0 | 0 | 0 | 0 |
| 15-16 | 2 | 2 | 0 | 0 |
| 17-19 | 0 | 0 | 0 | 0 |
| 20-23 | 1 | 1 | 0 | 0 |
| 24-27 | 0 | 0 | 0 | 0 |
| 28-31 | 2 | 2 | 0 | 0 |
| 32-33 | 0 | 0 | 0 | 0 |
| 34 | 0 | 0 | 0 | 0 |
| 35 | 0 | 0 | 0 | 0 |

– The array ScanTypeC[ log2TrafoSize ][ IntraPredModeC ], specifying the scanning order for various transform block sizes and chroma intra prediction modes, is set equal to ScanType[ log2TrafoSize + 1 ][ IntraPredModeC ]

[Ed. (WJ): the index log2TrafoSize + 1 is valid only for 4:2:0 chroma sampling]

**7.4.10**

### 7.4.10 Residual coding CABAC semantics

The array ScanOrder[ log2BlockSize − 2 ][ scanIdx ][ pos ][ comp ] specifies the mapping of the scan position pos, ranging from 0 to ( ( 1 << log2BlockSize ) * ( 1 << log2BlockSize ) ) – 1, to horizontal and vertical components of the scan-order matrix. The array index scanIdx equal to 0 specifies a zig-zag scan, scanIdx equal to 1 specifies a horizontal scan and scanIdx equal to 2 speicifes a vertical scan. The array index comp equal to 0 specifies the horizontal component and the array index comp equal to 1 specifies the vertical component. The array ScanOrder is derived as follows.

The scanning order array initialisation process as specified in 6.5 is invoked with 1 << log2TrafoSize − 2 as input and the output is assigned to ScanOrder[ log2TrafoSize − 2 ].

**last_significant_coeff_x, last_significant_coeff_y** specify the column and the row position of the last significant coefficient in scanning order within a transform block. The values of last_significant_coeff_x and last_significant_coeff_y are ranging from 0 to ( 1 << log2TrafoSize ) − 1.

When scanIdx is equal to 2, the coordinates are swapped as follows.

```
temp                 = last_significant_coeff_x
last_significant_coeff_x = last_significant_coeff_y        (7-35)
last_significant_coeff_y = temp
```

**significant_coeff_flag[** xC **][** yC **]** specifies for the transform coefficient position ( xC, yC ) within the current transform block whether the corresponding transform coefficient level at location ( xC, yC ) is non-zero as follows.

– If significant_coeff_flag[ xC ][ yC ] is equal to 0, the transform coefficient level at location ( xC, yC ) is set equal to 0;

– Otherwise (significant_coeff_flag[ xC ][ yC ] is equal to 1), the transform coefficient level at location ( xC, yC ) has a non-zero value.



Deleted: 7-15
Deleted: 7-15
Deleted: – 2
Deleted: n/a
Deleted: n/a
Deleted: n/a
Deleted: n/a
Deleted: – 2
Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left + Not at 0.28"
Deleted: derived as follows.specified in Table 7-13. The scanning order for IntraPredModeC equal to 4 is set equal to the corresponding luma scanning order ScanType[ log2TrafoSize − 2 ][ IntraPredMode ].¶ If IntraPredModeC is less than 4, ScanTypeC[ log2TrafoSize ][ IntraPredModeC ] is set equal to …
Deleted: ,¶ Otherwise (IntraPredModeC is equal to 4), ScanTypeC[ log2TrafoSize ][ IntraPredModeC ] is set equal to ScanType[ log2TrafoSize + 1 ][ IntraPredMode ].…
Formatted: No bullets or numbering, Tab stops: 0.59", Left + Not at 0.28"
Deleted: Table 7-167-167-16 – Specification of ScanTypeC[ log2TrafoSize – 2 ][ IntraPredModeC ][ IntraPredModeC ¶ ... [48]
Deleted: 23

The significant_coeff_flag[ last_significant_coeff_x ][ last_significant_coeff_y ] at the last significant location ( last_significant_coeff_x, last_significant_coeff_y ) in scan order is inferred to be equal to 1.

When significant_coeff_flag[ xC ][ yC ] is not present, it is inferred to be equal to 0.

**coeff_abs_level_greater1_flag[** n **]** specifies for the scanning position n whether there are transform coefficient levels greater than 1.

When coeff_abs_level_greater1_flag[ n ] is not present, it is inferred to be equal to 0.

**coeff_abs_level_greater2_flag[** n **]** specifies for the scanning position n whether there are transform coefficient levels greater than 2.

When coeff_abs_level_greater2_flag[ n ] is not present, it is inferred to be equal to 0.

**coeff_abs_level_minus3[** n **]** is the absolute value of a transform coefficient level minus 3 at the scanning position n. The value of coeff_abs_level_minus3 is constrained by the limits in subclause XX.

When coeff_abs_level_minus3[ n ] is not present, it is inferred as follows.

– If coeff_abs_level_greater1_flag[ n ] is equal to 0, coeff_abs_level_minus3[ n ] is inferred to be equal to −2.

– Otherwise (coeff_abs_level_greater1_flag[ n ] is equal to 1), coeff_abs_level_minus3[ n ] is inferred to be equal to −1.

**coeff_sign_flag[** n **]** specifies the sign of a transform coefficient level for the scanning position n as follows.

– If coeff_sign_flag[ n ] is equal to 0, the corresponding transform coefficient level has a positive value.

– Otherwise (coeff_sign_flag[ n ] is equal to 1), the corresponding transform coefficient level has a negative value.

When coeff_sign_flag[ n ] is not present, it is inferred to be equal to 0.

#### 7.4.11 Residual coding CAVLC semantics

The following CAVLC syntax elements are used to decode a block of nMax transform coefficients.

**last_pos_level_one** specifies the position of the last non-zero valued transform coefficient in the scan and whether the absolute value of the transform coefficient is larger than 1.

– The position of the last non-zero transform coefficient in scan order is set equal to last_pos_level_one%nMax.

– If last_pos_level_one is less than nMax, the absolute value of the transform coefficient is equal to 1.

– Otherwise (last_pos_level_one is not less than nMax), the absolute value of the transform coeficient is larger than 1.

**level_minus2_and_sign** specifies the value of a transform coefficient that has an absolute value that is larger than one as follows.

– The absolute value of the transform coefficient is equal to (level_minus2_and_sign>>1) + 2.

– If (level_minus2_and_sign & 1) is equal to 0, the transform coefficient has a positive value.

– Otherwise ((level_minus2_and_sign & 1) is equal to 1), the transform coefficient has a negative value.

**sign_flag** specifies the sign of a non-zero valued transform coefficient as follows.

– If sign_flag is equal to 0, the transform coefficient has a positive value.

– Otherwise (sign_flag is equal to 1), the transform coefficient has a negative value.

**run_level_one** specifies the number of consecutive transform coefficients in the scan with zero value before a non-zero valued transform coefficients and whether the absolute value of the non-zero valued transform coefficient is larger than 1.

– Given the scan position n of the current non-zero valued transform coefficient, the number of consecutive transform coefficients with zero value before the next non-zero valued transform coeficient is set equal to run_level_one%n.

– If run_level_one is less than n, the absolute value of the transform coefficient is equal to 1.

– Otherwise (run_level_one is not less than n), the absolute value of the next non-zero valued transform coefficient is larger than 1.

**level** specifies the absolute value of a non-zero valued transform coefficient.

## 8 Decoding process

Outputs of this process are decoded samples of the current picture (sometimes referred to by the variable CurrPic).

Depending on the value of chroma_format_idc, the number of sample arrays of the current picture is as follows.

– If chroma_format_idc is equal to 0, the current picture consists of 1 sample array $S_L$.

– Otherwise (chroma_format_idc is not equal to 0), the current picture consists of 3 sample arrays $S_L$, $S_{Cb}$, $S_{Cr}$.

This clause describes the decoding process, given syntax elements and upper-case variables from clause 6.6.

Deleted: 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 complete primary coded picture or part of a primary coded picture. Each slice referred to in this clause is a slice of a primary coded picture. Each slice data partition referred to in this clause is a slice data partition of a primary coded picture.

Depending on the value of separate_colour_plane_flag, the decoding process is structured as follows.

– If separate_colour_plane_flag is equal to 0, the decoding process is invoked a single time with the current picture being the output.

– Otherwise (separate_colour_plane_flag is equal to 1), the decoding process is invoked three times. Inputs to the decoding process are all NAL units of the primary coded picture with identical value of colour_plane_id. The decoding process of NAL units with a particular value of colour_plane_id is specified as if only a coded video sequence with monochrome colour format with that particular value of colour_plane_id would be present in the bitstream. The output of each of the three decoding processes is assigned to the 3 sample arrays of the current picture with the NAL units with colour_plane_id equal to 0 being assigned to $S_L$, the NAL units with colour_plane_id equal to 1 being assigned to $S_{Cb}$, and the NAL units with colour_plane_id equal to 2 being assigned to $S_{Cr}$.

NOTE – The variable ChromaArrayType is derived as 0 when separate_colour_plane_flag is equal to 1 and chroma_format_idc is equal to 3. In the decoding process, the value of this variable is evaluated resulting in operations identical to that of monochrome pictures with chroma_format_idc being equal to 0.

An overview of the decoding process is given as follows:

1. The decoding of NAL units is specified in subclause 8.1.
2. 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)
   – [Ed. (TW): clarify the following paragraph] 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 8.2.3.2 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 subclause 8.2.2 is invoked 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 8.2.2.4 specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.
3. The processes in subclauses 8.3, ABC specify decoding processes using syntax elements in the treeblock layer and above.

[Ed. (TW): insert corresponding text]

### 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 and 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.

## 8.2 Slice decoding process

### 8.2.1 Decoding process for picture order count

[Ed. (TW): clarify the following paragraphs]

#### 8.2.1.1 Decoding process for picture order count type 0

#### 8.2.1.2 Decoding process for picture order count type 1

#### 8.2.1.3 Decoding process for picture order count type 2

### 8.2.2 Decoding process for reference picture lists construction

[Ed. (TW): clarify the following paragraphs]

This process is invoked at the beginning of the decoding process for each P, SP, or B slice.

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 8.2.2.4. Short-term reference pictures are identified by the value of frame_num. Long-term reference pictures are assigned a long-term picture index as specified by the bitstream and specified in subclause 8.2.2.4.

Subclause 8.2.2.1 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 **Error! Reference source not found.** 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.

Deleted: Error! Reference source not found.

At the beginning of the decoding process for each slice, reference picture list RefPicList0, and for B slices RefPicList1, are derived as specified by the following ordered steps:

1. An initial reference picture list RefPicList0 and for B slices RefPicList1 are derived as specified in subclause 8.2.2.2.
2. When ref_pic_list_modification_flag_l0 is equal to 1 or, when decoding a B slice, ref_pic_list_modification_flag_l1 is equal to 1, the initial reference picture list RefPicList0 and, for B slices, RefPicList1 are modified as specified in subclause 8.2.2.3.

   NOTE – The modification process for reference picture lists specified in subclause 8.2.2.3 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 8.2.2.2.

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.

#### 8.2.2.1 Decoding process for picture numbers

This process is invoked when the decoding process for reference picture lists construction specified in subclause 8.2.2, the decoded reference picture marking process specified in subclause 8.2.2.4, or the decoding process for gaps in frame_num specified in subclause 8.2.3.2 is invoked.

The variables FrameNum, FrameNumWrap, PicNum, LongTermFrameIdx, and LongTermPicNum are used for the initialisation process for reference picture lists in subclause 8.2.2.2, the modification process for reference picture lists in

subclause 8.2.2.3, the decoded reference picture marking process in subclause 8.2.2.4, and the decoding process for gaps in frame_num in subclause 8.2.3.2.

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                    (8-1)
else
    FrameNumWrap = FrameNum
```

where the value of frame_num used in Equation 8-1 is the frame_num in the slice header(s) for the current picture.

Deleted: 8-1

Each long-term reference picture has an associated value of LongTermFrameIdx (that was assigned to it as specified in subclause 8.2.2.4).

To each short-term reference picture a variable PicNum is assigned, and to each long-term reference picture a variable LongTermPicNum is assigned.

$$PicNum = FrameNumWrap \tag{8-2}$$

$$LongTermPicNum = LongTermFrameIdx \tag{8-3}$$

### 8.2.2.2 Initialisation process for reference picture lists

This initialisation process is invoked when decoding a P or B slice header.

RefPicList0 and RefPicList1 have initial entries as specified in subclauses 8.2.2.2.1 through 8.2.2.2.2.

When the number of entries in the initial RefPicList0 or RefPicList1 produced as specified in subclauses 8.2.2.2.1 through 8.2.2.2.2 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 8.2.2.2.1 through 8.2.2.2.2 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".

#### 8.2.2.2.1 Initialisation process for the reference picture list for P slices

Formatted: Heading 5

This initialisation process is invoked when decoding a P slice in a coded picture.

When this process is invoked, there shall be at least one reference picture that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing". Pictures with higher values of temporal_id than the current picture cannot be used for reference, and are not included in the reference picture list.

The reference picture list RefPicList0 is ordered so that short-term reference pictures have lower indices than long-term reference pictures.

The short-term reference pictures are ordered starting with the picture with the highest PicNum value and proceeding through in descending order to the picture with the lowest PicNum value, excluding any picture with a temporal_id value higher than that of the current picture.

The long-term reference pictures are ordered starting with the picture with the lowest LongTermPicNum value and proceeding through in ascending order to the picture with the highest LongTermPicNum value, excluding any picture with a temporal_id value higher than that of the current picture.

For example, when three reference pictures are marked as "used for short-term reference" with PicNum equal to 300, 302, and 303 and two reference pictures 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,

– RefPicList0[4] is set equal to the long-term reference picture with LongTermPicNum = 3.

#### 8.2.2.2.2 Initialisation process for reference picture lists for B slices

This initialisation process is invoked when decoding a B slice in a coded picture.

For purposes of the formation of the reference picture lists RefPicList0 and RefPicList1 the term reference entry refers in the following to decoded reference pictures.

When this process is invoked, there shall be at least one reference entry that is currently marked as "used for reference" (i.e., as "used for short-term reference" or "used for long-term reference") and is not marked as "non-existing". Pictures with higher values of temporal_id than the current picture cannot be used for reference, and are not included in either RefPicList0 or RefPicList1.

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 8.2.3.2 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 modification 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" pictures).

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:

1. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference" and which have a value of temporal_id equal to or lower than the temporal_id of the current picture. 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 ).
2. The long-term reference entries which have a value of temporal_id equal to or lower than the temporal_id of the current picture 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:

1. Let entryShortTerm be a variable ranging over all reference entries that are currently marked as "used for short-term reference" and which have a value of temporal_id equal to or lower than the temporal_id of the current picture. 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 ).
2. Long-term reference entries which have a value of temporal_id equal to or lower than the temporal_id of the current picture are ordered starting with the long-term reference picture that has the lowest LongTermPicNum value and proceeding through in ascending order to the long-term reference entry that has the highest LongTermPicNum value.
3. 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.

### 8.2.2.3 Modification process for reference picture lists

After the invocation of this process, there shall be no reference pictures with greater temporal_id than the current slice included in the output RefPicList0 or RefPicList1.

When ref_pic_list_modification_flag_l0 is equal to 1, the following applies:

1. Let refIdxL0 be an index into the reference picture list RefPicList0. It is initially set equal to 0.
2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.
   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.2.3.1 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.
   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.2.3.2 is invoked with refIdxL0 as input, and the output is assigned to refIdxL0.

– Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList0 is finished.

When the current slice is a B slice and ref_pic_list_modification_flag_l1 is equal to 1, the following applies:

1. Let refIdxL1 be an index into the reference picture list RefPicList1. It is initially set equal to 0.
2. The corresponding syntax elements modification_of_pic_nums_idc are processed in the order they occur in the bitstream. For each of these syntax elements, the following applies.
   – If modification_of_pic_nums_idc is equal to 0 or equal to 1, the process specified in subclause 8.2.2.3.1 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.
   – Otherwise, if modification_of_pic_nums_idc is equal to 2, the process specified in subclause 8.2.2.3.2 is invoked with refIdxL1 as input, and the output is assigned to refIdxL1.
   – Otherwise (modification_of_pic_nums_idc is equal to 3), the modification process for reference picture list RefPicList1 is finished.

Deleted: 8.2.2.3.1

#### 8.2.2.3.1 Modification 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 modification_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      (8-4)
else
    picNumLXNoWrap = picNumLXPred − ( abs_diff_pic_num_minus1 + 1 )
```

– Otherwise (modification_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      (8-5)
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 modification_of_pic_nums_idc equal to 0 or 1 in the ref_pic_list_modification( ) 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 specified by the following pseudo-code:

```
if( picNumLXNoWrap > CurrPicNum )
    picNumLX = picNumLXNoWrap − MaxPicNum                                          (8-6)
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 short-term reference picture with PicNum equal to picNumLX shall not have greater temporal_id than the current slice.

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++ )           (8-7)
    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.

#### 8.2.2.3.2 Modification 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++ )                    (8-8)
    if( LongTermPicNumF( RefPicListX[ cIdx ] ) != long_term_pic_num )
        RefPicListX[ nIdx++ ] = RefPicListX[ cIdx ]
```

where the long-term reference picture with LongTermPicNum equal to long_term_pic_num shall not have greater temporal_id than the current slice, and 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.

#### 8.2.2.4 Mapping process for reference picture lists combination in B slices

[Ed.: (WJ) needs to be checked once again. Try to find better way to represent]

This initialisation process is invoked when decoding a B slice header.

Input to this process are the reference picture list RefPicListX and num_ref_idx_lX_active_minus1 with X being 0 or 1.

Outputs of this process are arrays PredLCToPredLx and RefIdxLCToRefIdxLx.

When the current slice is a B slice and ref_pic_list_modification_flag_lc is equal to 0, the following ordered steps apply:

1. Let refIdxL0 and refIdxL1 be indices into the reference picture lists RefPicListL0 and RefPicListL1. They are initially set equal to 0.
2. Let refIdxLC be an index into PredLCToPredLx and RefIdxLCToRefIdxLx. It is initially set equal to 0.
3. The following process is repeated until refIdxL0 and refIdxL1 are both greater than num_ref_idx_l0_active_minus1 and num_ref_idx_l1_active_minus1, respectively:
   – If refIdxL0 is less than or equal to num_ref_idx_l0_active_minus1,
     – If the entry RefPicListL0[ refIdxL0 ] is the first occurance of the reference picture,

```
PredLCToPredLx[ refIdxLC ] = Pred_L0,                                   (8-9)
RefIdxLCToRefIdxLx[ refIdxLC++ ] = refIdxL0.
```

     – refIdxL0++.

– If refIdxL1 is less than or equal to num_ref_idx_l1_active_minus1 and ref_pic_list_combination_flag equal to 1,

  – If the entry RefPicListL1[ refIdxL1 ] is the first occurance of the reference picture,

$$\begin{aligned} &\text{PredLCToPredLx[ refIdxLC ] = Pred\_L1,} \\ &\text{RefIdxLCToRefIdxLx[ refIdxLC++ ] = refIdxL1.} \end{aligned} \tag{8-10}$$

  – refIdxL1++.

When the current slice is a B slice and ref_pic_list_modification_flag_lc is equal to 1, the following ordered steps apply:

1. Let refIdxLC be an index into the reference picture list PredLCToPredLx and RefIdxLCToRefIdxLx. It is initially set equal to 0.
2. The corresponding syntax elements pic_from_list_0_flag and ref_idx_list_curr are processed in the order they occur in the bitstream. For each of these syntax elements pairs, the following applies.

– If pic_from_list_0_flag is equal to 1,

$$\text{PredLCToPredLx[ refIdxLC ] = Pred\_L0,} \tag{8-11}$$

– Otherwise,

$$\text{PredLCToPredLx[ refIdxLC ] = Pred\_L1} \tag{8-12}$$

– RefIdxLCToRefIdxLx[ refIdxLC++ ] = ref_idx_list_curr

When refIdxLC is greater than num_com_ref_list_active_minus1+ 1, the extra entries past position num_com_ref_list_active_minus1 are discarded from PredLCToPredLx and RefIdxLCToRefIdxLx.

When refIdxLC is less than num_com_ref_list_active_minus1 + 1, the remaining entries in PredLCToPredLx and RefIdxLCToRefIdxLx are set equal to Pred_L0 and 0, respectively.

### 8.2.3 Decoded reference picture marking process

This process is invoked for decoded pictures when nal_ref_idc is not equal to 0.

NOTE 1 – The decoding process for gaps in frame_num that is specified in subclause 8.2.3.2 may also be invoked when nal_ref_idc is equal to 0, as specified in clause 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". A picture that is marked as "used for short-term reference" is identified by its FrameNum. A picture that is marked as "used for long-term reference" is identified by its LongTermFrameIdx.

Pictures 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 picture until the picture is 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 8.2.3.3 or by the adaptive memory control reference picture marking process, a customised adaptive marking operation specified in subclause 8.2.3.4.

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 8.2.2.1 is invoked to specify the assignment of the variables FrameNum, FrameNumWrap, PicNum and LongTermPicNum.

#### 8.2.3.1 Sequence of operations for decoded reference picture marking process

Decoded reference picture marking proceeds in the following ordered steps:

4. All slices of the current picture are decoded.
5. Depending on whether the current picture is an IDR picture, the following applies.

– If the current picture is an IDR picture, the following ordered steps are specified:

a. All reference pictures are marked as "unused for reference"

b. 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 picture 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 8.2.3.3 is invoked.
- Otherwise (adaptive_ref_pic_marking_mode_flag is equal to 1), the process specified in subclause 8.2.3.4 is invoked.

6. 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".

It is a requirement of bitstream conformance that, after marking the current decoded reference picture, the total number of pictures marked as "used for reference" shall not be greater than Max( max_num_ref_frames, 1 ).

#### 8.2.3.2 Decoding process for gaps in frame_num

[Ed. (TW): insert text]

#### 8.2.3.3 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.

1. Let numShortTerm be the total number of reference pictures that are marked as "used for short-term reference". Let numLongTerm be the total number of reference pictures that are marked as "used for long-term reference".
2. When numShortTerm + numLongTerm is equal to Max( max_num_ref_frames, 1 ), the condition that numShortTerm is greater than 0 shall be fulfilled, and the short-term reference picture that has the smallest value of FrameNumWrap is marked as "unused for reference".

#### 8.2.3.4 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 8.2.3.4.1 to 8.2.3.4.6 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.

##### 8.2.3.4.1 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

$$picNumX = CurrPicNum - ( difference\_of\_pic\_nums\_minus1 + 1 ). \quad (8\text{-}13)$$

The value of picNumX is used to mark the corresponding short-term reference picture as "unused for reference".

##### 8.2.3.4.2 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.

The value of LongTermPicNum is used to mark the corresponding long-term reference picture as "unused for reference".

##### 8.2.3.4.3 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 8.2.3.4.1. picNumX shall refer to a picture 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 picture, that picture is marked as "unused for reference".

The value of LongTermFrameIdx is used to mark the corresponding picture from "used for short-term reference" to "used for long-term reference".

#### 8.2.3.4.4 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 picture 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.

#### 8.2.3.4.5 Marking process of all reference pictures as "unused for reference" and setting MaxLongTermFrameIdx to "no long-term picture 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 picture indices".

#### 8.2.3.4.6 Process for assigning a long-term picture 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 picture, that picture 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.

After marking the current decoded reference picture, the total number of pictures that are marked as "used for reference" shall not be greater than Max( 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, 3, or 4.

## 8.3 Decoding process for coding units coded in intra prediction mode

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

A variable nS is set equal to ( 1 << log2CUSize ).

Depending on pcm_flag and IntraSplitFlag, the decoding process for luma samples is specified as follows.

– If pcm_flag is equal to 1, the reconstucted picture is modified as follows:

recSamples$_Y$[ xB + i, yB + j ] =
pcm_sample_luma[ ( nS * j ) + i ] << ( BitDepth$_Y$ – PCMBitDepth$_Y$ ), with i, j = 0..nS-1 (8-14)

– Otherwise (pcm_flag is equal to 0), if IntraSplitFlag is equal to 0, the following ordered steps apply:

1. The derivation process for the intra prediction mode as specified in subclause **Error! Reference source not found.** is invoked with the luma location ( xB, yB ) and the variable log2TrafoSize set equal to log2CUSize as well as IntraPredMode that is previously (in decoding order) derived for adjacent coding units as the input and the output is the variable IntraPredMode[ xB ][ yB ].
2. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the luma location ( xB, yB ), the variable log2TrafoSize set equal to log2CUSize, the variable trafoDepth set equal to 0, the luma intra prediction mode IntraPredMode[ xB ][ yB ] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm_flag is equal to 0 and IntraSplitFlag is equal to 1), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

1. The variable xBS is set equal to xB + ( nS >> 1 ) * ( blkIdx % 2 ).
2. The variable yBS is set equal to yB + ( nS >> 1 ) * ( blkIdx / 2 ).
3. The derivation process for the intra prediction mode as specified in subclause **Error! Reference source not found.** is invoked with the luma location ( xBS, yBS ) and the variable log2TrafoSize set equal to log2CUSize – 1 as well as IntraPredMode that is previously (in decoding order) derived for adjacent coding units as the input and the output is the variable IntraPredMode[ xBS ][ yBS ].
4. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the luma location ( xBS, yBS ), the variable log2TrafoSize set equal to log2CUSize − 1, the variable trafoDepth set equal to 1, the luma intra prediction mode IntraPredMode[ xBS ][ yBS ] and the variable cIdx set equal to 0 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

Depending on pcm_flag, the decoding process for chroma samples is specified as follows:

– If pcm_flag is equal to 1, the reconstucted picture is modified as follows:

recSamples$_{Cb}$[ xB/2 + i, yB/2 + j ] =
pcm_sample_chroma[ ( nS/2 * j ) + i ] << ( BitDepth$_C$ – PCMBitDepth$_C$ ) with i, j = 0..nS/2-1 (8-15)

recSamples$_{Cr}$[ xB/2 + i, yB/2 + j ] =
pcm_sample_chroma[ ( nS/2 * ( j + nS ) ) + i ] << ( BitDepth$_C$ – PCMBitDepth$_C$ ) with i, j = 0..nS/2-1 (8-16)

– Otherwise (pcm_flag is equal to 0), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

1. The derivation process for the chroma intra prediction mode as specified in 8.3.2 is invoked with the luma location ( xB, yB ) as input and the output is the variable IntraPredModeC.
2. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the chroma location ( xB/2, yB/2 ), the variable log2TrafoSize set equal to log2CUSize-1, the variable trafoDepth set equal to 0, the chroma intra prediction mode IntraPredModeC, and the variable cIdx set equal to 1 as the inputs and the output is a modified reconstructed picture before deblocking filtering.
3. The decoding process for intra blocks as specified in subclause 8.3.3 is invoked with the chroma location ( xB/2, yB/2 ), the variable log2TrafoSize set equal to log2CUSize-1, the variable trafoDepth set equal to 0, the chroma intra prediction mode IntraPredModeC, and the variable cIdx set equal to 2 as the inputs and the output is a modified reconstructed picture before deblocking filtering.

### 8.3.1 Derivation process for luma intra prediction mode

Inputs to this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture,

– a variable log2TrafoSize specifying the size of the current prediction unit,

– variable arrays IntraPredMode (If available) that are previously (in decoding order) derived for adjacent coding units.



**Formatted:** Subscript
**Formatted:** Tab stops: 0.79", Left
**Deleted:** 13
**Formatted:** Subscript
**Formatted:** Subscript
**Deleted:** If
**Formatted:** Indent: Left: 0.2", Hanging: 0.3", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28"
**Deleted:** 8.3.1
**Deleted:** (1 << log2CUSize )
**Deleted:** (1 << log2CUSize )
**Deleted:** 8.3.1
**Deleted:** The
**Deleted:** in the following ordered steps
**Formatted:** Tab stops: 0.79", Left
**Deleted:** 13
**Deleted:** 13
**Deleted:** ¶

Output of this process is the variable IntraPredMode[ xB ][ yB ].

Table 8-1 specifies the value for the intra prediction mode and the associated names.

**Table 8-1 – Specification of intra prediction mode and associated names**

| Intra prediction mode | Associated names |
|---|---|
| 0 | Intra_Vertical |
| 1 | Intra_Horizontal |
| 2 | Intra_DC |
| Otherwise (3..33) | Intra_Angular |
| 34 | Intra_Planar |
| 35 | Intra_FromLuma (used only for chroma) |

Table 8-2 specifies the number of luma intra prediction modes intraPredModeNum depending on log2TrafoSize.

**Table 8-2 – Specification of intraPredModeNum**

| log2TrafoSize | intraPredModeNum |
|---|---|
| 2 | 18 |
| 3 | 35 |
| 4 | 35 |
| 5 | 35 |
| 6 | 4 |

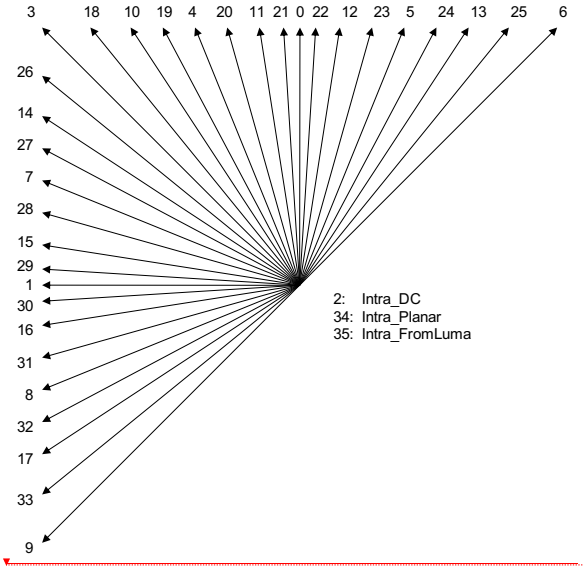
Table 8-3 specifies the mapping table used for converting the number of intra prediction modes.

**Table 8-3 – Specification of mapIntraPredMode3 and mapIntraPredMode9**

| value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **mapIntraPredMode3[ value ]** | 0 | 1 | 2 | 2 | 2 | 0 | 2 | 1 | 1 | 2 | 2 | 0 | 0 | 2 | 2 | 1 | 1 |
| **mapIntraPredMode9[ value ]** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **value** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| **mapIntraPredMode3[ value ]** | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 2 |
| **mapIntraPredMode9[ value ]** | 2 | 3 | 4 | 4 | 0 | 0 | 5 | 5 | 6 | 3 | 7 | 7 | 1 | 1 | 8 | 8 | 2 |

IntraPredMode[ xB ][ yB ] labelled 0, 1, 2, .., 33 represents directions of predictions as illustrated in Figure 8-1.

Deleted: Table 8-5
Deleted: 5
Deleted: ¶
Deleted: Table 8-1
Deleted: 1
Deleted: 8-1
Deleted: 8-1
Deleted: 7
Deleted: 34
Deleted: 34
Deleted: 34
Deleted: 3
Deleted: Table 8-2
Deleted: 2
Deleted: 8-2
Deleted: 8-2
Deleted: mapIntraPredMode5
Deleted: 5
Deleted: 3
Deleted: 4
Deleted: 5
Deleted: 3
Deleted: 4
Deleted: 4
Deleted: Figure 8-1



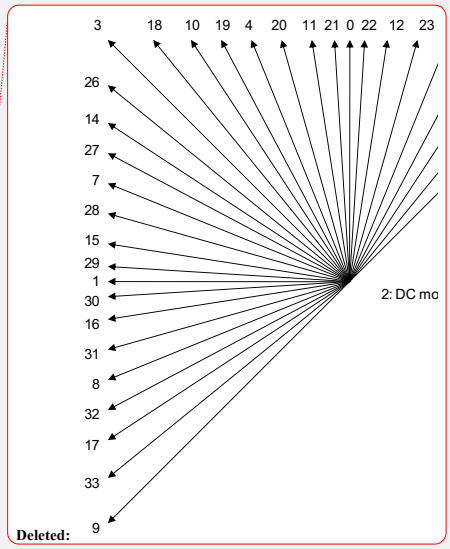




**Figure 8-1 – Intra prediction mode directions (informative)**

IntraPredMode[ xB ][ yB ] is derived as the following ordered steps. [Ed. (WJ): proponent suggests to move this part to the syntax since the other syntax elements utilize IntraPredMode. But it seems too complex to move all the following process to the syntax table. Maybe it's better to move this part to the semantics section or simply avoid the use of IntraPredMode to parse the syntax item]

1. The derivation process for neighbouring treeblocks specified in subclause XXX with ( xB, yB ) given as input and the output is assigned to tbAddrA and tbAddrB specifying the treeblock addresses of treeblocks covering ( xBA, yBA ) and ( xBB, yBB ) respectively where ( xBA, yBA ) is set equal to ( xB-1, yB ) and ( xBB, yBB ) is set equal to ( xB, yB-1 ).
2. For N being either replaced A or B, the variables intraPredModeN are derived as follows.
   - If the treeblock with address tbAddrN is not available, intraPredModeN is set equal to -1
   - Otherwise, if the coding unit covering ( xBN, yBN ) is not coded as intra mode, intraPredModeN is set equal to Intra_DC.
   - Otherwise, intraPredModeN is set equal to IntraPredMode[ xBN ][ yBN ], where IntraPredMode is the variable array assigned to the coding unit covering the luma location ( xBN, yBN ).
3. For N being either replaced A or B, the variables intraPredModeN are further modified to be equal to Intra_DC when intraPredModeN is equal to Intra_Planar.
4. For N being either replaced A or B, the variables candIntraPredModeN are derived as follows.
   - If intraPredModeN is greater than or equal to intraPredModeNum
     - If intraPredModeNum is equal to 4 then candIntraPredModeN is set equal to mapIntraPredMode3[ intraPredModeN ]
     - Otherwise candIntraPredModeN is set equal to mapIntraPredMode9[ intraPredModeN ].
   - Otherwise, candIntraPredModeN is set equal to intraPredModeN
5. The candModeList[x] and NumMPMCand are derived as follows:

**Deleted:** follows

**Deleted:** s

**Formatted:** Indent: Left: 0", First line: 0", Tab stops: 0", Left + Not at 0.2"

**Deleted:** –→

**Formatted:** Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5", Tab stops: 0.49", Left + Not at 0.55"

**Deleted:** –→

**Formatted:** Indent: Left: 0.51", Tab stops: 0.79", Left + Not at 0.49" + 0.75"

**Deleted:** 2

**Deleted:** .

**Formatted:** Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5", Tab stops: 0.2", Left + 0.49", Left + 0.83", Left + 1.1", Left + 1.38", Left + Not at 0.75" + 1" + 2.07"

**Formatted:** (Asian) Korean

**Formatted:** Indent: Left: 0.51", Tab stops: 0.79", Left + Not at 0.49" + 0.75"

**Deleted:** 5

**Deleted:** 3

**Formatted:** Indent: Left: 0.79"

**Deleted:** mapIntraPredMode5

**Deleted:** <#>If intraPredModeNum is equal to 3 then candIntraPredModeN is set equal to mapIntraPredMode3[ intraPredModeN ]¶

**Formatted:** Indent: Left: 0.51", Tab stops: 0.79", Left + Not at 0.49" + 0.75"

**Formatted:** Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5", Tab stops: 0.2", Left + 0.49", Left + 0.83", Left + 1.1", Left + 1.38", Left

– If both candIntraPredModeN are not available, then the value 2 is assigned to candModeList[ 0 ] and NumMPMCand is set equal to 1

– Otherwise, if only one candIntraPredModeN is available or if both candIntraPredModeN are the same, then this candIntraPredModeN is assigned to candModeList[ 0 ] and NumMPMCand is set equal to 1

– Otherwise, both candIntraPredModeN are assigned to the candidate modes list with the smaller of the two candidates at candModeList[ 0 ] and the larger at candModeList[ 1 ] and NumMPMCand is set equal to 2.

6. IntraPredMode[ xB ][ yB ] is derived by applying the following procedure.

– If prev_intra_pred_flag[ xB ][ yB ] is true, the IntraPredMode[ xB ][ yB ] is set equal to candModeList[mpm_idx[ xB ][ yB ]]

– Otherwise IntraPredMode[ xB ][ yB ] is derived by applying the following procedure

– IntraPredMode[ xB ][ yB ] = rem_intra_pred_mode

– for (cIdx = 0; cIdx < NumMPMCand; cIdx++)

if (IntraPredMode[ xB ][ yB ]>= candModeList[cIdx]) IntraPredMode[ xB ][ yB ]++

7. When IntraPredMode[ xB ][ yB ] is equal to Intra_DC and planar_flag_luma is equal to 1, IntraPredMode[ xB ][ yB ] is further modified to be equal to Intra_Planar.

### 8.3.2 Derivation process for chroma intra prediction mode

[Ed.: (WJ) this subclause may be moved to the semantics of intra_chroma_pred_mode syntax]

Input to this process is a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture.

Output of this process is the variable IntraPredModeC.

The chroma intra prediction mode IntraPredModeC is derived as specifed in Table 8-4, or Table 8-5, with intra_chroma_pred_mode, IntraPredMode[ xB ][ yB ] and chroma_pred_from_luma_enabled_flag as inputs.

IntraPredModeC is further modified to be equal to Intra_Planar when IntraPredModeC is equal to Intra_DC and planar_flag_chroma is equal to 1.

**Formatted:** Indent: Left: 0.51", Tab stops: 0.79", Left + Not at 0.49" + 0.75"

**Formatted:** Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5", Tab stops: 0.49", Left + Not at 0.55"

**Formatted:** (Asian) Korean

**Formatted:** Indent: Left: 0.51", Tab stops: 0.79", Left + 2.07", Left + Not at 0.5" + 0.75"

**Formatted:** (Asian) Korean

**Formatted:** Indent: Left: 0.79"

**Deleted:** →→

**Formatted:** Indent: Left: 0.5", Tab stops: 0.2", Left + 0.49", Left + Not at 0.55"

**Formatted:** Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Indent at: 0.5", Tab stops: 0.2", Left + 0.49", Left + Not at 0.55"

**Deleted:** Table 8-3Table 8-3

**Deleted:** Table 8-4

**Deleted:** and

**Table 8-4 – Specification of IntraPredModeC according to the values of intra_chroma_pred_mode and IntraPredMode[ xB ][ yB ] when chroma_pred_from_luma_enabled_flag is equal to 1**

| intra_chroma_pred_mode | IntraPredMode[ xB ][ yB ] | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | X ( 4 <= X < 35 ) |
| 0 | 35 | 35 | 35 | 35 | 35 |
| 1 | n/a | 0 | 0 | 0 | 0 |
| 2 | 1 | n/a | 1 | 1 | 1 |
| 3 | 2 | 2 | 2 | 2 | 2 |
| 4 | 0 | 1 | 2 | 3 | X |

**Table 8-5 – Specification of IntraPredModeC according to the values of intra_chroma_pred_mode and IntraPredMode[ xB ][ yB ] when chroma_pred_from_luma_enabled_flag is equal to 0**

| intra_chroma_pred_mode | IntraPredMode[ xB ][ yB ] | | | | |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | X ( 4 <= X < 35 ) |
| 0 | n/a | 0 | 0 | 0 | 0 |
| 1 | 1 | n/a | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 0 | 1 | 2 | 3 | X |

[Ed. (WJ): is it possible that intra_chroma_pred_mode = 2 and IntraPredMode[ xB ][ yB ] = 2? Since it is actualy overlapped with the case that intra_chroma_pred_mode = 3 (derivation from luma) and IntraPredMode[ xB ][ yB ] = 2]

### 8.3.3 Decoding process for intra blocks

Inputs to this process are:

– a sample location ( xB, yB ) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable log2TrafoSize specifying the size of the current block,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable intraPredMode specifying the intra prediction mode.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is:

– a modified reconstructed picture before deblocking filtering.

Depending split_transform_flag[ xB ][ yB ][ trafoDepth ], the following applies:

– If split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variable xB1 is set equal to xB + ( ( 1 << log2TrafoSize ) >> 1 ).
2. The variable yB1 is set equal to yB + ( ( 1 << log2TrafoSize ) >> 1 ).
3. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
4. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB1, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.
5. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the



Deleted: 3
Deleted: 8-3
Deleted: 8-3
Deleted: 34
Deleted: 0
Deleted: 4
Deleted: 1
Deleted: 4
Deleted: 2
Deleted: 4
Deleted: 3
Deleted: 4
Deleted: X
Deleted: 4
Deleted: 1
Deleted: 2
Deleted: 2
Deleted: 3
Deleted: 3
Deleted: 2
Deleted: n/a
Formatted: Highlight
Deleted: n/a
Deleted: n/a
Deleted: n/a
Deleted: n/a
Deleted: 3
Deleted:
Deleted: 4
Formatted: Caption
Deleted: 4
Deleted: n/a
Formatted: Highlight
Deleted: ¶
Deleted: luma

intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.

6. The decoding process for intra blocks as specified in this subclause is invoked with the location ( xB1, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode intraPredMode, and the variable cIdx as the inputs and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following ordered steps apply:

1. The variable nS is set equal to 1 << log2TrafoSize.
2. The intra sample prediction process as specified in subclause 8.3.3.1 is invoked with the location ( xB, yB ), the intra prediction mode intraPredMode, the prediction size nS and the variable cIdx as the inputs and the output is a (nS)x(nS) array predSamples.
3. The scaling and transformation process as specified in subclause 8.5.1 is invoked with the location ( xB, yB ), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nS as the inputs and the output is a (nS)x(nS) array resSamples.
4. The residual signal accumulation process as specified in subclause XXX is invoked with the variable arraySize set equal to nS, the (nS)x(nS) array predSamples, and the (nS)x(nS) array resSamples as the inputs and the output is a (nS)x(nS) array recSamples.
5. The picture reconstruction process for a component before deblocking filtering as specified in subclause XXX is invoked with the location ( xB, yB ), the variable arraySize set equal to nS, the variable cIdx set equal to 0, and the (nS)x(nS) array recSamples as the inputs and the output is a modified reconstructed picture before deblocking filtering.

#### 8.3.3.1 Intra sample prediction

Inputs to this process are:

– a sample location ( xB, yB ) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable intraPredMode specifying the luma intra prediction mode,

– a variable nS specifying the prediction size.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is:

– the predicted samples predSamples[ x, y ], with x, y =0..nS-1.

The nS*4+1 neighbouring samples p[ x, y ] that are constructed luma samples prior to the deblocking filter process, with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y=-1, are derived as follows.

– The luma location ( xBN, yBN ) is specified by

xBN = xB + x (8-17)

yBN = yB +y (8-18)

– The derivation process for neighbouring locations in subclause XXX is invoked for sample location with ( xBN, yBN ) as input and the treeblock address tbAddrN as output.

– Each sample p[ x, y ] with x = -1, y= -1..nS*2-1 and x = 0, y = -1 is derived as follows

– If any of the following condition is true, the sample p[ x, y ] is marked as “not available for intra prediction”

– tbAddrN is not available

– the coding unit covering ( xBN, yBN ) is not coded as intra mode and constrained_intra_pred_flag is equal to 1

– Otherwise, the sample p[ x, y ] is marked as “available for intra prediction” and the sample at the location ( xBN, yBN ) inside the treeblock tbAddrN is assigned to p[ x, y ]

When intraPredMode is equal to 2, if at least one sample p[ x, y ] with x = -1, y = 0..nS-1 and x = 0..nS-1, y = -1 is marked as “not available for intra prediction” or cIdx is not equal to 0, DCPredFilterFlag is set equal to 0. Otherwise, DCPredFilterFlag is set equal to 1.

Deleted: 14

Deleted: 15

[Ed. (WJ): reference sample subtitution process eliminates the needs to check the availability of samples except DC mode. Maybe it's better to remove this checking even for DC mode for the consistency]

When at least one sample p[ x, y ] with x = 1, y = 1..nS*2 1 and x = 0..nS*2 1, y = 1 is marked as "not available for intra prediction," the reference sample substitution process for intra sample prediction in subclause 8.3.3.1.1 is invoked with the samples p[ x, y ] with x = 1, y = 1..nS*2 1 and x = 0..nS*2 1, y = 1 as input and the modified samples p[ x, y ] with x = 1, y = 1..nS*2 1 and x = 0..nS*2 1, y = 1 as output.

Depending on intraPredMode, the following ordered steps apply:

1. When cIdx is equal to 0, filtering process of neighbouring samples specified in 8.3.3.1.2 is invoked with the sample array p and the prediction size nS as the inputs and the output is reassigned to the sample array p.
2. Intra sample prediction process according to intraPredMode applies as follows:
   - One of the intra prediction modes specified in subclause 8.3.3.1.3 to 8.3.3.1.8 is invoked with the sample array p and the prediction size nS as the inputs and the output is the predicted sample array predSamples according to intraPredMode.

#### 8.3.3.1.1 Reference sample substitution process for intra sample prediction

Inputs to this process are the reference samples p[ x, y ] with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y = -1 for intra sample prediction.

Outputs of this process are the modified reference samples p[ x, y ] with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y = -1 for intra sample prediction.

The values of the samples p[ x, y ] with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y = -1 are modified as follows:

- If all samples p[ x, y ] with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y = -1 are marked as "not available for intra prediction," the value ( 1 << ( BitDepth$_Y$ - 1 ) ) is substituted for the values of all samples p[ x, y ].
- Otherwise (at least one but not all samples p[ x, y ] are marked as "not available for intra prediction"), the following steps are performed for each sample p[ x0, y0 ] with x0 = -1, y0 = -1..nS*2-1 and x0 = 0..nS*2-1, y0 = -1 marked as "not available for intra prediction":
  - Let the variables q and r be initially set to -1.
  - The variable q is modified as follows:
    - If x0 is equal to -1, a reference sample p[ x, y ] marked as "available for intra prediction" is identified by searching sequentially among p[ x, y ] starting from x = -1, y = y0+1 to x = -1, y = nS*2-1. As soon as a sample marked as "available for intra prediction" is found, the search is terminated and the value of p[ x, y ] is assigned to q. [Ed.: (WJ) is there any elegant way rather than searching?]
    - Otherwise (x0 is not equal to -1), a sample marked as "available for intra prediction" is identified among the samples p[ x, y ] by searching sequentially starting from x = x0-1, y = -1 to x = -1, y = -1. When a sample marked as "available for intra prediction" is not found among p[ x, y ] with x = x0-1..-1, y = -1, the search is continued among p[ x, y ] sequentially starting from x = -1, y = 0 to x = -1, y = nS*2-1. As soon as a sample marked as "available for intra prediction" is found, the search is terminated and the value of p[ x, y ] is assigned to q. [Ed.: (WJ) is there any elegant way rather than searching?]
  - The variable r is modified as follows:
    - If x0 is equal to -1, a sample marked as "available for intra prediction" is identified among the samples p[ x, y ] by searching sequentially starting from x = -1, y = y0-1 to x = -1, y = -1. When a sample marked as "available for intra prediction" is not found among p[ x, y ] with x = -1, y = y0-1..-1, the search is continued among p[ x, y ] sequentially starting from x = 0, y = -1 to x = nS*2-1, y = -1. As soon as a sample marked as "available for intra prediction" is found, the search is terminated and the value of p[ x, y ] is assigned to r. [Ed.: (WJ) is there any elegant way rather than searching?]
    - Otherwise (x0 is not equal to -1), a sample marked as "available for intra prediction" is identified among the samples p[ x, y ] by searching sequentially starting from x = x0+1, y = -1 to x = nS*2-1, y = -1. As soon as a sample marked as "available for intra prediction" is found, the search is terminated and the value of p[ x, y ] is assigned to r. [Ed.: (WJ) is there any elegant way rather than searching?]
  - The value of the sample p[ x0, y0 ] is modified as follows:
    - If q is not equal to -1 and r is not equal to -1, the value ( ( q + r + 1 ) >> 1 ) is substituted for the value of p[ x0, y0 ].

**Deleted:** Table 8-4 specifies the value for intraPredMode and the associated names.¶
**Table 8-58-48-4 – Specification of intraPredMode and associated names**¶
**intraPredMode** ... [49]

**Deleted:** a (2*nS+1)x(2*nS+1) filtered sample array pF.

**Deleted:** process Intra_process and Otherwise, if intraPredMode is equal to 34, Intra_FromLuma prediction process specified in subclause 8.3.3.1.8 is invoked with the sample location ( xB, yB ), the sample array p and the prediction size nS as the inputs and the output is the predicted sample array predSamples.¶
<#>¶
O

**Deleted:** 8.3.3.1.2

**Deleted:** 8.3.3.1.6

**Formatted:** Indent: Left: 0.49", Hanging: 0.3", Bulleted + Level: 2 + Aligned at: 0.75" + Tab after: 1" + Indent at: 1", Tab stops: 0.79", Left + Not at 0.75"

**Deleted:** F

**Deleted:** .

**Formatted:** Heading 5, No bullets or numbering, Tab stops: Not at 0.75" + 1" + 2.07"

– Otherwise, if q is not equal to -1, the value of q is substituted for the value of p[ x0, y0 ].

– Otherwise (q is equal to -1 and r is not equal to -1), the value of r is substituted for the value of p[ x0, y0 ].

All samples p[ x, y ] with x = -1, y = -1..nS*2-1 and x = 0..nS*2-1, y = -1 are marked as “available for intra prediction.”

#### 8.3.3.1.2 Filtering process of neighbouring samples

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size.

Output of this process is:

– filtered samples pF[ x, y ],. with x, y = -1..2*nS-1.

**Table 8-6 – Specification of intraFilterType[ nS ][ IntraPredMode ] for various prediction unit sizes**

| IntraPredMode | intraFilterType for nS = 4 | intraFilterType for nS = 8 | intraFilterType for nS = 16 | intraFilterType for nS = 32 | intraFilterType for nS = 64 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 1 | 1 | 0 |
| 6 | 1 | 1 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 0 | 1 | 1 | 1 | 0 |
| 9 | 1 | 1 | 1 | 1 | 0 |
| 10 | 0 | 0 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 | 0 |
| 12 | 0 | 0 | 1 | 1 | 0 |
| 13 | 0 | 0 | 1 | 1 | 0 |
| 14 | 0 | 0 | 1 | 1 | 0 |
| 15 | 0 | 0 | 1 | 1 | 0 |
| 16 | 0 | 0 | 1 | 1 | 0 |
| 17 | 0 | 0 | 1 | 1 | 0 |
| 18-33 | 0 | 0 | 0 | 1 | 0 |
| 34 | 0 | 0 | 0 | 0 | 0 |
| 35 | n/a | n/a | n/a | n/a | n/a |

Filtered sample array pF[ x, y ] with x = -1..nS*2-1 and y = -1..nS*2-1 are derived as follows:

– When intraFilterType[ nS ][ IntraPredMode ] is equal to 1, the following applies:

pF[ -1, nS*2-1 ] = p[ -1, nS*2-1 ] (8-19)

pF[ nS*2-1, -1 ] = p[ nS*2-1, -1 ] (8-20)

Deleted: 8-5
Deleted: 8-5
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: 2
Deleted: If
Deleted: 17
Deleted: 18

pF[ -1, y ] = ( p[ -1, y+1 ] + 2*p[ -1, y ] + p[ -1, y-1 ] + 2 ) >> 2 for y = nS*2-2..0 (8-21)

pF[ -1, -1] = ( p[ -1, 0 ] + 2*p[ -1, -1] + p[ 0, -1 ] + 2) >> 2 (8-22)

pF[ x, -1 ] = ( p[ x-1, -1 ] + 2*p[ x, -1 ] + p[ x+1, -1 ] + 2 ) >> 2 for x = 0..nS*2-2 (8-23)

#### 8.3.3.1.3 Specification of Intra_Vertical prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is equal to 0.

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1, are derived by

predSamples[ x, y ] = p[ x, -1 ], with x, y = 0..nS-1 (8-24)

#### 8.3.3.1.4 Specification of Intra_Horizontal prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is equal to 1.

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1, are derived by

predSamples[ x, y ] = p[ -1, y ], with x, y = 0..nS-1 (8-25)

#### 8.3.3.1.5 Specification of Intra_DC prediction mode

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction block size.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is equal to 2.

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1, are derived as the following ordered steps:

1. A variable DCVal is derived as:

$$DCVal = \left( \sum_{x'=0}^{nS-1} p[x', -1] + \sum_{y'=0}^{nS-1} p[-1, y'] + nS \right) >> (k+1), \text{ with x, y = 0..nS-1} \quad (8\text{-}26)$$

where $k=\log_2(nS)$

2. Depending on the prediction block size nS, the following applies.

– If DCPredFilterFlag is equal to 1 and nS is less than 32, the following applies.

– If nS is equal to 4, the prediction samples predSamples[ x, y ] are derived as

predSamples[ 0, 0 ] = ( 3*p[ -1, 0 ] + 2*DCVal + 3*p[ 0, -1 ] + 4 ) >> 3 (8-27)

predSamples[ x, 0 ] = ( 3*p[ x, -1 ] + 5*DCVal + 4 ) >> 3, with x = 1..nS-1 (8-28)



Deleted: 19

Deleted: 20

Deleted: 21

Deleted: ¶
→Otherwise, intraFilterType[ nS ][ IntraPredMode ] is equal to 2, the following applies:¶
pF[ -1, nS*2-1 ] = p[ -1, nS*2-1 ]→→(8-22)¶
pF[ nS*2-1, -1 ] = p[ nS*2-1, -1 ]→→(8-23)¶
pF1[ -1, y ] = ( p[ -1, y+1 ] + 2*p[ -1, y ] + p[ -1, y-1 ] + 2 ) >> 2 for y = nS*2-2..0→(8-24)¶
pF1[ -1, -1] = ( p[ -1, 0 ] + 2*p[ -1, -1] + p[ 0, -1 ] + 2) >> 2→(8-25)¶
pF1[ x, -1 ] = ( p[ x-1, -1 ] + 2*p[ x, -1 ] + p[ x+1, -1 ] + 2 ) >> 2 for x = 0..nS*2-2→(8-26)¶
pF[ -1, y ] = ( pF1[ -1, y+1 ] + 2*pF1[ -1, y ] + pF1[ -1, y-1 ] + 2 ) >> 2 for y = nS*2-2..0→(8-27)¶
pF[ -1, -1] = ( pF1 [ -1, 0 ] + 2*pF1 [ -1, -1] + pF1 [ 0, -1 ] + 2) >> 2→(8-28)¶
pF[ x, -1 ] = ( pF1 [ x-1, -1 ] + 2* pF1 [ x, -1 ] + pF1 [ x+1, -1 ] + 2 ) >> 2 for x = 0..nS*2-2→(8-29)

Deleted: 30

Deleted: 31

Deleted: ,

Deleted: .

Deleted: → the bit-depth of the chroma component, bitDepth.¶

Deleted: ¶

Deleted: follows.

Deleted: → If all samples p[ x, -1 ], with x=0..nS-1 and p[ -1, y ], with y=0..nS-1 are marked as "available for intra prediction," the values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1 are derived as the following ordered steps:¶

Deleted: by¶
predSamples[ x, y ]

Deleted: 32

Formatted: Font: 10 pt

Formatted: Normal, Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + 1.38", List tab + Not at 0.28" + 0.59" + 0.79" + 0.98"

Formatted: English (UK)

Deleted: 32

Formatted: Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"

Deleted: 32

predSamples[ 0, y ] = ( 3*p[ -1, y ] + 5*DCVal + 4 ) >> 3, with y = 1..nS-1 (8-29)
predSamples[ x, y ] = DCVal, with x, y = 1..nS-1 (8-30)

– Otherwise, if nS is equal to 8, the prediction samples predSamples[ x, y ] are derived as

predSamples[ 0, 0 ] = ( 1*p[ -1, 0 ] + 2*DCVal + 1*p[ 0, -1 ] + 2 ) >> 2 (8-31)
predSamples[ x, 0 ] = ( 1*p[ x, -1 ] + 3*DCVal + 2 ) >> 2, with x = 1..nS-1 (8-32)
predSamples[ 0, y ] = ( 1*p[ -1, y ] + 3*DCVal + 2 ) >> 2, with y = 1..nS-1 (8-33)
predSamples[ x, y ] = DCVal, with x, y = 1..nS-1 (8-34)

– Otherwise, if nS is equal to 16, the prediction samples predSamples[ x, y ] are derived as

predSamples[ 0, 0 ] = ( 1*p[ -1, 0 ] + 6*DCVal + 1*p[ 0, -1 ] + 4 ) >> 3 (8-35)
predSamples[ x, 0 ] = ( 1*p[ x, -1 ] + 7*DCVal + 4 ) >> 3, with x = 1..nS-1 (8-36)
predSamples[ 0, y ] = ( 1*p[ -1, y ] + 7*DCVal + 4 ) >> 3, with y = 1..nS-1 (8-37)
predSamples[ x, y ] = DCVal, with x, y = 1..nS-1 (8-38)

– Otherwise, the prediction samples predSamples[ x, y ] are derived as

predSamples[ x, y ] = DCVal, with x, y = 0..nS-1 (8-39)

#### 8.3.3.1.6 Specification of Intra_Angular prediction mode

[Ed: (WJ) provided text from JCTVC-C042 was modified. It still needs to be improved]

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is in the range of 3..33.

Table 8-7 specifies the mapping table between intraPredMode and the rearranged intra prediction order intraPredOrder.

**Table 8-7 – Specification of intraPredOrder**

| **intraPredMode** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **intraPredOrder** | - | - | - | 1 | 5 | 13 | 17 | 21 | 29 | 33 | 3 | 7 | 11 | 15 | 19 | 23 | 27 |
| **intraPredMode** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| **intraPredOrder** | 31 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |

Figure 8-2 illustrates the total 33 intra angles and Table 8-8 specifies the mapping table between intraPredOrder and the angle parameter intraPredAngle.



**Deleted:** 32)↵ ... [50]
**Formatted:** Font: 10 pt, English (UK)
**Formatted:** English (UK)
**Formatted:** English (UK)
**Formatted:** Normal, Indent: Left: 0.73", Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"
**Deleted:** 32
**Formatted** ... [51]
**Formatted:** Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"
**Formatted** ... [52]
**Deleted:** 32
**Deleted:** 32
**Formatted** ... [53]
**Formatted:** English (UK)
**Formatted** ... [54]
**Formatted:** Normal, Indent: Left: 0.73", Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"
**Deleted:** 32
**Formatted** ... [55]
**Formatted:** Indent: Left: 0.98", Tab stops: Not at 0.59" + 0.79"
**Deleted:** 32
**Formatted** ... [56]
**Deleted:** 32
**Formatted** ... [57]
**Deleted:** 32
**Formatted** ... [58]
**Formatted:** English (UK)
**Formatted:** Normal, Bulleted + Level: 1 + Aligned at: 0.49" + Indent at: 0.74", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.59" + 0.79" + 0.98"
**Deleted:** 32
**Deleted:** ¶
**Deleted:** ¶
**Deleted:** –→ Otherwise, If all samples p[ x, -1 ], with x=0..nS-1 are marked as "not available for intra prediction" and all samples p[ -1, y ], with y=0..nS-1 are marked as "available for intra prediction," the values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1 are derived by¶
predSamples[ x, y ] = $\left( \sum_{y'=0}^{nS-1} p[-1, y'] + (nS >> 1) \right) >> k$,
with x, y = 0..nS-1→ (8-33)↵
where k=log2(nS)¶
–→ Otherwise, If all samples p[ x, -1 ], with x=0..nS-1 are marked as "available for intra prediction" and all samples p[ -1, y ], with y=0..nS-1 are marked as "not available for intra prediction," the values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1 are derived by¶ ... [59]
**Formatted:** Indent: Left: 0.59"
**Deleted:** [Ed: (WJ) current software implementation seems ... [60]
**Deleted:** not equal to 0, 1, and 2
**Deleted:** Table 8-6
**Deleted:** 8-6
**Deleted:** 8-6
**Deleted:** Figure 8-2… illustrates the total 33 intra angles and ... [61]

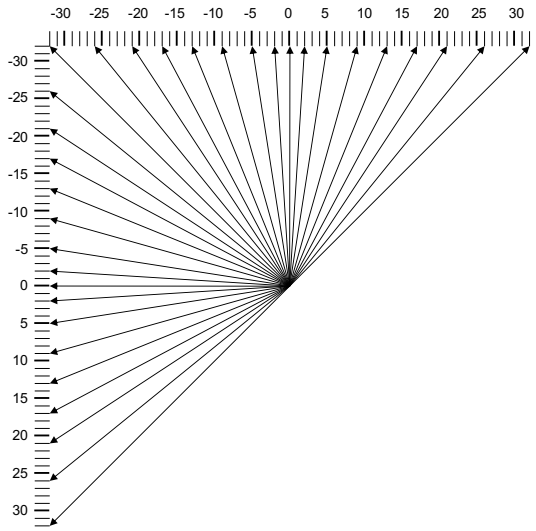



**Figure 8-2 – Intra prediction angle definition (informative)**

**Table 8-8 – Specification of intraPredAngle**

| **intraPredOrder** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **intraPredAngle** | - | -32 | -26 | -21 | -17 | -13 | -9 | -5 | -2 | - | 2 | 5 | 9 | 13 | 17 | 21 | 26 |
| **intraPredOrder** | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
| **intraPredAngle** | 32 | -26 | -21 | -17 | -13 | -9 | -5 | -2 | - | 2 | 5 | 9 | 13 | 17 | 21 | 26 | 32 |

Table 8-9 further specifies the mapping table between intraPredOrder and the inverse angle parameter invAngle.

**Table 8-9 – Specification of invAngle**

| **intraPredOrder** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| **invAngle** | -256 | -315 | -390 | -482 | -630 | -910 | -1638 | -4096 |
| **intraPredOrder** | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| **invAngle** | -315 | -390 | -482 | -630 | -910 | -1638 | -4096 | - |

The reference pixel array refMain[ x ], with x=-nS..2*nS is specified as follows.

– If intraPredOrder is less than 18,

refMain[ x ] = p[ -1+x, -1 ], with x=0..nS (8-40)

– If intraPredAngle is less than 0,

refMain[ x ] = p[ -1, -1+( ( x*invAngle+128 )>>8 ) ], with x=( nS*intraPredAngle ) >>5..-1 (8-41)

– Otherwise,

Deleted: 8-7
Deleted: 8-7
Deleted: Table 8-8
Deleted: 8-8
Deleted: 8-8
Deleted: 36
Deleted: 37

refMain[ x ] = p[ -1+x, -1 ], with x=nS+1..2*nS (8-42)

Otherwise,

refMain[ x ] = p[ -1, -1+x ], with x=0..nS (8-43)

– If intraPredAngle is less than 0,

refMain[ x ] = p[ -1+( ( x*invAngle+128 )>>8 ), -1 ], with x=( nS*intraPredAngle ) >>5..-1 (8-44)

– Otherwise,

refMain[ x ] = p[ -1, -1+x ], with x=nS+1..2*nS (8-45)

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1 are derived by the following procedures.

– The index variable iIdx and the multiplication factor iFact are derived by

iIdx = ( ( y + 1 )*intraPredAngle ) >> 5 (8-46)

iFact = ( ( y + 1 )*intraPredAngle ) && 31 (8-47)

– Depending on the value of iFact, the following applies.

– If iFact is not equal to 0, the value of the prediction samples predSamples[ x, y ] is derived by

predSamples[ x, y ] = ( ( 32 – iFact )*refMain[ x+iIdx+1 ] + iFact*refMain[ x+iIdx+2] + 16 ) >> 5 (8-48)

– Otherwise, the value of the prediction samples predSamples[ x, y ] is derived by

predSamples[ x, y ] = refMain[ x+iIdx+1 ] (8-49)

**8.3.3.1.7 Specification of Intra_Planar prediction mode**

Inputs to this process are:

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size,

– the bit-depth of the chroma component, bitDepth.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is equal to 34.

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1, are derived by

predSamples[ x, y ] = (
( nS – 1 – x ) * p[ -1, y ] + ( x + 1 ) * p[ nS – 1 ,-1 ] +
( nS – 1 – y ) * p[ x ,-1 ] + ( y + 1 ) * p[ -1, nS – 1 ] + nS ) >> ( k + 1 ) (8-50)

with x, y = 0..nS-1 where k = log2( nS )

**8.3.3.1.8 Specification of Intra_FromLuma prediction mode**

Inputs to this process are:

– a sample location ( xB, yB ) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– neighbouring samples p[ x, y ], with x, y = -1..2*nS-1,

– a variable nS specifying the prediction size.

Output of this process is:

– predicted samples predSamples[ x, y ], with x, y =0..nS-1.

This intra prediction mode is invoked when intraPredMode is equal to 35.

Deleted: 38

Deleted: 39

Deleted: 40

Deleted: 41

Deleted: 42

Deleted: 43

Deleted: 44

Deleted: 45

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

Deleted: The seedHor[ y ] and seedVer[ x ] arrays with x, y = 0..nS-1 are initialized as follows.¶
<#>If all samples p[ x, -1 ], with x=0..nS-1 and p[ -1, y ], with y=0..nS-1 are marked as “available for intra prediction,” the initialization is done as seedHor[ y ] = p[ -1, y ] and seedVer[ x ] = p[ x, -1 ].¶
<#>Otherwise, If all samples p[ x, -1 ], with x=0..nS-1 are marked as “not available for intra prediction” and all samples p[ -1, y ], with y=0..nS-1 are marked as “available for intra prediction,” the initialization is done as seedHor[ y ] = p[ -1, y ] and seedVer[ x ] = p[ -1, 0 ].¶
<#>Otherwise, If all samples p[ x, -1 ], with x=0..nS-1 are marked as “available for intra prediction” and all samples p[ -1, y ], with y=0..nS-1 are marked as “not available for intra prediction,” the initialization is done as seedHor[ y ] = p[ 0, -1 ] and seedVer[ x ] = p[ x, -1 ].¶
<#>Otherwise, the initialization is done as seedHor[ y ] = 1 << ( bitDepth – 1 ) and seedVer[ x ] = 1 << ( bitDepth – 1 ).¶

Deleted: seedHor

Deleted: seedVer

Deleted: seedVer

Deleted: seedHor

Deleted: 45

Formatted: Subscript

Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

Deleted: 4

The values of the prediction samples predSamples[ x, y ], with x, y = 0..nS-1, are derived as the following ordered steps:

1. Variable k3 and the sample array $p_Y$' are derived as:

$$k3 = \mathrm{Max}( 0, \mathrm{BitDepth}_C + \log_2( nS ) - 14 ) \quad (8\text{-}51)$$

$$p_Y'[ x, y ] = ( \mathrm{recSamples}_Y[ 2x, 2y ] + \mathrm{recSamples}_Y[ 2x, 2y+1 ] ) >> 1, \text{ with } x, y = -1..2*nS \quad (8\text{-}52)$$

2. Variables L, C, LL, LC and k2 are derived as follows:

$$L = \left( \sum_{y=0}^{nS-1} p_Y'[-1, y] + \sum_{x=0}^{nS-1} p_Y'[x, -1] \right) >> k3 \quad (8\text{-}53)$$

$$C = \left( \sum_{y=0}^{nS-1} p[-1, y] + \sum_{x=0}^{nS-1} p[x, -1] \right) >> k3 \quad (8\text{-}54)$$

$$LL = \left( \sum_{y=0}^{nS-1} p_Y'[-1, y]^2 + \sum_{x=0}^{nS-1} p_Y'[x, -1]^2 \right) >> k3 \quad (8\text{-}55)$$

$$LC = \left( \sum_{y=0}^{nS-1} p_Y'[-1, y] * p[-1, y] + \sum_{y=0}^{nS-1} p_Y'[x, -1] * p[x,-1] \right) >> k3 \quad (8\text{-}56)$$

$$k2 = \log_2( (2*nS) >> k3 ) \quad (8\text{-}57)$$

3. Variables a and b are derived as:

$$a1 = ( LC << k2 ) - L*C \quad (8\text{-}58)$$
$$a2 = ( LL << k2 ) - L*L \quad (8\text{-}59)$$
$$k = \mathrm{Max}( 0, \log_2( \mathrm{abs}( a2 ) ) - 5 ) - \mathrm{Max}( 0, \log_2( \mathrm{abs}( a1 ) ) - 14 ) + 2 \quad (8\text{-}60)$$
$$a1s = a1 >> \mathrm{Max}(0, \log_2( \mathrm{abs}( a1 ) ) - 14 ) \quad (8\text{-}61)$$
$$a2s = \mathrm{abs}( a2 >> \mathrm{Max}(0, \log_2( \mathrm{abs}( a2 ) ) - 5 ) ) + 1 \quad (8\text{-}62)$$

[Ed. (WJ): bug in software – a2s should be abs( a2 >> Max(0, $\log_2$( abs( a2 ) ) – 5 ) ) – without '+1']

$$a = \mathrm{Clip3}( -2^{15}, 2^{15}-1, a1s*\mathrm{lmDiv} + ( 1 << ( k1 - 1 ) ) >> k1 ) \quad (8\text{-}63)$$
$$b = ( L - ( ( a*C ) >> k1 ) + ( 1 << ( k2 - 1 ) ) ) >> k2 \quad (8\text{-}64)$$

where lmDiv is specified in Table 8-10 with the input a2s.

4. The values of the prediction samples predSamples[ x, y ] are derived as:

$$predSamples[ x, y ] = \mathrm{Clip1}_C( ( ( p_Y'[ x, y ] * a ) >> 13 ) + b ), \text{ with } x, y = 0..nS-1 \quad (8\text{-}65)$$



**Formatted:** Subscript
**Formatted:** Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" + 1.1" + 1.38"
**Deleted:** 30
**Formatted:** Subscript
**Formatted:** Subscript
**Deleted:** 30
**Formatted:** Subscript
**Formatted:** Subscript
**Formatted:** Subscript
**Formatted** ... [62]
**Deleted:** <#>If all samples p[ x, -1 ], with x = 0..nS-1 and p[ ... [63]
**Deleted:** 30
**Formatted:** Font: 10 pt
**Deleted:** 30
**Formatted:** Font: 10 pt
**Deleted:** 30
**Formatted:** Font: 10 pt
**Deleted:** 30
**Formatted:** Font: 10 pt
**Deleted:** 30
**Formatted:** Font: 10 pt
**Deleted:** 30
**Formatted:** Font: 10 pt
**Formatted:** Font: 10 pt, Subscript
**Formatted:** Font: 10 pt
**Formatted:** Font: 10 pt
**Formatted:** Font: 10 pt
**Deleted:** <#>Otherwise, if all samples p[ x, -1 ], with x = 0.. ... [64]
**Deleted:** 30
**Formatted** ... [65]
**Deleted:** 30
**Deleted:** 30
**Formatted:** Font: 10 pt, Subscript
**Formatted:** Font: 10 pt, Subscript
**Deleted:** 30
**Formatted:** Font: 10 pt, Subscript
**Deleted:** 30
**Formatted:** Font: 10 pt, Subscript
**Formatted:** Font: 10 pt, English (US)
**Deleted:** 30
**Formatted:** Font: 10 pt, Superscript
**Formatted:** Font: 10 pt, Superscript
**Deleted:** 30
**Formatted:** Font: 10 pt
**Formatted** ... [66]
**Deleted:** 30
**Formatted:** Subscript
**Formatted:** Subscript
**Formatted** ... [67]

**Table 8-10 – Specification of lmDiv**

| a2s | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lmDiv | 32768 | 16384 | 10923 | 8192 | 6554 | 5461 | 4681 | 4096 | 3641 | 3277 | 2979 | 2731 | 2521 |
| a2s | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| lmDiv | 2341 | 2185 | 2048 | 1928 | 1820 | 1725 | 1638 | 1560 | 1489 | 1425 | 1365 | 1311 | 1260 |
| a2s | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| lmDiv | 1214 | 1170 | 1130 | 1092 | 1057 | 1024 | 993 | 964 | 936 | 910 | 886 | 862 | 840 |
| a2s | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 |
| lmDiv | 819 | 799 | 780 | 762 | 745 | 728 | 712 | 697 | 683 | 669 | 655 | 643 | 630 |
| a2s | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | |
| lmDiv | 618 | 607 | 596 | 585 | 575 | 565 | 555 | 546 | 537 | 529 | 520 | 512 | |

Formatted: Caption

Formatted: (Asian) Korean

Formatted: Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

Deleted: ¶

## 8.4 Decoding process for coding units coded in inter prediction mode

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

Output of this process is a modified reconstructed picture before deblocking filtering.

The variable $nCS_L$ is set equal to 1 << log2CUSize and the variable $nCS_C$ is set equal to ( 1 << log2CUSize ) >> 1.

Decoding process for coding units coded in inter prediction mode consists of following ordered steps:

1. The inter prediction process as specified in subclause 8.4.1 is invoked with the luma location ( xC, yC ), coding unit size log2CUSize, inter prediction mode PredMode, and prediction partition mode PartMode as the inputs and the outputs are 3 arrays $predSamples_L$, $predSamples_{Cb}$, $predSamples_{Cr}$.
2. The decoding process for the residual signal of coding units coded in inter prediction mode specified in subclause 8.4.3 is invoked with the luma location ( xC, yC ), the size of the current coding unit log2CUSize as inputs and the outputs are 3 arrays $resSamples_L$, $resSamples_{Cb}$, $resSamples_{Cr}$.
3. The residual signal accumulation process as specified in subclause XXXX is invoked with arrays $predSamples_L$, $predSamples_{Cb}$, $predSamples_{Cr}$, and the arrays $resSamples_L$, $resSamples_{Cb}$, $resSamples_{Cr}$ as the inputs and the outputs are 3 arrays $recSamples_L$, $recSamples_{Cb}$, $recSamples_{Cr}$. [Ed: (WJ) it may be split per each color component. Revisit after writing residual signal accumulation process]
4. The picture reconstruction process for a component before deblocking filtering as specified in subclause XXXX is invoked with the luma location ( xB, yB ), and 3 arrays recSamplesL, recSamplesCb, recSamplesCr as the inputs and the output is a modified reconstructed picture before deblocking filtering. [Ed: (WJ) it may be split per each color component. Revisit after writing picture reconstruction process]

### 8.4.1 Inter prediction process

This process is invoked when decoding coding unit whose PredMode is not equal to MODE_INTRA.

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit,

– a variable PredMode specifying prediction mode of current coding unit,

– a variable PartMode specifying prediction partition mode of current coding unit.

Outputs of this process are:

– a $(nCS_L)x(nCS_L)$ array $predSamples_L$ of luma prediction samples, where $nCS_L$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCS_C$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCS_C$ is derived as specified below.

The variable $nCS_L$ is set equal to 1 << log2CUSize and the variable $nCS_C$ is set equal to ( 1 << log2CUSize ) >> 1. [Ed: (WJ) revisit for supporting other chroma formats]

The variable $nCS1_L$ is set equal to $nCS_L$ >> 1.

Depending on PartMode, the following applies: [Ed: (WJ) is PUType better?]

– If PartMode is equal to PART_2Nx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS_L$, the height of the luma prediction samples nPSH set equal to $nCS_L$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

– Otherwise, if PartMode is equal to PART_2NxN, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, $nCS1_L$ ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ and a partition index PartIdx set equal to 1 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

Otherwise, if PartMode is equal to PART_Nx2N, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS_L$ and a partition index PartIdx set equal to 0 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( $nCS1_L$, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS_L$ and a partition index PartIdx set equal to 1 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

Otherwise, if PartMode is equal to PART_NxN, the following ordered steps apply:

1. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ as and a partition index PartIdx set equal to 0 inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

2. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( $nCS1_L$, 0 ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ and a partition index PartIdx set equal to 1 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

3. The decoding process for prediction units in inter prediction mode as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, $nCS1_L$ ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ and a partition index PartIdx set equal to 2 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

4. The decoding process for inter prediction units as specified in subclause 8.4.2 is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( $nCS1_L$, $nCS1_L$ ), the size of the coding unit $nCS_L$, the width of the luma prediction samples nPSW set equal to $nCS1_L$, the height of the luma prediction samples nPSH set equal to $nCS1_L$ and a partition index PartIdx set equal to 3 as inputs, and the outputs are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cb}$ and $predSamples_{Cr}$.

### 8.4.2 Decoding process for prediction units in inter prediction mode

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current prediction unit relative to the top left luma sample of the current coding unit,

– a variable nCS specifying the size of the current coding unit,

– a variable nPSW specifying the width of the current prediction unit,

– a variable nPSH specifying the width of the current prediction unit,

– a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

– a ($nCS_L$)x($nCS_L$) array $predSamples_L$ of luma prediction samples, where $nCS_L$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCS_C$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCS_C$ is derived as specified below.

The variable $nCS_L$ is set equal to nCS and the variable $nCS_C$ is set equal to nCS >> 1. [Ed: (WJ) revisit for supporting other chroma formats]

The decoding process for prediction units in inter prediction mode consists of the following ordered steps:

1. Derivation process for motion vector components and reference indices as specified in subclause 8.4.2.1.

   Inputs to this process are

   – a luma location ( xC, yC ) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

   – a luma location ( xB, yB ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,

   – variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,

   – a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

   Outputs of this process are

   – luma motion vectors mvL0 and mvL1, and chroma motion vectors mvCL0 and mvCL1,

   – reference indices refIdxL0 and refIdxL1,

   – prediction list utilization flags predFlagL0 and predFlagL1.

2. Decoding process for inter sample prediction as specified in subclause 8.4.2.2.

   Inputs to this process are

   – a luma location ( xC, yC ) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

   – a luma location ( xB, yB ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,

   – a variable nCS specifying the size of the current coding unit,

   – variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.

– luma motion vectors mvL0 and mvL1, and chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and prefFlagL1.

Outputs of this process are

– inter prediction samples (predSamples); which are a ($nCS_L$)x($nCS_L$) array $predSamples_L$ of prediction luma samples and two ($nCS_C$)x($nCS_C$) arrays $predSamples_{Cr}$, and $predSamples_{Cr}$ 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[ xB, yB ] = mvL0 (8-66)
MvL1[ xB, yB ] = mvL1 (8-67)

RefIdxL0[ xB, yB ] = refIdxL0 (8-68)
RefIdxL1[ xB, yB ] = refIdxL1 (8-69)

PredFlagL0[ xB, yB ] = predFlagL0 (8-70)
PredFlagL1[ xB, yB ] = predFlagL1 (8-71)

Deleted: 46
Deleted: 47
Deleted: 48
Deleted: 49
Deleted: 50
Deleted: 51

#### 8.4.2.1 Derivation process for motion vector components and reference indices

Input to this process are

– a luma location ( xC, yC ) of the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a luma location ( xB, yB ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,

– variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,

– a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

– luma motion vectors mvL0 and mvL1 and chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags predFlagL0 and predFlagL1.

Let ( xP, yP ) specify the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture where xP = xC + xB and yP = yC + yB.

For the derivation of the variables mvL0 and mvL1, refIdxL0 and refIdxL1 as well as PredFlagL0 and PredFlagL1, the following applies.

– If PredMode is equal to MODE_SKIP, the derivation process for luma motion vectors for merge mode as specified in subclause 8.4.2.1.1 is invoked with the luma location ( xP, yP ), variables nPSW, nPSH and the partition index PartIdx as inputs and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1.

– Otherwise, if PredMode is equal to MODE_INTER and merge_flag[ xP ][ yP ] is equal to 1,, the derivation process for luma motion vectors for merge mode as specified in subclause 8.4.2.1.1 is invoked with the luma location ( xP, yP ), variables nPSW and nPSH and the partition index PartIdx as inputs and the outputs being the luma motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, 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 LcToLx, refIdxLX and predFlagLX are derived as follows.

   - If inter_pred_flag[ xP ][ yP ] is equal to Pred_LC and PredLCToPredLx[ ref_idx_lc[ xP ][ yP ] ] is equal to Pred_LX,

     refIdxLX = RefIdxLCToRefIdxLx[ ref_idx_lc[ xP ][ yP ] ] (8-72)
     predFlagLX = 1 (8-73)

Deleted: 8.4.2.1.3
Deleted: 8.4.2.1.3
Deleted: 52
Deleted: 53

mvd_lX[ xP ][ yP ][ 0 ] = mvd_lc[ xP ][ yP ][ 0 ] (8-74)
mvd_lX[ xP ][ yP ][ 1 ] = mvd_lc[ xP ][ yP ][ 1 ] (8-75)
mvp_idx_lX[ xP ][ yP ][ 0 ] = mvp_idx_lc[ xP ][ yP ] (8-76)
LcToLx = LX (8-77)

- Otherwise, if inter_pred_flag[ xP ][ yP ] is equal to Pred_LX or Pred_BI,

refIdxLX = ref_idx_lX[ xP ][ yP ] (8-78)
predFlagLX = 1 (8-79)

- Otherwise, the variables refIdxLX and predFlagLX are specified by

refIdxLX = -1 (8-80)
predFlagLX = 0 (8-81)

2. The variable mvdLX is derived as follows.

mvdLX[ 0 ] = mvd_lX[ xP ][ yP ][ 0 ] (8-82)
mvdLX[ 1 ] = mvd_lX[ xP ][ yP ][ 1 ] (8-83)

3. When predFlagLX is equal to 1, the variable mvpLX is derived as follows.

- The derivation process for luma motion vector prediction in subclause 8.4.2.1.3 is invoked with the luma location ( xP, yP ), variables nPSW and nPSH and refIdxLX as the inputs and the output being mvpLX.

4. When predFlagLX is equal to 1, the luma motion vector mvLX is derived as

mvLX[ 0 ] = mvpLX[ 0 ] + mvdLX[ 0 ] (8-84)
mvLX[ 1 ] = mvpLX[ 1 ] + mvdLX[ 1 ] (8-85)

When ChromaArrayType is not equal to 0 and predFlagLX (with X being either 0 or 1) is equal to 1, the derivation process for chroma motion vectors in subclause 8.4.2.1.7 is invoked with mvLX and refIdxLX as inputs and the output being mvCLX.

#### 8.4.2.1.1 Derivation process for luma motion vectors for merge mode

This process is only invoked when PredMode is equal to MODE_ INTER and merge_flag [ xP ][ yP ] is equal to 1, where ( xP, yP ) specify the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture.

Inputs of this process are

– a luma location ( xP, yP ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,

– variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,

– a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are

– the luma motion vectors mvL0 and mvL1,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags predFlagL0 and predFlagL1.

The motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction utilization flags predFlagL0 and predFlagL1 are derived as specified by the following ordered steps:

1. The derivation process for merging candidates from neighboring prediction unit partitions in subclause 8.4.2.1.4 is invoked with luma location ( xP, yP ), the width and the height of the prediction unit nPSW and nPSH and the partition index PartIdx as inputs and the output is assigned to the availability flags availableFlagN, the motion vectors mvL0N and mvL1N, the reference indices refIdxL0N and refIdxL1N and the prediction list utilization flags predFlagL0N and predFlagL1N with N being replaced by A, B, C or D.

2. The derivation process of reference indices for temporal merging candidate in subclause 8.4.2.1.3 is invoked with luma location ( xP, yP ), nPSW, nPSH as the inputs and the output is directly assigned to refIdxLX.



Deleted: 54
Deleted: 55
Deleted: 56
Deleted: 57
Deleted: 58
Deleted: 59
Deleted: 60
Deleted: 61
Deleted: 62
Deleted: 63
Deleted: 8.4.2.1.5
Deleted: 64
Deleted: 65
Deleted: 8.4.2.1.8
Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
Deleted: 8.4.2.1.4

3. The derivation process for temporal luma motion vector prediction in subclause 4 is invoked with luma location ( xP, yP ), refIdxLX as the inputs and with the output being the availability flag availableFlagLXCol and the temporal motion vector mvLXCol. The variables availableFlagCol and predFlagLXCol (with X being 0 or 1, respectively) are derived as specified below.

   availableFlagCol = availableFlagL0Col || availableFlagL1Col (8-86)

   predFlagLXCol = availableFlagLXCol (8-87)

4. The merging candidate list, mergeCandList, is constructed of which elements are given as specified order:

   1. A, if availableFlagA is equal to 1
   2. B, if availableFlagB is equal to 1
   3. Col, if availableFlagCol is equal to 1
   4. C, if availableFlagC is equal to 1
   5. D, if availableFlagD is equal to 1

5. If several merging candidates have the motion vectors and the same reference indices, the merging candidates are removed from the list except the merging candidate which has the smallest order in the mergeCandList.

6. If the number of elements NumMergeCand within the mergeCandList is equal to 1, mergeIdx is set equal to 0, otherwise, mergeIdx is set equal to merge_idx[ xP][ yP ].

7. The following assignments are made with N being the candidate at position mergeIdx in the merging candidate list mergeCandList ( N = mergeCandList[ mergeIdx ] ) and X being replaced by 0 or 1:

   mvLX[ 0 ] = mvLXN[ 0 ] (8-88)

   mvLX[ 1 ] = mvLXN[ 1 ] (8-89)

   refIdxLX = refIdxLXN (8-90)

   predFlagLX = predFlagLXN (8-91)

8. If all availability flags availableFlagN (with N being replaced by A, B, Col, C, or D) are equal to 0, mergeIdx is set equal to 0 and the variables mvLX, refIdxLX and predFlagLX (with X being replaced by 0 or 1) are inferred as follows.

   If slice_type is equal to P, the following applies.

   mvLX[ 0 ] = 0 (8-92)

   mvLX[ 1 ] = 0 (8-93)

   refIdxL0 = 0 (8-94)

   refIdxL1 = -1 (8-95)

   predFlagL0 = 1 (8-96)

   prefFlagL1 = 0 (8-97)

   Otherwise ( slice_type is equal to B ), the following applies.

   mvLX[ 0 ] = 0 (8-98)

   mvLX[ 1 ] = 0 (8-99)

   refIdxL0 = 0 (8-100)

   refIdxL1 = 0 (8-101)

   predFlagL0 = 1 (8-102)

   prefFlagL1 = 1 (8-103)

Deleted: 8.4.2.1.7
Deleted: 71
Deleted: 72
Deleted: 73
Deleted: 74
Deleted: 75
Deleted: 76
Deleted: 77
Deleted: 78
Deleted: 79
Deleted: 80
Deleted: 81
Deleted: 82
Deleted: 83
Deleted: 84
Deleted: 85
Deleted: 86
Deleted: 87
Deleted: 88

#### 8.4.2.1.2 Derivation process for spatial merging candidates

Inputs to this process are

– a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,

– variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,

– a variable PartIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are (with N being replaced by A, B, C, or D and with X being replaced by 0 or 1)

– the availability flags availableFlagN of the neighbouring prediction units,

– the reference indices refIdxLXN of the neighbouring prediction units,

– the prediction list utilization flags predFlagLXN of the neighbouring prediction units,

– the motion vectors mvLXN of the neighbouring prediction units.

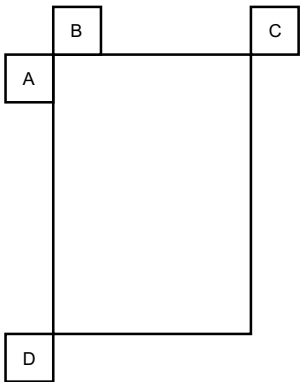



**Figure 8-3 – Spatial neighbours that can be used as merging candidates (informative) illustrates the position of the spatial neighbours A, B, C and D relative to the current prediction unit.**

For the derivation of availableFlagN, with N being A, B, C or D and ( xN, yN ) being ( xP – 1, yP ), ( xP, yP – 1 ), ( xP + nPSW, yP – 1 ) or ( xP – 1, yP+nPSH ) the following applies.

– If one of the following conditions is true, the availableFlagN is set equal to 0, both components mvLXN are set equal to 0, refIdxLXN and predFlagLX[ xN, yN ] of the prediction unit covering luma location ( xN, yN ) are assigned respectively to mvLXN, refIdxLXN and predFlagLXN.

  – The prediction unit covering luma location ( xN, yN ) is not available or PredMode is MODE_INTRA.

  – PartMode of the current prediction unit is PART_2NxN and PartIdx is equal to 1 and the prediction units covering luma location ( xP, yP – 1 ) ( PartIdx = 0 ) and luma location ( xN, yN ) (Cand. N) have identical motion parameters:

    – mvLX[ xP, yP – 1 ] = = mvLX[ xN, yN ]

    – refIdxLX[ xP, yP – 1 ] = = refIdxLX[ xN, yN ]

    – predFlagLX[ xP, yP – 1 ] = = predFlagLX[ xN, yN ]

  – PartMode of the current prediction unit is PART_Nx2N and PartIdx is equal to 1 and the prediction units covering luma location ( xP – 1, yP ) ( PartIdx = 0 ) and luma location ( xN, yN ) (Cand. N) have identical motion parameters:

    – mvLX[ xP – 1, yP ] = = mvLX[ xN, yN ]

– refIdxLX[ xP – 1, yP ] == refIdxLX[ xN, yN ]

– predFlagLX[ xP – 1, yP ] == predFlagLX[ xN, yN ]

– PartMode of the current prediction unit is PART_NxN and PartIdx is equal to 3 and the prediction units covering luma location ( xP – 1, yP ) ( PartIdx = 2 ) and luma location ( xP – 1, yP – 1 ) ( PartIdx = 0 ) have identical motion parameters:

– mvLX[ xP – 1, yP ] == mvLX[ xP – 1, yP – 1 ]

– refIdxLX[ xP – 1, yP ] == refIdxLX[ xP – 1, yP – 1 ]

– predFlagLX[ xP – 1, yP ] == predFlagLX[ xP – 1, yP – 1 ]

and the prediction units covering luma location ( xP, yP – 1 ) ( PartIdx = 1 ) and luma location ( xN, yN ) (Cand. N) have identical motion parameters:

– mvLX[ xP, yP – 1 ] == mvLX[ xN, yN ]

– refIdxLX[ xP, yP – 1 ] == refIdxLX[ xN, yN ]

– predFlagLX[ xP, yP – 1 ] == predFlagLX[ xN, yN ]

– PartMode of the current prediction unit is PART_NxN and PartIdx is equal to 3 and the prediction units covering luma location ( xP, yP – 1 ) ( PartIdx = 1 ) and luma location ( xP – 1, yP – 1 ) ( PartIdx = 0 ) have identical motion parameters:

– mvLX[ xP, yP – 1 ] == mvLX[ xP – 1, yP – 1 ]

– refIdxLX[ xP, yP – 1 ] == refIdxLX[ xP – 1, yP – 1 ]

– predFlagLX[ xP, yP – 1 ] == predFlagLX[ xP – 1, yP – 1 ]

and the prediction units covering luma location ( xP – 1, yP ) ( PartIdx = 2 ) and luma location ( xN, yN ) (Cand. N) have identical motion parameters:

– mvLX[ xP – 1, yP ] == mvLX[ xN, yN ]

– refIdxLX[ xP – 1, yP ] == refIdxLX[ xN, yN ]

– predFlagLX[ xP – 1, yP ] == predFlagLX[ xN, yN ]

– Otherwise, availableFlagN is set equal to 1 and the variables mvLX[ xN, yN ], refIdxLX[ xN, yN ] and predFlagLX[ xN, yN ] of the prediction unit covering luma location ( xN, yN ) are assigned respectively to mvLXN, refIdxLXN and predFlagLXN.

#### 8.4.2.1.3 Derivation process of reference indices for temporal merging candidate

Inputs to this process are

– a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,

– variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.

Outputs of this process are (with X being replaced by 0 or 1)

– the reference indices refIdxLX of the current prediction units.

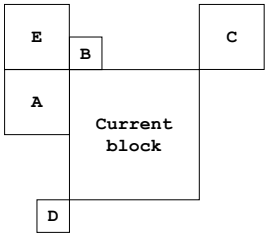



**Figure 8-4 – Spatial neighbours, that can be used to derive reference indices for temporal merging candidates, (informative) illustrates the position of the spatial neighbours A, B, C, D and E relative to the current prediction unit.**

For the derivation of refIdxLXA, the following applies.

– If the prediction unit covering luma location ( xP-1, yP ) is available and PredMode is not MODE_INTRA, refIdxLXA is assigned to refIdxLX[ xP-1, yP ].

– Otherwise, refIdxLXA is assigned to -1.

For the derivation of refIdxLXB, the following applies.

– If the prediction unit covering luma location ( xP, yP-1 ) is available and PredMode is not MODE_INTRA, refIdxLXB is assigned to refIdxLX[ xP, yP-1 ].

– Otherwise, refIdxLXB is assigned to -1.

For the derivation of refIdxLXC, the following applies.

– If the prediction unit covering luma location ( xP+nPSW, yP-1 ) is available and PredMode is not MODE_INTRA, refIdxLXC is assigned to refIdxLX[ xP+nPSW, yP-1 ].

– Otherwise if the prediction unit covering luma location ( xP-1, yP+nPSH ) is available and PredMode is not MODE_INTRA, refIdxLXC is assigned to refIdxLX[ xP-1, yP+nPSH ].

– Otherwise if the prediction unit covering luma location ( xP-1, yP-1 ) is available and PredMode is not MODE_INTRA refIdxLXC is assigned to refIdxLX[ xP-1, yP-1 ].

– Otherwise, refIdxLXC is assigned to -1.

The variable refIdxLX is derived as follows.

– If refIdxLXA is equal to refIdxLXB and refIdxLXB is equal to refIdxLXC, the following applies.
  – If refIdxLXA is equal to -1 then refIdxLX = 0
  – Otherwise refIdxLX = refIdxLXA
– Otherwise if refIdxLXA is equal to refIdxLXB, the following applies.
  – If refIdxLXA is equal to -1 then refIdxLX = refIdxLXC
  – Otherwise refIdxLX = refIdxLXA
– Otherwise if refIdxLXB is equal to refIdxLXC, the following applies.
  – If refIdxLXB is equal to -1 then refIdxLX = refIdxLXA
  – Otherwise refIdxLX = refIdxLXB
– Otherwise if refIdxLXA is equal to refIdxLXC, the following applies.
  – If refIdxLXA is equal to -1 then refIdxLX = refIdxLXB
  – Otherwise refIdxLX = refIdxLXA
– Otherwise if refIdxLXA is equal to -1, the following applies.
  – refIdxLX = min( refIdxLXB, refIdxLXC )
– Otherwise if refIdxLXB is equal to -1, the following applies.

  - refIdxLX = Min( refIdxLXA, refIdxLXC )
- Otherwise if refIdxLXC is equal to -1, the following applies.
  - refIdxLX = Min( refIdxLXA, refIdxLXB )
- Otherwise, the following applies.
  - refIdxLX = Min( Min( refIdxLXA, refIdxLXB ), refIdxLXC )

#### 8.4.2.1.4 Derivation process for luma motion vector prediction

Inputs to this process are

- a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH.
- the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Output of this process is

- the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The motion vector predictor mvpLX is derived in the following ordered steps.

1. The derivation process for motion vector predictor candidates from neighboring prediction unit partitions in subclause 8.4.2.1.5 is invoked with luma location ( xP, yP ), the width and the height of the prediction unit nPSW and nPSH, and refIdxLX (with X being 0 or 1, respectively) as inputs and the availability flags availableFlagLXN and the motion vectors mvLXN with N being replaced by A, B as the output.
2. The derivation process for temporal luma motion vector prediction in subclause 4 is invoked with luma location ( xP, yP ) , the width and the height of the prediction unit nPSW and nPSH, and refIdxLX (with X being 0 or 1, respectively) as the inputs and with the output being the availability flag availableFlagLXCol and the temporal motion vector predictor mvLXCol.
3. The motion vector predictor list, mvpListLX, is constructed of which elements are given as specified order:
    1. mvLXCol, if availableFlagLXCol is equal to 1
    2. mvLXA, if availableFlagLXA is equal to 1
    3. mvLXB, if availableFlagLXB is equal to 1
4. If several motion vectors have the same value, the motion vectors are removed from the list except the motion vector which has the smallest order in the mvpListLX.
5. If the number of elements NumMVPCand( LX ) within the mvpListLX is equal to 1, mvpIdx is set equal to 0, otherwise, mvpIdx is set equal to mvp_idx_lX[ xP, yP ].
6. The motion vector of mvListLX[ mvpIdx ] is assigned to mvpLX.

#### 8.4.2.1.5 Derivation process for motion vector predictor candidates

Inputs to this process are

- a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,
- variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,
- the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are (with N being replaced by A,or B)

- the motion vectors mvLXN of the neighbouring prediction units,
- the availability flags availableFlagLXN of the neighbouring prediction units.

Deleted: 8.4.2.1.6

Deleted: 8.4.2.1.7

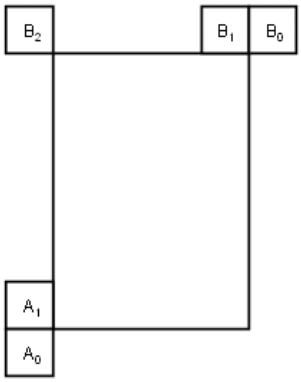



**Figure 8-5 – Spatial motion vector neighbours**

The function RefPicOrderCnt( pic, refidx, LX ) is specified by the value of PicOrderCnt of the picture that is the reference picture RefPicListX[ refidx ] of pic with X being 0 or 1. PicOrderCnt of the reference picture shall be maintained until the picture is marked as "non-exisiting."

The motion vector mvLXA and the availability flag availableFlagLXA are derived in the following ordered steps:

1. Let a set of two sample locations be ($xA_k$, $yA_k$), with k = 0 .. 1, specifies sample locations with $xA_k = xP - 1$, $yA_0 = yP + nPSH$ and $yA_1 = yA_0$ - MinPuSize. The set of sample locations ( $xA_k$, $yA_k$ ) represent the sample locations immediately to the left side of the left partition boundary and it's extended line. [Ed.: (WJ) MinPuSize should be defined somewhere]
2. Let the availability flag availableFlagLXA be initially set equal to 0 and the both components of mvLXA are set equal to 0.
3. For ( $xA_k$, $yA_k$ ) from ( $xA_0$, $yA_0$ ) to ( $xA_1$, $yA_1$ ) where $yA_1 = yA_0$ - MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:
   - When the prediction unit covering luma location ( $xA_k$, $yA_k$ ) is available, PredMode is not MODE_INTRA, predFlagLX[ $xA_k$ ][ $yA_k$ ] is equal to 1 and the reference index refIdxLX[ $xA_k$ ][ $yA_k$ ] is equal to the reference index of the current prediction unit refIdxLX, availableFlagLXA is set equal to 1 and the motion vector mvLXA is set equal to the motion vector mvLX[ $xA_k$ ][ $yA_k$ ], refIdxA is set equal to refIdxLX[ $xA_k$ ][ $yA_k$ ] and ListA is set equal to LX.
4. When availableFlagLXA is equal to 0, for ( $xA_k$, $yA_k$ ) from ( $xA_0$, $yA_0$ ) to ( $xA_1$, $yA_1$ ) where $yA_1 = yA_0$ - MinPuSize, the following applies repeatedly until availableFlagLXA is equal to 1:
   - If the prediction unit covering luma location ( xAk, yAk ) is available, PredMode is not MODE_INTRA, predFlagLY[ $xA_k$ ][ $yA_k$ ] (with Y = !X) is equal to 1 and RefPicOrderCnt( currPic, refIdxLY[ $xA_k$ ][ $yA_k$ ], LY) is equal to RefPicOrderCnt( currPic, refIdxLX, LX), availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ $xA_k$ ][ $yA_k$ ], refIdxA is set equal to refIdxLY[ $xA_k$ ][ $yA_k$ ] and ListA is set equal to LY.
   - Otherwise if the prediction unit covering luma location ( $xA_k$, $yA_k$ ) is available, PredMode is not MODE_INTRA, predFlagLX[ $xA_k$ ][ $yA_k$ ] is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLX[ $xA_k$ ][ $yA_k$ ], refIdxA is set equal to refIdxLX[ $xA_k$ ][ $yA_k$ ], ListA is set equal to LX.
   - Otherwise if the prediction unit covering luma location ( $xA_k$, $yA_k$ ) is available, PredMode is not MODE_INTRA, predFlagLY[ $xA_k$ ][ $yA_k$ ] (with Y = !X) is equal to 1, availableFlagLXA is set equal to 1, the motion vector mvLXA is set equal to the motion vector mvLY[ $xA_k$ ][ $yA_k$ ], refIdxA is set equal to refIdxLY[ $xA_k$ ][ $yA_k$ ], ListA is set equal to LY.

5. When availableFlagLXA is equal to 1, the following applies.

   – If RefPicOrderCnt( currPic , refIdxA, ListA ) is equal to RefPicOrderCnt( currPic, refIdxLX, LX ), mvLXA is set equal to mvLXA

   – Otherwise, mvLXA is derived as specified below

$$tx = ( 16384 + Abs( td / 2 ) ) / td \quad (8\text{-}104)$$

$$DistScaleFactor = Clip3( -1024, 1023, ( tb * tx + 32 ) >> 6 ) \quad (8\text{-}105)$$

$$mvLXA = ClipMv( ( DistScaleFactor * mvLXA + 128 ) >> 8 ) \quad (8\text{-}106)$$

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

where td and tb are derived as

$$td = Clip3( -128, 127, PicOrderCnt( currPic ) – RefPicOrderCnt( currPic , refIdxA, ListA ) ) \quad (8\text{-}107)$$

$$tb = Clip3( -128, 127, PicOrderCnt( currPic ) – RefPicOrderCnt( currPic, refIdxLX, LX ) ) \quad (8\text{-}108)$$

The motion vector mvLXB and the availability flag availableFlagLXB are derived in the following ordered steps:

1. Let a set of three sample location ($xB_k$, $yB_k$), with k = 0,1,2, specifies sample locations with $xB_0 = xP + nPSW$, $xB_1 = xB_0$ MinPuSize , $xB_2 = xP$ - MinPuSize and $yB_k = yP – 1$. The set of sample locations ( $xB_k$, $yB_k$ ) represent the sample locations immediately to the upper side of the above partition boundary and it's extended line. [Ed.: (WJ) MinPuSize should be defined somewhere]
2. Let the availability flag availableFlagLXB be initially set equal to 0 and the both components of mvLXB are set equal to 0.
3. For ( $xB_k$, $yB_k$ ) from ( $xB_0$, $yB_0$ ) to ( $xB_2$, $yB_2$ ) where $xB_0 = xP + nPSW$, $xB_1 = xB_0$ - MinPuSize , and $xB_2 =$ xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:
   – When the prediction unit covering luma location ( $xB_k$, $yB_k$ ) is available, PredMode is not MODE_INTRA, predFlagLX[ $xB_k$ ][ $yB_k$ ] is equal to 1, and the reference index refIdxLX[ $xB_k$ ][ $yB_k$ ] is equal to the reference index of the current prediction unit refIdxLX and mvLX[ $xB_k$ ][ $yB_k$ ] is not identical to mvLXA, availableFlagLXB is set equal to 1 and the motion vector mvLXB is set equal to the motion vector mvLX[ $xB_k$ ][ $yB_k$ ], refIdxB is set equal to refIdxLX[ $xB_k$ ][ $yB_k$ ] and ListB is set equal to LX.
4. When availableFlagLXB is equal to 0, for ( $xB_k$, $yB_k$ ) from ( $xB_0$, $yB_0$ ) to ( $xB_2$, $yB_2$ ) where $xB_0 = xP + nPSW$, $xB_1 = xB_0$ - MinPuSize , and $xB_2 =$ xP - MinPuSize, the following applies repeatedly until availableFlagLXB is equal to 1:
   – If the prediction unit covering luma location ( $xB_k$, $yB_k$ ) is available, PredMode is not MODE_INTRA, predFlagLY[ $xB_k$ ][ $yB_k$ ](with Y = !X) is equal to 1 and RefPicOrderCnt( currPic , refIdxLY[ $xB_k$ ][ $yB_k$ ], LY) is equal to RefPicOrderCnt( currPic, refIdxLX, LX ), availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ $xB_k$ ][ $yB_k$ ], refIdxB is set equal to refIdxLY[ $xB_k$ ][ $yB_k$ ] and ListB is set equal to LY.
   – Otherwise if the prediction unit covering luma location ( $xB_k$, $yB_k$ ) is available, PredMode is not MODE_INTRA, predFlagLX[ $xB_k$ ][ $yB_k$ ] is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLX[ $xB_k$ ][ $yB_k$ ], refIdxB is set equal to refIdxLX[ $xB_k$ ][ $yB_k$ ], ListB is set equal to LX.
   – Otherwise if the prediction unit covering luma location ( $xB_k$, $yB_k$ ) is available, PredMode is not MODE_INTRA, predFlagLY[ $xB_k$ ][ $yB_k$ ] (with Y = !X) is equal to 1, availableFlagLXB is set equal to 1, the motion vector mvLXB is set equal to the motion vector mvLY[ $xB_k$ ][ $yB_k$ ], refIdxB is set equal to refIdxLY[ $xB_k$ ][ $yB_k$ ], ListB is set equal to LY.
   – If availableFlagLXB is equal to 1 and RefPicOrderCnt( currPic , refIdxB, ListB ) is equal to RefPicOrderCnt( currPic, refIdxLX, LX ), mvLXB is set equal to mvLXB.
   – Otherwise if availableFlagLXB is equal to 1 and RefPicOrderCnt( currPic , refIdxB, ListB ) is not equal to RefPicOrderCnt( currPic, refIdxLX, LX ),, mvLXB is derived as specified below.

tx = ( 16384 + Abs( td / 2 ) ) / td (8-109)

DistScaleFactor = Clip3( -1024, 1023, ( tb * tx + 32 ) >> 6 ) (8-110)

mvLXB = ClipMv( ( DistScaleFactor * mvLXB + 128 ) >> 8 ) (8-111)

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

where td and tb are derived as

td = Clip3( -128, 127, PicOrderCnt( currPic ) – RefPicOrderCnt( currPic , refIdxB, ListB ) ) (8-112)

tb = Clip3( -128, 127, PicOrderCnt( currPic ) – RefPicOrderCnt( currPic, refIdxLX, LX ) ) (8-113)

– When availableFlagLXB is equal to 1 and mvLXB is identical to mvLXA, availableFlagLXB is set equal to 0.

#### 8.4.2.1.6 Derivation process for temporal luma motion vector prediction

Inputs to this process are

– a luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture,

– variables specifying the width and the height of the prediction unit for luma, nPSW and nPSH,

– the reference index of the current prediction unit partition refIdxLX (with X being 0 or 1).

Outputs of this process are

– the motion vector prediction mvLXCol,

– the availability flag availableFlagLXCol.

The function RefPicOrderCnt( pic, refidx, LX ) is specified by the value of PicOrderCnt of the picture that is the reference picture RefPicListX[ refidx ] of pic with X being 0 or 1. PicOrderCnt of the reference picture shall be maintained until the picture is marked as “non-exisiting.”

Depending on the values of slice_type and collocated_from_l0_flag, the variable colPic, specifying the picture that contains the co-located partition, is derived as follows.

– If slice_type is equal to B and collocated_from_l0_flag is equal to 0, the variable colPic specifies the picture that contains the co-located partition as specified by RefPicList1[ 0 ].

– Otherwise (slice_type is equal to B and collocated_from_l0_flag is equal to 1 or slice_type is equal to P) , the variable colPic specifies the picture that contains the co-located partition as specified by RefPicList0[ 0 ].

Variable colPu and its position ( xPCol, yPCol ) are derived in the following ordered steps:

1. Right-bottom luma position of the current prediction unit is defined by

   xPRb = xP + nPSW (8-114)

   yPRb = yP + nPSH (8-115)

2. The variable colPu is set as the prediction unit covering the modified position given by ( ( xPRb >> 4 ) << 4, ( yPRb >> 4 ) << 4 ) inside the colPic.

3. If colPu is coded in an intra prediction mode or colPu is unavailable, the following applies.

   – Central luma position of the current prediction unit is defined by

   xPCtr = ( xP + ( nPSW >> 1 ) – 1 (8-116)

   yPCtr = ( yP + ( nPSH >> 1 ) – 1 (8-117)

Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 77
Deleted: 78

– The variable colPu is set as the prediction unit covering the modified position given by ( ( xPCtr >> 4 ) << 4, ( yPCtr >> 4 ) << 4 ) inside the colPic.

4. ( xPCol, yPCol ) is set equal to the top-left luma sample of the colPu relative to the top-left luma sample of the colPic.

The variables mvLXCol and availableFlagLXCol are derived as follows.

– If colPu is coded in an intra prediction mode or colPu is unavailable, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise (colPu is not coded in an intra prediction mode and colPu is available), the variables mvCol and refIdxCol are derived as follows,

– If PredFlagL0[ xPCol ][ yPCol ] is equal to 0, the motion vector mvCol and the reference index refIdxCol are set equal to MvL1[ xPCol ][ yPCol ] and RefIdxL1[ xPCol ][ yPCol ], respectively.

– Otherwise (PredFlagL0[ xPCol ][ yPCol ] is equal to 1), the following applies.

  – If PredFlagL1[ xPCol ][ yPCol ] is equal to 0, the motion vector mvCol and the reference index refIdxCol are set equal to MvL0[ xPCol ][ yPCol ] and RefIdxL0[ xPCol ][ yPCol ], respectively.

  – Otherwise (PredFlagL1[ xPCol ][ yPCol ] is equal to 1), the following applies.

    a. The following asignments are made with X being 0 or 1.

      – RefIdxColLX = RefIdxLX[ xPCol ][ yPCol ]

      – If PicOrderCnt( colPic ) is less than PicOrderCnt( currPic ) and RefPicOrderCnt( colPic, RefIdxColLX, LX ) is greater than PicOrderCnt( currPic ) or PicOrderCnt( colPic ) is greater than PicOrderCnt( currPic ) and RefPicOrderCnt( colPic, RefIdxColLX, LX ) is less than PicOrderCnt( currPic ), the variable MvXCross is derived as follows.

        – MvXCross = 1

      – Otherwise (PicOrderCnt( colPic ) is less than PicOrderCnt( currPic ) and RefPicOrderCnt( colPic, RefIdxColLX, LX ) is less than or equal to PicOrderCnt( currPic ) or PicOrderCnt( colPic ) is greater than PicOrderCnt( currPic ) and RefPicOrderCnt( colPic, RefIdxColLX, LX ) is greater than or equal to PicOrderCnt( currPic )), the variable MvXCross is derived as follows.

        – MvXCross = 0

    b. If one of the following conditions is true, the motion vector mvCol, the reference index refIdxCol and ListCol are set equal to MvL1[ xPCol ][ yPCol ], RefIdxColL1 and L1, respectively.

      – Mv0Cross is equal to 0 and Mv1Cross is equal to 1

      – Mv0Cross is equal to Mv1Cross and reference index list is equal to L1

    c. Otherwise, the motion vector mvCol, the reference index refIdxCol and ListCol are set equal to MvL0[ xPCol ][ yPCol ], RefIdxColL0 and L0, respectively.

and the variable availableFlagLXCol is set equal to 1 and the following applies.

– If PicOrderCnt( colPic ) – RefPicOrderCnt( colPic, refIdxCol, ListCol ) is equal to PicOrderCnt( currPic ) - RefPicOrderCnt( currPic, refIdxLX, LX ),

mvLXCol = mvCol (8-118)

– Otherwise, mvLXCol is derived as scaled version of the motion vector mvCol as specified below

tx = ( 16384 + Abs( td / 2 ) ) / td (8-119)

DistScaleFactor = Clip3( -1024, 1023, ( tb * tx + 32 ) >> 6 ) (8-120)

mvLXCol = ClipMv( ( DistScaleFactor * mvCol + 128 ) >> 8 ) (8-121)

[Ed. (WJ): software has a clip function for scaled motion vector. Do we need this? AVC does not have it]

Deleted: 79

Deleted: 80

Deleted: 81

Deleted: 82

where td and tb are derived as

$$td = Clip3( -128, 127, PicOrderCnt( colPic ) – RefPicOrderCnt( colPic, refIdxCol, ListCol ) ) \quad (8\text{-}122)$$

Deleted: 83

$$tb = Clip3( -128, 127, PicOrderCnt( currPic ) – RefPicOrderCnt( currPic, refIdxLX, LX ) ) \quad (8\text{-}123)$$

Deleted: 84

#### 8.4.2.1.7 Derivation process for chroma motion vectors

[Ed.: (WJ) 4:2:0 assumption yet]

Inputs to this process are a luma motion vector mvLX and a reference index refIdLX.

Output of this process is a chroma motion vector mvCLX.

A chroma motion vector is derived from the corresponding luma motion vector.

For the derivation of the chroma motion vector mvCLX, the following applies.

$$mvCLX[ 0 ] = mvLX[ 0 ] \quad (8\text{-}124)$$

Deleted: 85

$$mvCLX[ 1 ] = mvLX[ 1 ] \quad (8\text{-}125)$$

Deleted: 86

### 8.4.2.2 Decoding process for inter prediction samples

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current coding unit,

– a variable nCS specifying the size of the current coding unit,

– variables specifying the width and the height of the prediction unit, nPSW and nPSH,

– luma motion vectors mvL0 and mvL1, and chroma motion vectors mvCL0 and mvCL1,

– reference indices refIdxL0 and refIdxL1,

– prediction list utilization flags, predFlagL0 and predFlagL1.

Outputs of this process are:

– a ($nCS_L$)x($nCS_L$) array $predSamples_L$ of luma prediction samples, where $nCS_L$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $preSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCS_C$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $predSamples_{Cr}$ of chroma residual samples for the component Cr, where $nCS_C$ is derived as specified below.

The variable $nCS_L$ is set equal to nCS and the variable $nCS_C$ is set equal to nCS >> 1. [Ed: (WJ) revisit for supporting other chroma formats]

Let $predSamplesL0_L$ and $predSamplesL1_L$ be (nPSW)x(nPSH) arrays of predicted luma sample values and $predSampleL0_{Cb}$, $predSampleL1_{Cb}$, $predSampleL0_{Cr}$, and $predSampleL1_{Cr}$ be (nPSW/2)x(nPSH/2) arrays of predicted chroma sample values.

For LX being replaced by either L0 or L1 in the variables predFlagLX, RefPicListX, refIdxLX, refPicLX, and 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 two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.4.2.2.1 with refIdxLX and RefPicListX given as input.

– The arrays $predSamplesLX_L$, $predSamplesLX_{Cb}$, and $predSamplesLX_{Cr}$ are derived by invoking the fractional sample interpolation process specified in subclause 8.4.2.2.2 with the luma locations ( xC, yC ), ( xB, yB ), the width an the

height of the current prediction unit nPSW, nPSH, the motion vectors mvLX, mvCLX, and the reference arrays with $refPicLX_L$, $refPicLX_{Cb}$ and $refPicLX_{Cr}$ given as input.

The array $predSample_L$ of the prediction samples of luma component is derived by invoking the weighted sample prediction process specified in subclause 8.4.2.2.3 with the luma location ( xB, yB ), the width an the height of the current prediction unit nPSW, nPSH, and the sample arrays $predSamplesL0_L$ and $predSamplesL1_L$ as well as predFlagL0, predFlagL1 and $BitDepth_Y$ given as input.

For C being replaced by Cb, or Cr, the array $predSample_C$ of the prediction samples of component C is derived by invoking the weighted sample prediction process specified in subclause 8.4.2.2.3 with the chroma location ( xB/2, yB/2 ), the width an the height of the current prediction unit $nPSW_C$ set equal to nPSW/2, $nPSH_C$ set equal to nPSH/2, and the sample arrays $predSamplesL0_C$ and $predSamplesL1_C$ as well as predFlagL0, predFlagL1 and $BitDepth_C$ given as input.

#### 8.4.2.2.1 Reference picture selection process

[Ed: (WJ) same as AVC]

#### 8.4.2.2.2 Fractional sample interpolation process

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current prediction unit relative to the top left luma sample of the current coding unit,

– the width and height of this prediction unit, nPSW and nPSH, 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,

– the selected reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

Outputs of this process are:

– a (nPSW)x(nPSH) array $predSampleLX_L$ of prediction luma sample values,

– two (nPSW/2)x(nPSH/2) arrays $predSampleLX_{Cb}$, and $predSampleLX_{Cr}$ of prediction chroma sample values.

The location ( xP, yP ) given in full-sample units of the upper-left luma samples of the current prediction unit relative to the upper-left luma sample location of the given reference sample arrays is derived by

$$xP = xC + xB \quad (8\text{-}126)$$
$$yP = yC + yB \quad (8\text{-}127)$$

Deleted: 87

Deleted: 88

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 $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

For each luma sample location ( $0 <= x_L < nPSW, 0 <= y_L < nPSH$ ) inside the prediction luma sample array $predSampleLX_L$, the corresponding prediction luma sample value $predSampleLX_L[x_L, y_L]$ is derived as follows:

– The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by

$$xInt_L = xP + ( mvLX[ 0 ] >> 2 ) + x_L \quad (8\text{-}128)$$
$$yInt_L = yP + ( mvLX[ 1 ] >> 2 ) + y_L \quad (8\text{-}129)$$

Deleted: 89

Deleted: 90

$$xFrac_L = mvLX[ 0 ] \& 3 \quad (8\text{-}130)$$
$$yFrac_L = mvLX[ 1 ] \& 3 \quad (8\text{-}131)$$

Deleted: 91

Deleted: 92

– The prediction luma sample value $predSampleLX_L[ x_L, y_L ]$ is derived by invoking the process specified in subclause 8.4.2.2.2.1 with ( $xInt_L$, $yInt_L$ ), ( $xFrac_L$, $yFrac_L$ ) and $refPicLX_L$ given as input.

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 ( $0 <= x_C < nPSW/2, 0 <= y_C < nPSH/2$) inside the prediction chroma sample arrays $predSampleLX_{Cb}$ and $predSampleLX_{Cr}$, the corresponding prediction chroma sample values $predSampleLX_{Cb}[ x_C, y_C ]$ and $predSampleLX_{Cr}[ x_C, y_C ]$ are derived asfollows:

– The variables $xInt_C$, $yInt_C$, $xFrac_C$, and $yFrac_C$ are derived by

$$xInt_C = ( xP / 2 ) + ( mvCLX[ 0 ] >> 3 ) + x_C \quad (8\text{-}132)$$
$$yInt_C = ( yP / 2 ) + ( mvCLX[ 1 ] >> 3 ) + y_C \quad (8\text{-}133)$$

$$xFrac_C = mvLX[ 0 ] \& 7 \quad (8\text{-}134)$$
$$yFrac_C = mvLX[ 1 ] \& 7 \quad (8\text{-}135)$$

Deleted: 93
Deleted: 94
Deleted: 95
Deleted: 96

– The prediction sample value $predSampleLX_{Cb}[ x_C, y_C ]$ is derived by invoking the process specified in subclause 8.4.2.2.2.2 with ( $xInt_C$, $yInt_C$ ), ( $xFrac_C$, $yFrac_C$ ) and $refPicLX_{Cb}$ given as input.

– The prediction sample value $predSampleLX_{Cr}[ x_C, y_C ]$ is derived by invoking the process specified in subclause 8.4.2.2.2.2 with ( $xInt_C$, $yInt_C$ ), ( $xFrac_C$, $yFrac_C$ ) and $refPicLX_{Cr}$ given as input.

#### 8.4.2.2.2.1 Luma sample interpolation process

Inputs to this process are:

– a luma location in full-sample units ( $xInt_L$, $yInt_L$ ),

– a luma location in fractional-sample units ( $xFrac_L$, $yFrac_L$ ),

– the luma reference sample array $refPicLX_L$.

Output of this process is a predicted luma sample value $predSampleLX_L[ x_L, y_L ]$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_{-1,-1}$ | | | | $A_{0,-1}$ | $a_{0,-1}$ | $b_{0,-1}$ | $c_{0,-1}$ | $A_{1,-1}$ | | | | $A_{2,-1}$ |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| $A_{-1,0}$ | | | | $A_{0,0}$ | $a_{0,0}$ | $b_{0,0}$ | $c_{0,0}$ | $A_{1,0}$ | | | | $A_{2,0}$ |
| $d_{-1,0}$ | | | | $d_{0,0}$ | $e_{0,0}$ | $f_{0,0}$ | $g_{0,0}$ | $d_{1,0}$ | | | | $d_{2,0}$ |
| $h_{-1,0}$ | | | | $h_{0,0}$ | $i_{0,0}$ | $j_{0,0}$ | $k_{0,0}$ | $h_{1,0}$ | | | | $h_{2,0}$ |
| $n_{-1,0}$ | | | | $n_{0,0}$ | $p_{0,0}$ | $q_{0,0}$ | $r_{0,0}$ | $n_{1,0}$ | | | | $n_{2,0}$ |
| $A_{-1,1}$ | | | | $A_{0,1}$ | $a_{0,1}$ | $b_{0,1}$ | $c_{0,1}$ | $A_{1,1}$ | | | | $A_{2,1}$ |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |
| $A_{-1,2}$ | | | | $A_{0,2}$ | $a_{0,2}$ | $b_{0,2}$ | $c_{0,2}$ | $A_{1,2}$ | | | | $A_{2,2}$ |

**Figure 8-6 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation**

In Figure 8-6, the positions labelled with upper-case letters $A_{i,j}$ 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 predSampleLX$_L$[ $x_L$, $y_L$ ]. The locations ( $xA_{i,j}$, $yA_{i,j}$ ) for each of the corresponding luma samples $A_{i,j}$ inside the given array refPicLX$_L$ of luma samples are derived as follows:

$$xA_{i,j} = \text{Clip3}( 0, \text{PicWidthInSamples}_L - 1, xInt_L + i ) \quad (8\text{-}136)$$

$$yA_{i,j} = \text{Clip3}( 0, \text{PicHeightInSamples}_L - 1, yInt_L + j ) \quad (8\text{-}137)$$

Variables shift1, shift2, shift3, offset1 and offset2 are derived as follows.

– The variable shift1 is set equal to $\text{BitDepth}_Y - 8$, the variable shift2 is set equal to $\text{BitDepth}_Y - 2$, and the variable shift3 is set equal to $14 - \text{BitDepth}_Y$.

– If the variable shift1 is equal to 0, the variable offset1 is set equal to 0, otherwise, the variable offset1 is set equal to $1 << ( \text{shift1} - 1 )$.

– The variable offset2 is set equal to $1 << ( \text{shift2} - 1 )$.

Given the luma samples $A_{i,j}$ at full-sample locations ( $xA_{i,j}$, $yA_{i,j}$ ), the luma samples '$a_{0,0}$' to '$r_{0,0}$' at fractional sample positions are derived by the following rules.

– The samples labelled $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, $d_{0,0}$, $h_{0,0}$, and $n_{0,0}$ shall be derived by applying the 8-tap filter to the nearest integer position samples and clipping the filtered value:

$$a_{0,0} = ( -A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 57*A_{0,0} + 19*A_{1,0} - 7*A_{2,0} + 3*A_{3,0} - A_{4,0} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}138)$$

$$b_{0,0} = ( -A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}139)$$

$$c_{0,0} = ( -A_{-3,0} + 3*A_{-2,0} - 7*A_{-1,0} + 19*A_{0,0} + 57*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}140)$$

$$d_{0,0} = ( -A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 57*A_{0,0} + 19*A_{0,1} - 7*A_{0,2} + 3*A_{0,3} - A_{0,4} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}141)$$

$$h_{0,0} = ( -A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}142)$$

$$n_{0,0} = ( -A_{0,-3} + 3*A_{0,-2} - 7*A_{0,-1} + 19*A_{0,0} + 57*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4} + \text{offset1} ) >> \text{shift1} \quad (8\text{-}143)$$

– The samples labelled $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ and $r_{0,0}$ shall be derived by first calculating intermediate values denoted as $d1_{i,0}$, $h1_{i,0}$ and $n1_{i,0}$ where $i = -3..4$ by applying the 8-tap filter to the nearest integer position samples in vertical direction: [Ed: (WJ) horizontal first yields the same result]

$$d1_{i,0} = -A_{i,-3} + 4*A_{i,-2} - 10*A_{i,-1} + 57*A_{i,0} + 19*A_{i,1} - 7*A_{i,2} + 3*A_{i,3} - A_{i,4} \quad (8\text{-}144)$$

$$h1_{i,0} = -A_{i,-3} + 4*A_{i,-2} - 11*A_{i,-1} + 40*A_{i,0} + 40*A_{i,1} - 11*A_{i,2} + 4*A_{i,3} - A_{i,4} \quad (8\text{-}145)$$

$$n1_{i,0} = -A_{i,-3} + 3*A_{i,-2} - 7*A_{i,-1} + 19*A_{i,0} + 57*A_{i,1} - 10*A_{i,2} + 4*A_{i,3} - A_{i,4} \quad (8\text{-}146)$$

– The final prediction values $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ and $r_{0,0}$ shall be derived by applying the 8-tap filter to the intermediate values $d1_{i,0}$, $h1_{i,0}$ and $n1_{i,0}$ where $i = -3..4$ in horizontal direction:

$$e_{0,0} = ( -d1_{-3,0} + 4*d1_{-2,0} - 10*d1_{-1,0} + 57*d1_{0,0} + 19*d1_{1,0} - 7*d1_{2,0} + 3*d1_{3,0} - d1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}147)$$

$$f_{0,0} = ( -d1_{-3,0} + 4*d1_{-2,0} - 11*d1_{-1,0} + 40*d1_{0,0} + 40*d1_{1,0} - 11*d1_{2,0} + 4*d1_{3,0} - d1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}148)$$

$$g_{0,0} = ( -d1_{-3,0} + 3*d1_{-2,0} - 7*d1_{-1,0} + 19*d1_{0,0} + 57*d1_{1,0} - 10*d1_{2,0} + 4*d1_{3,0} - d1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}149)$$

$$i_{0,0} = ( -h1_{-3,0} + 4*h1_{-2,0} - 10*h1_{-1,0} + 57*h1_{0,0} + 19*h1_{1,0} - 7*h1_{2,0} + 3*h1_{3,0} - h1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}150)$$

$$j_{0,0} = ( -h1_{-3,0} + 4*h1_{-2,0} - 11*h1_{-1,0} + 40*h1_{0,0} + 40*h1_{1,0} - 11*h1_{2,0} + 4*h1_{3,0} - h1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}151)$$

$$k_{0,0} = ( -h1_{-3,0} + 3*h1_{-2,0} - 7*h1_{-1,0} + 19*h1_{0,0} + 57*h1_{1,0} - 10*h1_{2,0} + 4*h1_{3,0} - h1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}152)$$

$$p_{0,0} = ( -n1_{-3,0} + 4*n1_{-2,0} - 10*n1_{-1,0} + 57*n1_{0,0} + 19*n1_{1,0} - 7*n1_{2,0} + 3*n1_{3,0} - n1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}153)$$

$$q_{0,0} = ( -n1_{-3,0} + 4*n1_{-2,0} - 11*n1_{-1,0} + 40*n1_{0,0} + 40*n1_{1,0} - 11*n1_{2,0} + 4*n1_{3,0} - n1_{4,0} + \text{offset2} ) >> \text{shift2} \quad (8\text{-}154)$$

Deleted: Figure 8-5
Deleted: 97
Deleted: 98
Deleted: 99
Deleted: 100
Deleted: 101
Deleted: 102
Deleted: 103
Deleted: 104
Deleted: 105
Deleted: 106
Deleted: 107
Deleted: 108
Deleted: 109
Deleted: 110
Deleted: 111
Deleted: 112
Deleted: 113
Deleted: 114
Deleted: 115

$r_{0,0} = ( -n1_{-3,0} + 3*n1_{-2,0} - 7*n1_{-1,0} + 19*n1_{0,0} + 57*n1_{1,0} - 10*n1_{2,0} + 4*n1_{3,0} - n1_{4,0} + offset2 ) >> shift2$ (8-155)

The positions labelled with lower-case letters within un-shaded blocks represent luma samples at quarter-pel sample fractional locations. 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 $predSampleLX_L[ x_L, y_L ]$. This assignment is done according to Table 8-11. The value of $predSampleLX_L[ x_L, y_L ]$ shall be the output.

**Table 8-11 – Assignment of the luma prediction sample $predSampleLX_L[ x_L, y_L ]$**

| **xFracL** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **yFracL** | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| **$predSampleLX_L[ x_L, y_L ]$** | A << shift3 | d | h | n | a | e | i | p | b | f | j | q | c | g | k | r |

#### 8.4.2.2.2.2 Chroma sample interpolation process

Inputs to this process are:

– a chroma location in full-sample units ( $xInt_C$, $yInt_C$ ),

– a chroma location in fractional-sample units ( $xFrac_C$, $yFrac_C$ ),

– the chroma reference sample array $refPicLX_C$.

Output of this process is a predicted chroma sample value $predSampleLX_C[ x_C, y_C ]$

| | $ha_{0,-1}$ | $hb_{0,-1}$ | $hc_{0,-1}$ | $hd_{0,-1}$ | $he_{0,-1}$ | $hf_{0,-1}$ | $hg_{0,-1}$ | $hh_{0,-1}$ | |
|---|---|---|---|---|---|---|---|---|---|
| $ah_{-1,0}$ | $B_{0,0}$ | $ab_{0,0}$ | $ac_{0,0}$ | $ad_{0,0}$ | $ae_{0,0}$ | $af_{0,0}$ | $ag_{0,0}$ | $ah_{0,0}$ | $B_{1,0}$ |
| $bh_{-1,0}$ | $ba_{0,0}$ | $bb_{0,0}$ | $bc_{0,0}$ | $bd_{0,0}$ | $be_{0,0}$ | $bf_{0,0}$ | $bg_{0,0}$ | $bh_{0,0}$ | $ba_{1,0}$ |
| $ch_{-1,0}$ | $ca_{0,0}$ | $cb_{0,0}$ | $cc_{0,0}$ | $cd_{0,0}$ | $ce_{0,0}$ | $cf_{0,0}$ | $cg_{0,0}$ | $ch_{0,0}$ | $ca_{1,0}$ |
| $dh_{-1,0}$ | $da_{0,0}$ | $db_{0,0}$ | $dc_{0,0}$ | $dd_{0,0}$ | $de_{0,0}$ | $df_{0,0}$ | $dg_{0,0}$ | $dh_{0,0}$ | $da_{1,0}$ |
| $eh_{-1,0}$ | $ea_{0,0}$ | $eb_{0,0}$ | $ec_{0,0}$ | $ed_{0,0}$ | $ee_{0,0}$ | $ef_{0,0}$ | $eg_{0,0}$ | $eh_{0,0}$ | $ea_{1,0}$ |
| $fh_{-1,0}$ | $fa_{0,0}$ | $fb_{0,0}$ | $fc_{0,0}$ | $fd_{0,0}$ | $fe_{0,0}$ | $ff_{0,0}$ | $fg_{0,0}$ | $fh_{0,0}$ | $fa_{1,0}$ |
| $gh_{-1,0}$ | $ga_{0,0}$ | $gb_{0,0}$ | $gc_{0,0}$ | $gd_{0,0}$ | $ge_{0,0}$ | $gf_{0,0}$ | $gg_{0,0}$ | $gh_{0,0}$ | $ga_{1,0}$ |
| $hh_{-1,0}$ | $ha_{0,0}$ | $hb_{0,0}$ | $hc_{0,0}$ | $hd_{0,0}$ | $he_{0,0}$ | $hf_{0,0}$ | $hg_{0,0}$ | $hh_{0,0}$ | $ha_{1,0}$ |
| | $B_{0,1}$ | $ab_{0,1}$ | $ac_{0,1}$ | $ad_{0,1}$ | $ae_{0,1}$ | $af_{0,1}$ | $ag_{0,1}$ | $ah_{0,1}$ | $B_{1,1}$ |

**Figure 8-7 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for eighth sample chroma interpolation**

In Figure 8-6, the positions labelled with upper-case letters $B_{i,j}$ within shaded blocks represent chroma samples at full-sample locations inside the given two-dimensional array $refPicLX_C$ of chroma samples. These samples may be used for

Deleted: 116

Deleted: Table 8-9

Deleted: 8-9

Deleted: 8-9

Deleted: Figure 8-5

generating the predicted chroma sample value predSampleLX$_C$[ $x_C$, $y_C$ ]. The locations ( $xB_{i,j}$, $yB_{i,j}$ ) for each of the corresponding chroma samples $B_{i,j}$ inside the given array refPicLX$_C$ of chroma samples are derived as follows:

$$xB_{i,j} = Clip3( 0, PicWidthInSamples_C - 1, xInt_C + i ) \quad (8\text{-}156)$$
$$yB_{i,j} = Clip3( 0, PicHeightInSamples_C - 1, yInt_C + j ) \quad (8\text{-}157)$$

Variables shift1, shift2, shift3, offset1 and offset2 are derived as follows.

– The variable shift1 is set equal to $BitDepth_C - 8$, the variable shift2 is set equal to $BitDepth_C - 2$, and the variable shift3 is set equal to $14 - BitDepth_C$.

– If the variable shift1 is equal to 0, the variable offset1 is set equal to 0, otherwise, the variable offset1 is set equal to $1 << ( shift1 - 1 )$.

– The variable offset2 is set equal to $1 << ( shift2 - 1 )$.

Given the chroma samples $B_{i,j}$ at full-sample locations ( $xB_{i,j}$, $yB_{i,j}$ ), the chroma samples '$ab_{0,0}$' to '$hh_{0,0}$' at fractional sample positions are derived by the following rules.

– The samples labelled $ab_{0,0}$, $ac_{0,0}$, $ad_{0,0}$, $ae_{0,0}$, $af_{0,0}$, $ag_{0,0}$, and $ah_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples and clipping the filtered value:

$$ab_{0,0} = ( -3*B_{-1,0} + 60*B_{0,0} + 8*B_{1,0} - B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}158)$$
$$ac_{0,0} = ( -4*B_{-1,0} + 54*B_{0,0} + 16*B_{1,0} - 2*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}159)$$
$$ad_{0,0} = ( -5*B_{-1,0} + 46*B_{0,0} + 27*B_{1,0} - 4*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}160)$$
$$ae_{0,0} = ( -4*B_{-1,0} + 36*B_{0,0} + 36*B_{1,0} - 4*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}161)$$
$$af_{0,0} = ( -4*B_{-1,0} + 27*B_{0,0} + 46*B_{1,0} - 5*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}162)$$
$$ag_{0,0} = ( -2*B_{-1,0} + 16*B_{0,0} + 54*B_{1,0} - 4*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}163)$$
$$ah_{0,0} = ( -B_{-1,0} + 8*B_{0,0} + 60*B_{1,0} - 3*B_{2,0} + offset1 ) >> shift1 \quad (8\text{-}164)$$

– The samples labelled $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$, and $ha_{0,0}$ shall be derived by applying the 4-tap filter to the nearest integer position samples and clipping the filtered value:

$$ba_{0,0} = ( -3*B_{0,-1} + 60*B_{0,0} + 8*B_{0,1} - B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}165)$$
$$ca_{0,0} = ( -4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}166)$$
$$da_{0,0} = ( -5*B_{0,-1} + 46*B_{0,0} + 27*B_{0,1} - 4*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}167)$$
$$ea_{0,0} = ( -4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}168)$$
$$fa_{0,0} = ( -4*B_{0,-1} + 27*B_{0,0} + 46*B_{0,1} - 5*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}169)$$
$$ga_{0,0} = ( -2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}170)$$
$$ha_{0,0} = ( -B_{0,-1} + 8*B_{0,0} + 60*B_{0,1} - 3*B_{0,2} + offset1 ) >> shift1 \quad (8\text{-}171)$$

– The samples labelled $Xb_{0,0}$, $Xc_{0,0}$, $Xd_{0,0}$, $Xe_{0,0}$, $Xf_{0,0}$, $Xg_{0,0}$ and $Xh_{0,0}$ for X being replaced by b, c, d, e, f, g and h, respectively, shall be derived by first calculating intermediate values denoted as $ba_{i,0}$, $ca_{i,0}$, $da_{i,0}$, $ea_{i,0}$, $fa_{i,0}$, $ga_{i,0}$ and $ha_{i,0}$ where i = -1..2 by applying the 4-tap filter to the nearest integer position samples in vertical direction: [Ed: (WJ) horizontal first yields the same result]

$$ba_{i,0} = -3*B_{0,-1} + 60*B_{0,0} + 8*B_{0,1} - B_{0,2} \quad (8\text{-}172)$$
$$ca_{i,0} = -4*B_{0,-1} + 54*B_{0,0} + 16*B_{0,1} - 2*B_{0,2} \quad (8\text{-}173)$$
$$da_{i,0} = -5*B_{0,-1} + 46*B_{0,0} + 27*B_{0,1} - 4*B_{0,2} \quad (8\text{-}174)$$
$$ea_{i,0} = -4*B_{0,-1} + 36*B_{0,0} + 36*B_{0,1} - 4*B_{0,2} \quad (8\text{-}175)$$

Deleted: 117
Deleted: 118
Deleted: 119
Deleted: 120
Deleted: 121
Deleted: 122
Deleted: 123
Deleted: 124
Deleted: 125
Deleted: 126
Deleted: 127
Deleted: 128
Deleted: 129
Deleted: 130
Deleted: 131
Deleted: 132
Deleted: 133
Deleted: 134
Deleted: 135
Deleted: 136

$fa_{i,0} = -4*B_{0,-1} + 27*B_{0,0} + 46*B_{0,1} - 5*B_{0,2}$ (8-176)

$ga_{i,0} = -2*B_{0,-1} + 16*B_{0,0} + 54*B_{0,1} - 4*B_{0,2}$ (8-177)

$ha_{i,0} = -B_{0,-1} + 8*B_{0,0} + 60*B_{0,1} - 3*B_{0,2}$ (8-178)

– The final prediction values $Xb_{0,0}$, $Xc_{0,0}$, $Xd_{0,0}$, $Xe_{0,0}$, $Xf_{0,0}$, $Xg_{0,0}$ and $Xh_{0,0}$ for X being replaced by b, c, d, e, f, g and h, respectively, shall be derived by applying the 4-tap filter to the intermediate values $Xa_{i,0}$ where i = -1..2 in horizontal direction:

$Xb_{0,0} = ( -3*Xa_{-1,0} + 60*Xa_{0,0} + 8*Xa_{1,0} - Xa_{2,0} + offset2 ) >> shift2$ (8-179)

$Xc_{0,0} = ( -4*Xa_{-1,0} + 54*Xa_{0,0} + 16*Xa_{1,0} - 2*Xa_{2,0} + offset2 ) >> shift2$ (8-180)

$Xd_{0,0} = ( -5*Xa_{-1,0} + 46*Xa_{0,0} + 27*Xa_{1,0} - 4*Xa_{2,0} + offset2 ) >> shift2$ (8-181)

$Xe_{0,0} = ( -4*Xa_{-1,0} + 36*Xa_{0,0} + 36*Xa_{1,0} - 4*Xa_{2,0} + offset2 ) >> shift2$ (8-182)

$Xf_{0,0} = ( -4*Xa_{-1,0} + 27*Xa_{0,0} + 46*Xa_{1,0} - 5*Xa_{2,0} + offset2 ) >> shift2$ (8-183)

$Xg_{0,0} = ( -2*Xa_{-1,0} + 16*Xa_{0,0} + 54*Xa_{1,0} - 4*Xa_{2,0} + offset2 ) >> shift2$ (8-184)

$Xh_{0,0} = ( -Xa_{-1,0} + 8*Xa_{0,0} + 60*Xa_{1,0} - 3*Xa_{2,0} + offset2 ) >> shift2$ (8-185)

The positions labelled with lower-case letters within un-shaded blocks represent chroma samples at eighth-pel sample fractional locations. The chroma location offset in fractional-sample units ( $xFrac_C$, $yFrac_C$ ) specifies which of the generated chroma samples at full-sample and fractional-sample locations is assigned to the predicted chroma sample value $predSampleLX_C[ x_C, y_C ]$. This assignment is done according to Table 8-12. The value of $predSampleLX_C[ x_C, y_C ]$ shall be the output.

**Table 8-12 – Assignment of the chroma prediction sample $predSampleLX_C[ x_C, y_C ]$ for ( X, Y ) being replaced by ( 1, b ), ( 2, c ), ( 3, d ), ( 4, e ), ( 5, f ), ( 6, g ), and ( 7, h ), respectively**

| **xFracC** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **yFracC** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **predSampleLX$_C$[ x$_C$, y$_C$ ]** | B << shift3 | ba | ca | da | ea | fa | ga | ha | aY | bY | cY | dY | eY | fY | gY | hY |

#### 8.4.2.2.3 Weighted sample prediction process

[Ed.: (KU) only the default weighted prediction part is added for bi-directional averaging]

Inputs to this process are:

– a location ( xB, yB ) specifying the top-left sample of the current prediction unit relative to the top left sample of the current coding unit,

– the width and height of this prediction unit, nPSW and nPSH,

– two (nPSW)x(nPSH) arrays predSamplesL0 and predSamplesL1,

– prediction list utilization flags, predFlagL0 and predFlagL1,

– the bit-depth of the chroma component, bitDepth.

Outputs of this process are:

– the (nPSW)x(nPSH) array predSamples of prediction sample values.

Variables shift1, shift2, offset1 and offset2 are derived as follows.

– The variable shift1 is set equal to 14 – bitDepth and the variable shift2 is set equal to 15 – bitDepth,

– The variable offset1 is set equal to 1 << ( shift1 – 1 ) and the variable offset2 is set equal to 1 << ( shift2 – 1 ).

Depending on the value of predFlagL0 and predFlagL1, the prediction samples predSamples[ x, y ] with x = 0..(nPSW)-1 and y = 0..(nPSH)-1 are derived as follows.



Deleted: 137
Deleted: 138
Deleted: 139
Deleted: 140
Deleted: 141
Deleted: 142
Deleted: 143
Deleted: 144
Deleted: 145
Deleted: 146
Deleted: Table 8-10
Deleted: 8-10
Deleted: 8-10

– If predFlagL0 is equal to 1 and predFlagL1 is equal to 0,

predSamples[ x, y ] = ( predSamplesL0[ x, y ] + offset1 ) >> shift1 (8-186)

– Otherwise, if predFlagL0 is equal to 0 and predFlagL1 is equal to 1,

predSamples[ x, y ] = ( predSamplesL1[ x, y ] + offset1 ) >> shift1 (8-187)

– Otherwise (both predFlagL0 and predFlagL1 are equal to 1),

predSamples[ x, y ] = Clip3( 0, ( 1 << bitDepth ) – 1 ,
predSamplesL0[ x, y ] + predSamplesL1[ x, y ] + offset2 ) >> shift2 ) (8-188)

Deleted: 147

Deleted: 148

Deleted: 149

### 8.4.3 Decoding process for the residual signal of coding units coded in inter prediction mode

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

Outputs of this process are:

– a ($nCS_L$)x($nCS_L$) array $resSamples_L$ of luma residual samples, where $nCS_L$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $resSamples_{Cb}$ of chroma residual samples for the component Cb, where $nCS_C$ is derived as specified below,

– a ($nCS_C$)x($nCS_C$) array $resSamples_{Cr}$ of chroma residual samples for the component Cr, where $nCS_C$ is derived as specified below.

The variable $nCS_L$ is set equal to 1 << log2CUSize and the variable $nCS_C$ is set equal to ( 1 << log2CUSize ) >> 1.

Let $resSamples_L$ be a ($nCS_L$)x($nCS_L$) array of luma residual samples and let $resSamples_{Cb}$ and $resSamples_{Cr}$ be two ($nCS_C$)x($nCS_C$) arrays of chroma residual samples.

Depending on no_residual_data_flag, the following applies:

– If no_residual_data_flag is equal to 1, all samples of the ($nCS_L$)x($nCS_L$) array $resSamples_L$ and all samples of the two ($nCS_C$)x($nCS_C$) arrays $resSamples_{Cb}$ and $resSamples_{Cr}$ are set equal to 0.

– Otherwise (no_residual_data_flag is equal to 0), the following ordered steps apply:

1. The decoding process for luma residual blocks as specified in subclause 8.4.3.1 below is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the variable log2TrafoSize set equal to log2CUSize, the variable trafoDepth set equal to 0, the variable nCS set equal to nCSL, and the (nCSL)x(nCSL) array resSamplesL as the inputs and the output is a modified version of the (nCSL)x(nCSL) array resSamplesL.

2. The decoding process for chroma residual blocks as specified in subclause 8.4.3.2 below is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the variable log2TrafoSize set equal to log2CUSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 1, the variable nCS set equal to $nCS_C$, and the ($nCS_C$)x($nCS_C$) array $resSamples_{Cb}$ as the inputs and the output is a modified version of the ($nCS_C$)x($nCS_C$) array $resSamples_{Cb}$.

3. The decoding process for chroma residual blocks as specified in subclause 8.4.3.2 below is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ) set equal to ( 0, 0 ), the variable log2TrafoSize set equal to log2CUSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 2, the variable nCS set equal to $nCS_C$, and the ($nCS_C$)x($nCS_C$) array $resSamples_{Cr}$ as the inputs and the output is a modified version of the ($nCS_C$)x($nCS_C$) array $resSamples_{Cr}$.

#### 8.4.3.1 Decoding process for luma residual blocks

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,

– a variable log2TrafoSize specifying the size of the current block,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable nCS specifying the size, in luma samples, of the current coding unit,

– a (nCS)x(nCS) array resSamples of luma residual samples.

Output of this process is:

– a modified version of the (nCS)x(nCS) array of luma residual samples.

Depending split_transform_flag[ xB ][ yB ][ trafoDepth ], the following applies:

– If split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variable xB1 is set equal to xB + ( ( 1 << log2TrafoSize ) >> 1 ).
2. The variable yB1 is set equal to yB + ( ( 1 << log2TrafoSize ) >> 1 ).
3. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
4. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB1, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
5. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
6. The decoding process for luma residual blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB1, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

– Otherwise (split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following ordered steps apply:

1. The variable nS is set equal to 1 << log2TrafoSize.
2. The scaling and transformation process as specified in subclause 8.5.1 is invoked with the luma location ( xC + xB, yC +yB ), the variable trafoDepth, the variable cIdx set equal to 0, and the transform size trafoSize set equal to nS as the inputs and the output is a (nS)x(nS) array resSamplesBlock.
3. The array construction process as specified in subclause 8.5.5 is invoked with the luma location ( xB, yB ), the variable cIdx set equal to 0, the variable inputArraySize set equal to nS, the variable outputArraySize set equal to nCS, the (nS)x(nS) array resSamplesBlock, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

Deleted: 8.5.1

#### 8.4.3.2 Decoding process for chroma residual blocks

Inputs to this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,

– a variable log2TrafoSize specifying the size of the current block,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable cIdx specifying the chroma component of the current block,

– a variable nCS specifying the size, in chroma samples, of the current coding unit,

– a (nCS)x(nCS) array resSamples of chroma residual samples.

Output of this process is:

– a modified version of the (nCS)x(nCS) array of chroma residual samples.

The variable splitChromaFlag is derived as follows:

– If split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 1 and log2TrafoSize is greater than Log2MinTrafoSize + 1, splitChromaFlag is set equal to 1.

– Otherwise (split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 0 or log2TrafoSize is equal to Log2MinTrafoSize + 1), splitChromaFlag is set equal to 0.

Depending splitChromaFlag, the following applies:

– If splitChromaFlag is equal to 1, the following ordered steps apply:

1. The variable xB1 is set equal to xB + ( ( 1 << log2TrafoSize ) >> 1 ).
2. The variable yB1 is set equal to yB + ( ( 1 << log2TrafoSize ) >> 1 ).
3. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
4. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB1, yB ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
5. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.
6. The decoding process for residual chroma blocks as specified in this subclause is invoked with the luma location ( xC, yC ), the luma location ( xB1, yB1 ), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the variable cIdx, the variable nCS, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

– Otherwise (splitChromaFlag is equal to 0), the following ordered steps apply:

1. The variable nS is set equal to ( 1 << log2TrafoSize ) >> 1.
2. The scaling and transformation process as specified in subclause 8.5.1 is invoked with the luma location ( xC + xB, yC +yB ), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nS as the inputs and the output is a (nS)x(nS) array resSamplesBlock.
3. The array construction process as specified in subclause 8.5.5 is invoked with the luma location ( xB, yB ), the variable cIdx, the variable inputArraySize set equal to nS, the variable outputArraySize set equal to nCS, the (nS)x(nS) array resSamplesBlock, and the (nCS)x(nCS) array resSamples as the inputs and the output is a modified version of the (nCS)x(nCS) array resSamples.

## 8.5 Scaling, transformation and array construction process prior to deblocking filter process

### 8.5.1 Scaling and transformation process

Inputs to this process are:

– a luma location ( xT, yT ) specifying the top-left luma sample of the current transform unit relative to the top-left luma sample of the current picture,

– a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,

– a variable cIdx specifying the chroma component of the current block,

– a variable nS specifying the size of the current transform unit.

Output of this process is:

– a modified version of the (nS)x(nS) array of residual samples r with elements $r_{ij}$.

The quantization parameter qP is derived as follows.

– If cIdx is equal to 0,

qP = QP'$_Y$ (8-189)

Deleted: 150

– Otherwise,

$$qP = QP'_C \quad (8\text{-}190)$$

The (nS)x(nS) array of residual samples r are derived as specified in the following ordered steps.

1. The inverse scanning process for transform coefficients as specified in subclause 8.5.2 is invoked with the size of the current transform unit nS, the list of transform coefficients transCoeffLevel[ xT ][ yT ][ trafoDepth ][ cIdx ] and the variable cIdx as the inputs and the output is a (nS)x(nS) array $c_{ij}$.
2. The scaling process for transform coefficients as specified in subclause 8.5.3 is invoked with the size of the transform unit nS, the (nS)x(nS) array c, the chroma component variable cIdx and the quantization parameter qP as the inputs and the output is a scaled transform coefficient (nS)x(nS) array d.
3. The transformation process for scaled transform coefficients as specified in subclause 8.5.4 is invoked with the size of the transform unit nS, the scaled transform coefficient (nS)x(nS) array d and the chroma component variable cIdx as the inputs and the output is a residual samples (nS)x(nS) array r.

### 8.5.2 Inverse scanning process for transform coefficients

Inputs to this process are:

– a variable nS specifying the size of the current transform unit,

– a list of (nS)x(nS) values transCoeffLevel[ xT ][ yT ][ trafoDepth ][ cIdx ].

– a variable cIdx specifying the chroma component of the current block,

Output of this process is a variable c containing a two-dimensional array of (nS)x(nS) values.

– The variable c[ i ][ j ] which is located in the ( i, j ) position in the array c is derived as follows.

– The current transform unit is divided into ( nS >> 2 ) * ( nS >> 2 ) subsets of 4 * 4 samples. The subsets are processed in zig-zag scan order. The position ( k0, l0 ) in the array c, corresponding to the top left position of a subset s, is derived as follows.

$$k0 = ZigZag[\ \log2(\ nS >> 2\ )\ ][\ s\ ][\ 0\ ] << 2$$
$$l0 = ZigZag[\ \log2(\ nS >> 2\ )\ ][\ s\ ][\ 1\ ] << 2,$$
$$\text{with } s = 0..(\ (\ nS >> 2\ ) * (\ nS >> 2\ )\ ) - 1\ ) \quad (8\text{-}191)$$

– For every subset s, the 16 transform coefficient level at scanning position n are mapped to the position ( i, j ) in the array c using the reverse zig-zag scan.

$$i = k0 + ZigZag[\ 2\ ][\ n\ ][\ 0\ ]$$
$$j = l0 + ZigZag[\ 2\ ][\ n\ ][\ 1\ ]$$
$$c[\ i\ ][\ j\ ] = transCoeffLevel[\ xT\ ][\ yT\ ][\ trafoDepth\ ][\ cIdx\ ][\ n + (\ s << 4\ )\ ],$$
$$\text{with } n = 15..0 \quad (8\text{-}192)$$

### 8.5.3 Scaling process for transform coefficients

Inputs of this process are:

– a variable nS specifying the size of the current transform unit,

– a (nS)x(nS) array c of transform coefficients with elements $c_{ij}$,

– a variable cIdx specifying the chroma component of the current block,

– a variable qP specifying the quantization parameter.

Output of this process is scaled transform coefficients as a (nS)x(nS) array of d with elements $d_{ij}$.

The variable trafoPrecisionExt is derived as follows:

– If nS is greater than 4,

  – If cIdx is equal to 0 and bit_depth_luma_minus8 is equal to 0, trafoPrecisionExt is set equal to 2.

  – Otherwise, if cIdx is not equal to 0 and bit_depth_chroma_minus8 is equal to 0, trafoPrecisionExt is set equal to 2.

  – Otherwise, trafoPrecisionExt is set equal to 0.

Deleted: 151

Deleted: 152

Deleted: 152

– Otherwise, trafoPrecisionExt is set equal to 0.

[Ed.: (WJ) wait decision on transform precision extension]

The scaled transform coefficient array dij is derived as follows.

$$d_{ij} = ( c_{ij} * \text{LevelScale}_{(nS)x(nS)}[ qP\%6 ][ i ][ j ] ) << ( qP/6 + \text{trafoPrecisionExt} ), \text{ with } i, j = 0..nS-1 \quad (8\text{-}193)$$

[Ed: (WJ) $\text{LevelScale}_{(nS)x(nS)}$ should be very large table up to 32x32. Recent contributions (JCTVC-C209 and JCTVC-C255) shown that this matrix could be replaced by one constants per qP without coding efficiency penalty. Maybe it is better to wait.]

[Ed.: (WJ) dependent on transformation process – should be modified again after transformation process is fixed]

### 8.5.4 Transformation process for scaled transform coefficients

Inputs of this process are:

– a variable nS specifying the size of the current transform unit,

– a (nS)x(nS) array d of scaled transform coefficients with elements $d_{ij}$.

– a variable cIdx specifying the chroma component of the current block,

Output of this process is residual samples as a (nS)x(nS) array r with elements $r_{ij}$.

The variable trafoPrecisionExt is derived as follows:

– If nS is greater than 4,

  – If cIdx is equal to 0 and bit_depth_luma_minus8 is equal to 0, trafoPrecisionExt is set equal to 4.

  – Otherwise, if cIdx is not equal to 0 and bit_depth_chroma_minus8 is equal to 0, trafoPrecisionExt is set equal to 4.

  – Otherwise, trafoPrecisionExt is set equal to 0.

– Otherwise, trafoPrecisionExt is set equal to 0.

[Ed.: (WJ) wait decision on transform precision extension]

[Ed: (WJ) derivation of trafoPrecisionExt is duplicated to that of the scaling process. Better to remove one of two.]

Depending on PredMode and IntraPredMode, the following applies:

– If PredMode is equal to MODE_INTRA, nS is equal to 4, and cIdx is equal to 0, the variables horizTrType and vertTrType are specified as Table 8-13 with IntraPredMode as input. [Ed. (WJ): DST is applied only for luma 4x4 block]

– Otherwise, the variables horizTrType and vertTrType are set equal to 0.

**Table 8-13 – Specification of horizTrType and vertTrType**

| IntraPredMode | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| horizTrType | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| vertTrType | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

| IntraPredMode | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| horizTrType | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| vertTrType | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

The constructed residual samples are derived as specified in the following ordered steps.



Deleted: 158

Deleted: Depending on cIdx, a variable planarFlag is derived as follows:¶
→ If cIdx is equal to 0, planarFlag is set equal to planar_flag_luma¶
→ Otherwise, planarFlag is set equal to planar_flag_chroma [Ed. (WJ): it is not necessary now since DST is not applied to chroma anyway]¶

Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left

Deleted: Table 8-11

Deleted: 8-11

Formatted: Caption

Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left

1. Each (horizontal) row of scaled transform coefficients $d_{ij}$ (i, j=0..nS-1) is transformed to $e_{ij}$ (i, j=0..nS-1) by invoking the one-dimensional transformation process as specified in subclause 8.5.4.1 to 8.5.4.4 according to the size of the transform unit nS, with the (nS)x(nS) array d and the transform type variable horizTrType as the inputs and the output is the (nS)x(nS) array e.
2. Each (vertical) column of the resulting matrix $e_{ij}$ (i, j=0..nS-1) is transformed to $f_{ij}$ (i, j=0..nS-1) by invoking the one-dimensional transformation process as specified in subclause 8.5.4.1 to 8.5.4.4 according to the size of the transform unit nS, with the (nS)x(nS) array e and the transform type variable vertTrType as the inputs and the output is the (nS)x(nS) array f.
3. If nS is equal to 4 or 8, the residual sample value $r_{ij}$ is derived by

$$r_{ij} = ( f_{ij} + 2^{5+trafoPrecisionExt} ) >> ( 6 + trafoPrecisionExt ), \text{ with } i, j=0..(nS)x(nS)-1 \quad (8\text{-}194)$$

4. Otherwise (nS is equal to 16 or 32), the residual sample value $r_{ij}$ is derived by

$$r_{ij} = ( f_{ij} + 2^{9+trafoPrecisionExt} ) >> ( 10 + trafoPrecisionExt ), \text{ with } i, j=0..(nS)x(nS)-1 \quad (8\text{-}195)$$

[Ed.: (WJ) dependent on transformation process – should be modified again after transformation process is fixed]

### 8.5.4.1 Transformation process for 4 samples

Inputs of this process are an array of 4 samples of the scaled transform coefficients x with elements $x_i$, with i = 0..3 and the transform type variable trType.

Output of this process is an array of 4 samples of the residual samples y with elements $y_i$, with i = 0..3.

Depending on trType, the following applies:

– If tyType is equal to 1, the following ordered steps apply:

1. A set of intermediate values $c_0$, $c_1$, $c_2$ and $c_3$ is calculated as follows:

$$c_0 = x_0 + x_2 \quad (8\text{-}196)$$
$$c_1 = x_2 + x_3 \quad (8\text{-}197)$$
$$c_2 = x_0 - x_3 \quad (8\text{-}198)$$
$$c_3 = 74 * x_1 \quad (8\text{-}199)$$

2. The output values $y_i$ with i = 0..3 are then specified as follows:

$$y_0 = ( 29*c_0 + 55*c_1 + c_3 + 64 ) >> 7 \quad (8\text{-}200)$$
$$y_1 = ( 55*c_2 - 29*c_1 + c_3 + 64 ) >> 7 \quad (8\text{-}201)$$
$$y_2 = ( 74*( x_0 - x_2 + x_3 ) + 64 ) >> 7 \quad (8\text{-}202)$$
$$y_3 = ( 55*c_0 + 29 * c_2 - c_3 + 64 ) >> 7 \quad (8\text{-}203)$$

[Ed. (WJ): proponent's comment - to share the quantization scaling value, transform precision (7-bit in this case) should be synchronized with DCT]

– Otherwise, the following ordered steps apply: [Ed. (WJ): TBD – 4-point DCT]

### 8.5.4.2 Transformation process for 8 samples

Inputs of this process are an array of 8 samples of the scaled transform coefficients x with elements $x_i$, with i = 0..7 and the transform type variable trType.

Output of this process is an array of 8 samples of the residual samples y with elements $y_i$, with i = 0..7.

[Ed: (WJ) TBD]

### 8.5.4.3 Transformation process for 16 samples

Inputs of this process are an array of 16 samples of the scaled transform coefficients x with elements $x_i$, with i = 0..15 and the transform type variable trType.

Output of this process is an array of 16 samples of the residual samples y with elements $y_i$, with i = 0..15.

[Ed: (WJ) TBD]

Deleted: 8.5.4.4
Deleted:
Deleted: 159
Deleted: 160
Deleted: is are an array of 16 … samples of the scaled transf ... [68]
Deleted: 16
Deleted: [Ed: (WJ) TBD]
Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left
Formatted ... [69]
Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Keep with next, Keep lines together, Tab stops: 0.2", Left
Deleted: 159
Formatted ... [70]
Deleted: 159
Formatted ... [71]
Deleted: 159
Formatted ... [72]
Deleted: 159
Formatted: Subscript
Formatted: Subscript
Formatted: Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.25" + Tab after: 0.5" + Indent at: 0.5", Keep with next, Keep lines together, Tab stops: 0.2", Left
Deleted: 159
Formatted ... [73]
Deleted: 159
Formatted ... [74]
Deleted: 159
Formatted ... [75]
Deleted: 159
Formatted ... [76]
Formatted: Indent: Left: 0", Hanging: 0.3", Tab stops: 0.3", Left
Deleted: is are an array of 16 … samples of the scaled transf ... [77]
Deleted: 16
Deleted: is are an array of 16 samples of the scaled transform ... [78]

#### 8.5.4.4 Transformation process for 32 samples

Inputs of this process are an array of 32 samples of the scaled transform coefficients x with elements $x_i$, with i=0..31 and the transform type variable trType.

Output of this process is an array of 32 samples of the residual samples y with elements $y_i$, with i=0..31.

[Ed: (WJ) TBD]

### 8.5.5 Array construction process

Inputs of this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,

– a variable cIdx specifying the chroma component of the current block,

– a variable nS specifying the size of the current block,

– a variable nCS specifying the size of the current coding unit,

– a (nS)x(nS) array resSampleBlock specifying the current block samples,

– a (nCS)x(nCS) array resSample specifying the samples of the current coding unit.

Output of this process is a modified (nCS)x(nCS) array resSample specifying the samples of the current coding unit.

The modified array resSample is derived as follows.

resSample[ xB+i, yB+j ] += resSampleBlock[ i, j ], with i, j = 0..nS-1 (8-204)

## 8.6 In-loop filter process

[Ed: (WJ) overall process description need to be inserted]

### 8.6.1 Deblocking filter process

A conditional filtering process shall be performed on a treeblock basis after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture (as specified in subclauses XXX and YYY) [Ed.: (WJ) those subclauses seem not defined yet], with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process. For each coding unit, vertical edges are filtered first, starting with the edge on the left-hand side of the coding unit proceeding through the edges towards the right-hand side of the coding unit in their geometrical order. Then, the horizontal edges are filtered starting with the edge on the top of the coding unit proceeding through the edges towards the bottom of the coding unit in their geometrical order.

Sample values above of the current coding unit that may have already been modified by the filtering of horizontal edges of deblocking filter process operation on previous coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit may be modified by the filtering of vertical edge and may be further modified by the filtering of horizontal edges.

Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges. For sample values modified by both filtering of horizontal edges and filtering of vertical edges, filtering of horizontal edges is applied after filtering of vertical edges.

The deblocking filter process shall be applied to all prediction unit edges and transform unit edges of a picture, except edges at the boundary of the picture, any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc and any edges coinside with slice boundaries when loop_filter_across_slice_flag is equal to 0. For the transform units and prediction units with edges smaller than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

When disable_deblocking_filter_idc is not equal to 1, the deblocking filter process is invoked as the following ordered steps for each coding unit with the same order as decoding process.

1. The coding unit size nS is set equal to 1 << log2CUSize.



Deleted: is

Deleted: .

Deleted: 427

Deleted: and

Deleted: If disable_deblocking_filter_idc is equal to 1, the deblocking filter process is simply omitted and if loop_filter_across_slice_flag is equal to 0, the deblocking filter process is omitted at slice boundaries, otherwise …

Deleted: apply

2. The variables FilterInternalEdgesFlag, FilterLeftCuEdgeFlag and FilterTopCuEdgeFlag are derived as follows.
   - The variable FilterInternalEdges is set equal to 1.
   - If the left boundary of current coding unit is the left boundary of the picture or if the left boundary of current coding unit is the left boundary of the slice and loop_filter_across_slice_flag is equal to 0, the variable FilterLeftCuEdgeFlag is set equal to 0, otherwise set equal to 1.
   - If the top boundary of current coding unit is the top boundary of the picture or if the top boundary of current coding unit is the top boundary of the slice and loop_filter_across_slice_flag is equal to 0, the variable FilterTopCuEdgeFlag is set equal to 0, otherwise set equal to 1.
3. All elements of two-dimensional array of size (nS)x(nS), horEdgeFlags and verEdgeFlags are initialized to zero.
4. The derivation process of transform unit boundary specified in subclause 8.6.1.1 are invoked with the luma location ( xB, yB ) set equal to ( 0, 0 ), the transform unit size log2TrafoSize set equal to log2CUSize and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeFlags and verEdgeFlags as outputs.
5. The derivation process of prediction unit boundary specified in subclause 8.6.1.2 are invoked with the coding unit size log2CUSize and the prediction partition mode PartMode as inputs, and the modified horEdgeFlags and verEdgeFlags as outputs.
6. The derivation process of the boundary filtering strength specified in subclause 8.6.1.3 is invoked with the luma location ( xC, yC ), the coding unit size log2CUSize, horEdgeFlags and verEdgeFlags as inputs and an array of size (2)x(nS)x(nS), bS as output.
7. The filtering process for coding unit specified in subclause 8.6.1.4 are invoked with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size log2CUSize and the array bS as inputs and the modified reconstructued picture as output.

#### 8.6.1.1 Derivation process of transform unit boundary

Inputs of this process are:

- a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current coding unit,
- a transform unit size log2TrafoSize,
- a variable trafoDepth.

Outputs of this process are:

- two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on split_transform_flag[ xB ][ yB ][ trafoDepth ], the following applies:

- If split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:
  1. The variable xB1 is set equal to xB + ( ( 1 << log2TrafoSize ) >> 1 ).
  2. The variable yB1 is set equal to yB + ( ( 1 << log2TrafoSize ) >> 1 ).
  3. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB, yB ), the variable log2TrafoSize1 set equal to log2TrafoSize – 1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
  4. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB1, yB ), the variable log2TrafoSize1 set equal to log2TrafoSize – 1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
  5. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB, yB1 ), the variable log2TrafoSize1 set equal to log2TrafoSize – 1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
  6. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB1, yB1 ), the variable log2TrafoSize1 set equal to log2TrafoSize – 1 and the variable trafoDepth1

set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.

– Otherwise (split_transform_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following applies:
  – If yB is equal to zero, horEdgeFlags[ xB + k ][ yB ] is set equal to FilterTopCuEdgeFlag, otherwise horEdgeFlags[ xB + k ][ yB ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoSize ) – 1.
  – If xB is equal to zero, verEdgeFlags[ xB ][ yB + k ] is set equal to FilterLeftCuEdgeFlag, otherwise verEdgeFlags[ xB ][ yB + k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoSize ) – 1.

#### 8.6.1.2 Derivation process of prediction unit boundary

Inputs of this process are:

– a variable log2CUSize specifying the coding unit size,

– a prediction partition mode PartMode.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on PartMode, the following applies:

– If PartMode is equal to PART_2NxN or PART_NxN, horEdgeFlags[ k ][ 1 << ( log2CUSize – 1 ) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART_Nx2N or PART_NxN, verEdgeFlags[ 1 << ( log2CUSize – 1 ) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

#### 8.6.1.3 Derivation process of boundary filtering strength

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit,

– a two-dimensional arrays of size (nS)x(nS), horEdgeFlags and verEdgeFlags.

Output of this process is an array of size (2)x(nS)x(nS), bS specifying the boundary filtering strength.

Let ( $xE_k$, $yE_j$ ) with k = 0..nE-1 and j = 0..nE-1 specify a set of edge sample locations where nE is set equal to ( ( 1 << log2CUSize ) >> 2 ), $xE_0 = 0$, $yE_0 = 0$, $xE_{k+1} = xE_k + 4$ and $yE_{j+1} = yE_j + 4$.

For ( $xE_k$, $yE_j$ ) with k = 0..nE-1 and j = 0..nE-1, the following applies.

– If horEdgeFlags[ $xE_k$ ][ $yE_j$ ] is equal to 1,
  – Set sample $p_0$ = recPicture[ xC + $xE_k$ ][ yC + $yE_j$ – 1 ] and $q_0$ = recPicture[ xC + $xE_k$ ][ yC + $yE_j$ ].
  – The variable filterDir is set equal to 1.
– Otherwise, if verEdgeFlags[ $xE_k$ ][ $yE_j$ ] is equal to 1,
  – Set sample $p_0$ = recPicture[ xC + $xE_k$ – 1 ][ yC + $yE_j$ ] and $q_0$ = recPicture[ xC + $xE_k$ ][ yC + $yE_j$ ].
  – The variable filterDir is set equal to 0.
– Depending on the value of filterDir, the variable bS[ filterDir ][ $xE_k$ ][ $yE_j$ ] is derived as follows.
  – If the block edge is also a coding unit edge and the following condition is true, the variable bS[ 0 ][ $xE_k$ ][ $yE_j$ ] is set equal to 4.
    – The sample $p_0$ or $q_0$ is in a coding unit coded with intra prediction mode
  – Otherwise, if the following condition is true, the variable bS[ filterDir ][ $xE_k$ ][ $yE_j$ ] is set equal to 3.
    – The sample $p_0$ or $q_0$ is in a coding unit coded with intra prediction mode
  – Otherwise, if the following condition is true, the variable bS[ filterDir ][ $xE_k$ ][ $yE_j$ ] is set equal to 2.
    – The sample $p_0$ or $q_0$ is in a transform unit which contains non-zero transform coefficient level.
  – Otherwise, if any of the following conditions are true, the variable bS[ filterDir ][ $xE_k$ ][ $yE_j$ ] is set equal to 1.

– The prediction unit containing sample $p_0$ has different reference pictures or a different number of motion vectors with the prediction unit containing the sample $q_0$.

NOTE – The determination of whether the reference pictures used for the two prediction are the same or different is based on which pictures are referenced, without regard to whether a prediction is formed using an index into list 0 or an index into list 1, and also without regard to whether or not the index position within a reference picture list is different or not.

– One motion vector is used to predict the prediction unit containing sample $p_0$, one motion vector is used to predict the prediction unit containing sample $q_0$, and the absolute difference between the horizontal or vertical component of the motion vector used is greater than or equal to 4 in units of quarter luma samples.

– [Ed.: (WJ) needs to be checked again whether this condition covers all 2-motion cases] Two motion vectors are used to predict the prediction unit containing sample $p_0$, two motion vectors are used to predict the prediction unit containing sample $q_0$, and at least one of the motion vector pairs corresponding the same reference pictures and the different boundary samples $p_0$ and $q_0$ satisfies the following condition:

1. The absolute difference between the horizontal or vertical component of a motion vector used in the prediction of the two prediction units is greater than or equal to 4 in units of quarter luma samples.

– Otherwise, the variable bS[ filterDir ][ $xE_k$ ][ $yE_j$ ] is set equal to 0.

#### 8.6.1.4 Filtering process for coding unit

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the coding unit size,

– an array bS specifying the boundary filtering strength.

Output of this process is:

– modified reconstruction of the picture.

The filtering process for luma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( log2CUSize – 3 ).
2. All elements of the three-dimensional array of size (2)x(nD)x(nD), dEdge are initialized to zero.
3. All elements of the three-dimensional array of size (2)x(nD)x(1<<log2CUSize), dSample are initialized to zero.
4. All elements of the three-dimensional array of size (2)x(nD)x(nD), bStrength ae initialized to zero.
5. For $xD_k$ set equal to xC+( k << 3 ), k=0..nD – 1, the following applies:
   – For $yD_m$ set equal to yC+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
     a. Boundary filtering strength bSVer is derived as follows:

        bSVer = Max( bS[ 0 ][ $xD_k$ ][ $yD_m$ + i ] ) for i = 0..7 (8-205)

     b. bStrength[1][k][m] is set equal to bSVer.
     c. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 1, and the boundary filtering strength bSVer as inputs and the decision dEdge[1][k][m] and an array dS of size (8) as outputs.
     d. dSample[1][k][(m<<3)+i] is set equal to dS[i] for i=0..7.
     e. Boundary filtering strength bSHor is derived as follows:

        bSHor = Max( bS[ 1 ][ $xD_k$ + i ][ $yD_m$ ] ) for i = 0..7 (8-206)

     f. bStrength[0][k][m] is set equal to bSHor.
     g. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set

Deleted: 428

Deleted: 428

equal to 0, the boundary filtering strength bSHor as inputs, the decision dEdge[0][k][m] and an array dS of size (8) as outputs.

h. dSample[0][m][(k<<3)+i] is set equal to dS[i] for i=0..7.

6. For $xD_k$ set equal to xC+( k << 3 ), k=0..nD - 1, the following applies:
   – For $yD_m$ set equal to yB+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
     a. dS[i] is set equal to dSample[1][k][(m<<3)+i] for i=0..7.
     b. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 1, the boundary filtering strength bStrength[1][k][m], the decision dEdge[1][k][m], and the array of size (8), dS as inputs and the modified luma picture buffer as outputs.
7. For $yD_m$ set equal to yC+( m << 3 ), m=0..nD - 1, the following applies:
   – For $xD_k$ set equal to xC+( k << 3 ), k=0..nD – 1, the following ordered steps apply:
     a. If $xD_k$ is equal to 0, the parameter xPOS is set equal to 1. If $xD_k$ is equal to xB+( ( nD – 1) << 3 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
     b. dS[i] is set equal to dSample[0][m][ (k << 3) + i ] for i = 0..7.
     c. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 0, the boundary filtering strength bStrength[0][k][m], the decision dEdge[0][k][m], and the array of size (8), dS, xPOS, $dS_L$[m][], $dE_L$[m], $bS_L$[m], and $t_{CL}$[m], as inputs and the modified luma picture buffer as output.
   – The elements of the two dimensional array of size (3)x(nD), $dS_L$ are set as follows. $dS_L$[m][0], $dS_L$[m][1], and $dS_L$[m][2], are set equal to dS[5], dS[6] and dS[7].
   – The elements of the array of size (nD), $dE_L$ are set as follows. $dE_L$[m] is set equal to dEdge[0][k][m].
   – The elements of the array of size (nD), $bS_L$ are set as follows. $bS_L$[m] is set equal to bStrength[0][k][m].
   – The elements of the array of size (nD), $t_C$ are set as follows. $t_{CL}$[m] is set equal to $t_c$.

The filtering process for chroma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( Max( log2CUSize, 4 ) – 4 ).
2. For $xD_k$ set equal to ( xC / 2 )+( k << 3 ), k=0..nD – 1, the following applies:
   – For $yD_m$ set equal to ( yC / 2)+( m << 2 ), m=0..nD*2 – 1, the following ordered steps apply:
     a. Boundary filtering strength bSVer is derived as follows:

$$bSVer = bS[\ 0\ ][\ xD_k*2\ ][\ yD_m*2\ ] \quad (8\text{-}207)$$

     b. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 1 and the boundary filtering strength bSVer as inputs and the modified chroma picture buffer as output.
     c. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 2 and the boundary filtering strength bSVer as inputs and the modified chroma picture buffer as output.
3. For $yD_m$ set equal to ( yC / 2 )+( m << 3 ), m=0..nD – 1, the following applies:
   – For $xD_k$ set equal to ( xC / 2 )+( k << 2 ), k=0..nD*2 – 1, the following ordered steps apply:
     a. If $xD_k$ is equal to 0, the parameter xPOS is set equal to 1. If $xD_k$ is equal to xB+( ( nD*2 – 1) << 2 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
     b. Boundary filtering strength bSHor is derived as follows:

$$bSHor = bS[\ 1\ ][\ xD_k*2\ ][\ yD_m*2\ ] \quad (8\text{-}208)$$

Deleted: 430

Deleted: 431

c. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 1 and the boundary filtering strength bSHor, xPOS, $bS_L[m]$ and $t_{CL}[m]$ as inputs and the modified chroma picture buffer as output.

d. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( $xD_k$, $yD_m$ ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 2 and the boundary filtering strength bSHor, xPOS, $bS_L[m]$ and $t_{CL}[m]$ as inputs and the modified chroma picture buffer as output.

#### 8.6.1.4.1 Decision process for luma block edge

Formatted: Heading 5

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,

– a variable verticalEdgeFlag,

– a variable bS specifying the boundary filtering strength,

Output of this process is:

– a variable dE containing a decision,

– one-dimensional array of size (8), dS containing decisions.

Let s' represent the luma sample array $recPicture_L$ of the current picture.

A variables β is specified as Table 8-14 with luma quantization parameter $qP_L$ as input.

Deleted: Table 8-13

A variable $t_C$ is specified as follows:

– If bS is greater than 2, the variable $t_C$ is specified specified as Table 8-14 with luma quantization parameter ( $qP_L$ + 4 ) as input,

Deleted: Table 8-13

– Otherwise (bS is equal or less than 2), the variable $t_C$ is specified specified as Table 8-14 with luma quantization parameter $qP_L$ as input.

Deleted: Table 8-13

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 2,5 are derived as follows:

$$q_{i,k} = s'[\ xC + xB + i, yC + yB + k\ ] \quad (8\text{-}209)$$

Deleted: 447

$$p_{i,k} = s'[\ xC + xB - i - 1, yC + yB + k\ ] \quad (8\text{-}210)$$

Deleted: 448

2. The variable d is derived as follows:

$$d = |\ p_{2,2} - 2*p_{1,2} + p_{0,2}\ | + |\ q_{2,2} - 2*q_{1,2} + q_{0,2}\ | + |\ p_{2,5} - 2*p_{1,5} + p_{0,5}\ | + |\ q_{2,5} - 2*q_{1,5} + q_{0,5}\ | \quad (8\text{-}211)$$

Deleted: 449

3. The variable dE is set equal to 0.
4. If bS is not equal to 0 and d is less than β, the following ordered steps apply:
   a. for each sample location ( xC + xB, yC + yB + k ), k = 0..7, the following ordered steps apply:
      a. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, the boundary filtering strength bS and the variables d, β and $t_C$ as inputs and a decision dSam as output.
      b. The variable dS[k] is set equal to dSam
   b. The variable dE is set equal to 1.

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 2,5 are derived as follows:

$$q_{i,k} = s'[\ xC + xB + k, yC + yB + i\ ] \quad (8\text{-}212)$$

$$p_{i,k} = s'[\ xC + xB + k, yC + yB - i - 1\ ] \quad (8\text{-}213)$$

2. The variable d is derived as follows:

$$d = |\ p_{2,2} - 2*p_{1,2} + p_{0,2}\ | + |\ q_{2,2} - 2*q_{1,2} + q_{0,2}\ | + |\ p_{2,5} - 2*p_{1,5} + p_{0,5}\ | + |\ q_{2,5} - 2*q_{1,5} + q_{0,5}\ | \quad (8\text{-}214)$$

3. The variable dE is set equal to 0.
4. If bS is not equal to 0 and d is less than β, the following ordered steps apply:
   a. For each sample location ( xC + xB + k, yC + yB ), k = 0..7, the following ordered steps apply:
      a. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, the boundary filtering strength bS and the variables d, β and $t_C$ as inputs and a decision dSam as output.
      b. The variable dS[k] is set equal to dSam.
   b. The variable dE is set equal to 1.

**8.6.1.4.2 Filtering process for luma block edge**

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,
– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,
– a variable verticalEdgeFlag,
– a variable bS specifying the boundary filtering strength,
– a variable dE containing a decision,
– one-dimensional array of size (8), dS containing decisions,
– a variable $bS_L$,
– a variable $t_{CL}$,

Output of this process is:

– modified reconstruction of the picture.

Let s' represent the luma sample array $recPicture_L$ of the current picture.

Depending on pcm_flag, a variable β is specified as follows:

– If pcm_flag is equal to 1, the variables β is specified as Table 8-14 with luma quantization parameter 0 as input.
– Otherwise, the variables β is specified as Table 8-14 with luma quantization parameter $qP_L$ as input.

A variable $t_C$ is specified as follows:

– If bS is greater than 2, the variable $t_C$ is specified as Table 8-14 with luma quantization parameter ( $qP_L$ + 4 ) as input,
– Otherwise (bS is equal or less than 2), the variable $t_C$ is specified as Table 8-14 with luma quantization parameter $qP_L$ as input.

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, the following ordered steps apply:
  1. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0..7 are derived as follows:

$$q_{i,k} = s'[\ xC + xB + i, yC + yB + k\ ] \quad (8\text{-}215)$$

$$p_{i,k} = s'[\ xC + xB - i - 1, yC + yB + k\ ] \quad (8\text{-}216)$$



Deleted: 435
Deleted: 436
Deleted: 437
Deleted: A
Deleted: Table 8-11
Formatted: Indent: Left: 0", Hanging: 0.2", Tab stops: 0.2", Left
Deleted: Table 8-11
Deleted: Table 8-11
Deleted: Table 8-11
Deleted: 432
Deleted: 433

2. If dE is not equal to 0, for each sample location ( xC + xB, yC + yB + k ), k = 0..7, the following ordered steps apply:
   a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, the decision dS[k], the boundary filtering strength bS and the variable $t_C$ as inputs and the number of filtered samples nD and the filtered sample values $p_i$' and $q_i$' as outputs.
   b. The filtered sample values $p_i$' and $q_i$' with i = 0..nD – 1 replace the corresponding samples inside the sample array s' as follows:

$$s'[\ xC + xB + i, yC + yB + k\ ] = q_i' \quad (8\text{-}217)$$

$$s'[\ xC + xB - i - 1, yC + yB + k\ ] = p_i' \quad (8\text{-}218)$$

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. If xPOS is equal to 1, the parameters $k_s$ and $k_e$ are set to -3 and 4 respectively. If xD is equal to 2, the parameters $k_s$ and $k_e$ are set to 0 and 4 respectively. Otherwise $k_s$ and $k_e$ are set to 0 and 7 respectively.
2. The sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = $k_s$..$k_e$ are derived as follows:

$$q_{i,k} = s'[\ xC + xB + k, yC + yB + i\ ] \quad (8\text{-}219)$$

$$p_{i,k} = s'[\ xC + xB + k, yC + yB - i - 1\ ] \quad (8\text{-}220)$$

3. If xPOS is less than 0 and $dE_L$ is not equal to 0, for each sample location ( xC + xB + k, yC + yB ), k = -3..-1, the following ordered steps apply:
   a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, decision $dS_L$[k+3], the boundary filtering strength $bS_L$ and the variable $t_{CL}$ as inputs and the number of filtered samples nD and the filtered sample values $p_i$' and $q_i$' as outputs.
   b. The filtered sample values $p_i$' and $q_i$' with i = 0..nD – 1 replace the corresponding samples inside the sample array s' as follows:

$$s'[\ xC + xB + k, yC + yB + i\ ] = q_i' \quad (8\text{-}221)$$

$$s'[\ xC + xB + k, yC + yB - i - 1\ ] = p_i' \quad (8\text{-}222)$$

5. If dE is not equal to 0, for each sample location ( xC + xB + k, yC + yB ), k = 0.. $k_e$ , the following ordered steps apply:
   a. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values $p_{i,k}$, $q_{i,k}$ with i = 0..3, decision dS[k], the boundary filtering strength bS and the variable $t_C$ as inputs and the number of filtered samples nD and the filtered sample values $p_i$' and $q_i$' as outputs.
   b. The filtered sample values $p_i$' and $q_i$' with i = 0..nD – 1 replace the corresponding samples inside the sample array s' as follows:

$$s'[\ xC + xB + k, yC + yB + i\ ] = q_i' \quad (8\text{-}223)$$

$$s'[\ xC + xB + k, yC + yB - i - 1\ ] = p_i' \quad (8\text{-}224)$$

#### 8.6.1.4.3 Filtering process for chroma block edge

[Ed.: (WJ) cIdx cannot be 0 here]

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture,
– a luma location ( xB, yB ) specifying the top-left chroma sample of the current block relative to the top left chroma sample of the current coding unit,
– a variable verticalEdgeFlag,
– a variable bS specifying the boundary filtering strength,
– a variable cIdx specifying the chroma component index.

Deleted: 435
Deleted: 436
Deleted: 437
Deleted: 438
Deleted: 459
Deleted: 460
Deleted: 461
Deleted: 462
Formatted: Heading 5

– a variable xPOS,

– a variable $bS_L$,

– a variable $t_{CL}$

Output of this process is:

– modified reconstruction of the picture.

Let s' be a variable specifying chroma sample array which is derived as follows.

– If cIdx is equal to 1, s' represents the chroma sample array $recPicture_{Cb}$ of the current picture.

– Otherwise (cIdx is equal to 2), s' represents the chroma sample array $recPicture_{Cr}$ of the current picture.

A variable $t_C$ is specified as follows:

– If bS is greater than 2, the variable $t_C$ is specified specified as Table 8-14 with luma quantization parameter ( $qP_L + 4$ ) as input,

– Otherwise (bS is equal or less than 2), the variable $t_C$ is specified as Table 8-14 with luma quantization parameter $qP_L$ as input.

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, for each sample location ( xC + xB, yC + yB + k ), k = 0..3, the following ordered steps apply:

1. The sample values $p_i$ and $q_i$ with i = 0..1 are derived as follows:

$$q_i = s'[\ xC + xB + i, yC + yB + k\ ] \quad (8\text{-}225)$$

$$p_i = s'[\ xC + xB - i - 1, yC + yB + k\ ] \quad (8\text{-}226)$$

2. If bS is greater than 2, the following ordered steps apply:

   a. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values $p_i$, $q_i$, with i = 0..1, the boundary filtering strength bS and the variable $t_C$ as inputs and the filtered sample values $p_0$' and $q_0$' as outputs.

   b. The filtered sample values $p_0$' and $q_0$' replace the corresponding samples inside the sample array s' as follows:

$$s'[\ xC + xB\ , yC + yB + k\ ] = q_0' \quad (8\text{-}227)$$

$$s'[\ xC + xB - 1, yC + yB + k\ ] = p_0' \quad (8\text{-}228)$$

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. If xPOS is equal to 1, the parameters $k_s$ and $k_e$ are set to -1 and 2 respectively. If xD is equal to 2, the parameters $k_s$ and $k_e$ are set to 0 and 2 respectively. Otherwise $k_s$ and $k_e$ are set to 0 and 3 respectively.

2. The sample values $p_i$ and $q_i$ with i = 0..1 and k = $k_s$..$k_e$ are derived as follows:

$$q_i = s'[\ xC + xB + k, yC + yB + i\ ] \quad (8\text{-}229)$$

$$p_i = s'[\ xC + xB + k, yC + yB - i - 1\ ] \quad (8\text{-}230)$$

3. If xPOS is less than 0, and if $bS_L$, is greater than 2, for each sample location ( xC + xB - 1, yC + yB ), the following ordered steps apply:

   a. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values $p_i$, $q_i$, with i = 0..1, the boundary filtering strength $bS_L$ and the variable $t_{CL}$ as inputs and the filtered sample values $p_0$' and $q_0$' as outputs.

   b. The filtered sample values $p_0$' and $q_0$' replace the corresponding samples inside the sample array s' as follows:

$$s'[\ xC + xB + k, yC + yB\ ] = q_0' \quad (8\text{-}231)$$

Deleted: Table 8-13

Deleted: Table 8-13

Deleted: 442

Deleted: 443

Deleted: 444

Deleted: 445

Deleted: 446

Deleted: 447

Deleted: 452

$s'[ xC + xB + k, yC + yB - i - 1 ] = p_i'$ (8-232) Deleted: 445

4. If bS is greater than 2, for each sample location ( xC + xB + k, yC + yB ), k = 0.. $k_e$, the following ordered steps apply:

   c. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values $p_i$, $q_i$, with i = 0..1, the boundary filtering strength bS and the variable $t_C$ as inputs and the filtered sample values $p_0'$ and $q_0'$ as outputs.

   d. The filtered sample values $p_0'$ and $q_0'$ replace the corresponding samples inside the sample array s' as follows:

$s'[ xC + xB + k, yC + yB ] = q_0'$ (8-233) Deleted: 448

$s'[ xC + xB + k, yC + yB - 1 ] = p_0'$ (8-234) Deleted: 449

#### 8.6.1.4.4 Decision process for a luma sample

[Ed: (WJ) no filtering when bS is equal to 0]

Inputs of this process are:

– sample values, $p_i$ and $q_i$ with i = 0..2,

– a variable bS specifying the boundary filtering strength,

– variables d, β and $t_C$.

Output of this process is:

– a variable dSam containing a decision

When the variable bS is not equal to 0, the following applies:

– If d is less than ( β >> 2 ), | $p_3 - p_0$ | + | $q_0 - q_3$ | is less than ( β >> 3 ) and | $p_0 - q_0$ | is less than ( 5*$t_C$ + 1 ) >> 1, dSam is set equal to 1.

– Otherwise, dSam is set equal to 0.

#### 8.6.1.4.5 Filtering process for a luma sample

Inputs of this process are:

– sample values, $p_i$ and $q_i$ with i = 0..3,

– a variable dSam containing a decision,

– a variable $t_C$.

Output of this process is:

– number of filtered samples nD,

– filtered sample values, $p_i'$ and $q_i'$ with i = 0..nD – 1,

Depending on dSam, the following applies:

– When the variable dSam is equal to 1, the following strong filtering applies while nD is set equal to 3:

$p_0' = Clip1_Y( ( p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4 ) >> 3 )$ (8-235) Deleted: 450

$p_1' = Clip1_Y( ( p_2 + p_1 + p_0 + q_0 + 2 ) >> 2 )$ (8-236) Deleted: 451

$p_2' = Clip1_Y( ( 2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4 ) >> 3 )$ (8-237) Deleted: 452

$q_0' = Clip1_Y( ( p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4 ) >> 3 )$ (8-238) Deleted: 453

$q_1' = Clip1_Y( ( p_0 + q_0 + q_1 + q_2 + 2 ) >> 2 )$ (8-239) Deleted: 454

$q_2' = Clip1_Y( ( p_0 + q_0 + q_1 + 3*q_2 + 2*q_3 + 4 ) >> 3 )$ (8-240) Deleted: 455

– Otherwise, the following weak filtering applies while nD is set equal to 2:

$\Delta = Clip3( -t_C, t_C, ( 13*( q0 – p0 ) + 4*( q1 – p1 ) - 5*( q2 – p0 ) + 16 ) >> 5 )$ (8-241)

$p_0' = Clip1_Y( p_0 + \Delta )$ (8-242)

$q_0' = Clip1_Y( q_0 - \Delta )$ (8-243)

$p_1' = Clip1_Y( p_1 + \Delta/2 )$ (8-244)

$q_1' = Clip1_Y( q_1 – \Delta/2 )$ (8-245)

Each of the filtered sample values, $p_i'$ with i = 0..nD-1, is substituted by the corresponding input sample value $p_i$ if all of the following conditions are true.

– $p_i$ is a sample of an I_PCM block.

– pcm_loop_filter_disable_flag value is equal to 1.

Similary, each of the filtered sample values, $q_i'$ with i = 0..nD-1, is substituted by the corresponding input sample value $q_i$ if all of the following conditions are true.

– $q_i$ is a sample of an I_PCM block.

– pcm_loop_filter_disable_flag value is equal to 1.

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it's better to skip the filtering itself for PCM samples since first filtering is actually not needed.]

**Table 8-14 – Derivation of threshold variables β and $t_C$ from input Q**

| **Q** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **β** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 7 | 8 |
| **$t_C$** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Q** | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| **β** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 |
| **$t_C$** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |
| **Q** | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | |
| **β** | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 | 64 | 64 | 64 | 64 | |
| **$t_C$** | 5 | 5 | 6 | 6 | 7 | 8 | 9 | 9 | 10 | 10 | 11 | 11 | 12 | 12 | 13 | 13 | 14 | 14 | |

#### 8.6.1.4.6 Filtering process for a chroma sample

[Ed: (WJ) no filtering when bS is equal or less than 2]

Inputs of this process are:

– sample values, $p_i$ and $q_i$ with i = 0..1,

– a variable bS specifying the boundary filtering strength.

– a variable $t_C$.

Output of this process is:

– The filtered sample values, $p_0'$ and $q_0'$.

When the variable bS is greater than 2, the filtered sample values $p_0'$ and $q_0'$ are derived by

$\Delta = Clip3( -t_C, t_C, ( ( ( ( q_0 – p_0 ) << 2 ) + p_1 – q_1 + 4 ) >> 3 ) )$ (8-246)

$p_0' = Clip1_C( p_0 + \Delta )$ (8-247)

$q_0' = Clip1_C( q_0 - \Delta )$ (8-248)

Deleted: 456
Deleted: 457
Deleted: 458
Deleted: 459
Deleted: 460
Formatted: Subscript
Formatted: Subscript
Formatted: Indent: Left: 0.2", Hanging: 0.2", Tab stops: 0.39", Left
Formatted: Subscript
Formatted: Subscript
Deleted: ¶
Deleted: 8-12
Deleted: 8-11
Deleted: 461
Deleted: 462
Deleted: 463

The filtered sample value, $p_0'$ is substituted by the corresponding input sample value $p_0$ if all of the following conditions are true.

– $p_0$ is a sample of an I_PCM block.

– pcm_loop_filter_disable_flag value is equal to 1.

Similary, the filtered sample value, $q_0'$ is substituted by the corresponding input sample value $q_0$ if all of the following conditions are true.

– $q_0$ is a sample of an I_PCM block.

– pcm_loop_filter_disable_flag value is equal to 1.

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it's better to skip the filtering itself for PCM samples since first filtering is actually not needed.]

**Formatted:** Normal, Indent: Hanging: 0.2", Tab stops: 0.39", Left + Not at 0.59" + 0.79" + 0.98"

**Formatted:** Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

### 8.6.2 Sample adaptive offset process

A sample adaptive offset process shall be conditionally performed after the completion of the deblocking filter process for the decoded picture.

This process is invoked when both sample_adaptive_offset_enabled_flag and sample_adaptive_offset_flag are equal to 1.

This process is performed on a region basis which is aligned with the LCU boundaries after the completion of the slice construction process prior to sample adaptive offset process for the entire decoded process.

The sample adaptive offset process is invoked with the region equal to entire slice here.

[Ed.: (WJ) invoking is needed]

#### 8.6.2.1 Modification process for sample adaptive offset

[Ed.: (WJ) recursive partitioning is invoked here]

##### 8.6.2.1.1 Modification process for luma samples

[Ed:. (WJ) offset is added here]

### 8.6.3 Adaptive loop filter process

An adaptive loop filtering process shall be conditionally performed on a treeblock basis after the completion of the sample adaptive offset process for the entire decoded picture, with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process.

This process is invoked when both adaptive_loop_filter_enabled_flag and adaptive_loop_filter_flag are equal to 1.

This process is performed on a coding unit basis after the completion of the slice construction process prior to adaptive loop filter process for the entire decoded slice, with all coding units in a slice processed in order of coding unit scan order.

Filter coefficients $c_L$ for luma samples and $c_C$ for chroma samples are derived by invoking the process specified in subclause 8.6.3.2 Filter index array fIdx for luma samples are derived by invoking the process specified in subclause 8.6.3.3 with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture and the coding unit size log2CUSize as inputs.

**Deleted:** 8.6.2.1

**Deleted:** ¶

**Deleted:** 8.6.2.2

When interpreting the luma samples of the coding unit as to be filtered, depending on pcm_flag, pcm_loop_filter_disable_flag and alf_cu_control_flag, the following applies.

– If pcm_loop_filter_disable_flag is equal to 1 and pcm_flag is equal to 1, the luma samples are not filtered.

– Otherwise, if alf_cu_control_flag is equal to 0, the luma samples are filtered,

**Deleted:** If

– Otherwise (alf_cu_control_flag is equal to 1), if AlfCuFlag[ xC ][ yC ] is equal to 1 where xC and yC are the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the luma samples are filtered.

– Otherwise, the luma samples are not filtered.

When interpreting the chroma samples of the coding unit as to be filtered, depending on pcm_flag, pcm_loop_filter_disable_flag and alf_chroma_idc, the following applies.

– If pcm_loop_filter_disable_flag is equal to 1 and pcm_flag is equal to 1, both chroma samples are not filtered.

– Otherwise, if alf_chroma_idc is equal to 1, the Cr samples are filtered.

– Otherwise, if alf_chroma_idc is equal to 2, the Cb samples are filtered.

– Otherwise, if alf_chroma_idc is equal to 3, both chroma samples are filtered.

– Otherwise (alf_chroma_idc is equal to 0), both chroma samples are not filtered.

For the luma samples of the coding unit interpreted as to be filtered, the filtering process for luma samples specified in subclause 8.6.3.4 is invoked with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size log2CUSize, and the filter index array fIdx as inputs and the output is the modified filtered picture, $recFiltPicture_L$.

For the chroma samples of the coding unit interpreted as to be filtered, the filtering process for chroma samples specified in subclause 8.6.3.5 is invoked with the chroma location ( xC/2, yC/2 ) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture, the coding unit size log2CUSize-1 and the chroma component index cIdx equal to 1 as inputs and the output is the modified filtered picture, $recFiltPicture_{Cb}$.

For the chroma samples of the coding unit interpreted as to be filtered, the filtering process for chroma samples specified in subclause 8.6.3.5 is invoked with the chroma location ( xC/2, yC/2 ) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture, the coding unit size log2CUSize-1 and the chroma component index cIdx equal to 2 as inputs and the output is the modified filtered picture, $recFiltPicture_{Cr}$.

[Ed.: (WJ) recPicture: deblocked/output picture and recFiltPicture: ALFed picture]

[Ed.: (WJ) depending adaptive_loop_filter_flag, recFiltPicture should be copied to recPicture in subclause 8.6]

### 8.6.3.1 Boundary padding process

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

– a variable cIdx specifying the chroma component index.

Output of this process is the padded luma sample array s''.

Depending on cIdx, the following applies:

– If cIdx is equal to 0, a luma sample array s' is set equal to $recPicture_L$, a variable alfTap is set equal to ( alf_length_luma_minus_5_div2 << 1 ) + 5, a variable nS is set equal to (1 << log2CUSize) and a variable nExtPixels is set equal to (alfTap >> 1).

– Otherwise, if cIdx is equal to 1, a chroma sample array s' is set equal to $recPicture_{Cb}$, a variable alfTap is set equal to ( alf_length_chroma_minus_5_div2 << 1 ) + 5, a variable nS is set equal to (1 << ( log2CUSize – 1)) and a variable nExtPixels is set equal to (alfTap >> 1).

– Otherwise (cIdx is equal to 2), a chroma sample array s' is set equal to $recPicture_{Cr}$, a variable alfTap is set equal to ( alf_length_chroma_minus_5_div2 << 1 ) + 5, a variable nS is set equal to (1 << ( log2CUSize – 1)) and a variable nExtPixels is set equal to (alfTap >> 1).

Depending on loop_filter_across_slice_flag, a varaible s'' specifying the padded luma sample array is derived as:

– If loop_filter_across_slice_flag is equal to 1, s''[ xC+x ][ yC+y ] is set equal to s'[ xC+x ][ yC+y ] for x = -nExtPixels..nS+nExtPixels-1 and y = -nExtPixels..nS+nExtPixels-1.

[Ed. (WJ): picture boundaries should be considered even for this case]

– Otherwise (loop_filter_across_slice_flag is equal to 0), s''[ xC+x ][ yC+y ] is derived by the following ordered steps:

1. s''[ xC+x ][ yC+y ] is set equal to s'[ xC+x ][ yC+y ] for x=0..nS – 1 and y=0..nS – 1.

2. If the left boundary of current block coinsides the slice,

s''[ xC+x ][ yC+y ] = s'[ xC ][ yC+y ] where x = -nExtPixels..-1 and y = 0..nS-1 (8-249)

Deleted: If

Deleted: 8.6.2.3

Deleted: 8.6.2.4

Deleted: 8.6.2.4

Formatted: Heading 4,Heading 4 Char1,Heading 4 Char Char

Formatted: Subscript

Formatted: Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: Not at 0.55"

Formatted: Subscript

Deleted: 462

3. If the right boundary of current block coinsides the slice,

s''[ xC+nS+x ][ yC+y ] = s'[ xC+nS-1 ][ yC+y ] where x = 0..nExtPixels-1 and y = 0..nS-1 (8-250)

4. If the top boundary of current block coinsides the slice,

s''[ xC+x ][ yC+y ] = s'[ xC+x ][ yC ] where x = 0..nS-1 and y = -nExtPixels..-1 (8-251)

5. If the bottom boundary of current block coinside the slice,

s''[ xC+ x ][ yC+nS+y ] = s'[ xC+x ][ yC+nS-1 ] where x = 0..nS-1 and y = 0..nExtPixels-1 (8-252)

6. If both the left and the top boundaries of current block coinside the slice and the current coding unit and its upper-left coding unit are not at the same slice,

s''[ xC+x ][ yC+y ] = s'[ xC ][ yC ] where x = -nExtPixels..-1 and y = -nExtPixels..-1 (8-253)

7. If both the right and the top boundaries of current block coinside the slice and the current coding unit and its upper-right coding unit are not at the same slice,

s''[ xC+nS+x ][ yC+y ] = s'[ xC+nS-1 ][ yC ] where x = 0..nExtPixels-1 and y = -nExtPixels..-1 (8-254)

8. If both the left and the bottom boundaries of current block coinside the slice and the current coding unit and its bottom-left coding unit are not at the same slice,

s''[ xC+x ][ yC+nS+y ] = s'[ xC ][ yC+nS-1 ] where x = -nExtPixels..-1 and y = 0..nExtPixels-1 (8-255)

9. If both the right and the bottom boundaries of current block coinside the slice and the current coding unit and its bottom-right coding unit are not at the same slice,

s''[ xC+nS+x ][ yC+nS+y ] = s'[ xC+nS-1 ][ yC+nS-1 ] where x = 0..nExtPixels-1 and y = 0..nExtPixels-1 (8-256)

### 8.6.3.2 Derivation process for filter coefficients

Outputs of this process are filter coefficients $c_L$ for the luma samples and filter coefficients $c_C$ for the chroma samples.

The luma filter coefficients $c_L$ with elements $c_L[ i ][ j ]$, i = 0..AlfNumFilters – 1, j = 0..AlfCodedLengthLuma – 1 is derived as follows:

– If alf_pred_method is equal to 0 or the value of i is equal to 0,

$c_L[ i ][ j ] = alf\_coeff\_luma[ i ][ j ]$ (8-257)

– Otherwise (alf_pred_method is equal to 1 and the value of i is greater than 1),

$c_L[ i ][ j ] = alf\_coeff\_luma[ i ][ j ] + c_L[ i – 1 ][ j ]$ (8-258)

Considering the symmetry of the filter, the luma filter coefficients $c_L$ with elements $c_L[ i ][ j ]$, i = 0..AlfNumFilters – 1 is derived as follows:

$c_L[ i ][ AlfLengthLuma – 1 ] = c_L[ i ][ AlfCodedLengthLuma – 1 ]$ (8-259)

$c_L[ i ][ AlfLengthLuma – 2 – j ] = c_L[ i ][ j ]$ for j = 0..AlfCodedLengthLuma – 2 (8-260)

The chroma filter coefficients $c_C$ with elements $c_C[ i ]$, i = 0..AlfCodedLengthChroma – 1 is derived as follows:

– If i is equal to AlfCodedLengthChroma – 1, the coefficient $c_C[i]$ is derived as

$c_C[ i ] = 255 – sum – alf\_coeff\_chroma[ i ]$ (8-261)

where

$sum = alf\_coeff\_chroma[ AlfCodedLengthChroma – 2 ] + \Sigma_j( alf\_coeff\_chroma[ j ] << 1 )$ (8-262)

with j = 0..AlfCodedLengthChroma – 3

Deleted: 462
Deleted: 462
Deleted: 462
Deleted: 462
Deleted: 462
Deleted: 462
Formatted: Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.2" + 0.55" + 0.83" + 1.1" + 1.38"
Deleted: 462
Deleted: ¶
Deleted: 464
Deleted: 465
Deleted: 466
Deleted: 467
Deleted: 468
Deleted: 469

– Otherwise (i is less than AlfCodedLengthChroma – 1),

$c_C[ i ] = alf\_coeff\_chroma[ i ]$ (8-263)

Considering the symmetry of the filter, the chroma filter coefficients $c_C$ with elements $c_C[ i ]$, i = 0..AlfLengthChroma – 1 is derived as follows:

$c_C[ i ][ AlfLengthChroma – 1 ] = c_C[ i ][ AlfCodedLengthChroma – 1 ]$ (8-264)

$c_C[ i ][ AlfLengthChroma – 2 – j ] = c_C[ i ][ j ]$ for j = 0..AlfCodedLengthChroma – 2 (8-265)

### 8.6.3.3 Derivation process for filter index array for luma samples

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

Output of this process is the two-dimensional filter index array of (nS)x(nS), fIdx.

A variable nS is set equal to ( 1 << log2CUSize ).

The boundary padding process specified in subclause 8.6.3.1 is invoked with the luma location ( xC, yC ), the size of coding unit log2CUSize and the chroma component index cIdx set equal to 0, and the output is assigned to the luma sample array s''. [Ed. (WJ): s'' is now a picture-size array, but actually CU size + appropriate margin is enough]

The filter index array fIdx is specified in the follows:

– When alf_region_adaptation_flag is equal to 1, the following ordered steps apply.

1. The variables xIdx and yIdx are derived as

regionTab[16] = { 0, 1, 4, 5, 15, 2, 3, 6, 14, 11, 10, 7, 13, 12, 9, 8 } (8-266)
offset = 1 << (Log2MaxCUSize – 1) (8-267)
xInterval = ( ( ( $PicWidthInSamples_L$ + offset ) >> Log2MaxCUSize ) + 1 ) >> 2 (8-268)
yInterval = ( ( ( $PicHeightInSamples_L$ + offset ) >> Log2MaxCUSize ) + 1 ) >> 2 (8-269)
xIdx = Min( 3, Floor( ( xC + x ) / ( xInterval << Log2MaxCUSize ) ) ) (8-270)
yIdx = Min( 3, Floor( ( yC + y ) / ( yInterval << Log2MaxCUSize ) ) ) (8-271)

2. The filter index fIdx[ x, y ] with x, y = 0..(nS)-1 is derived as

fIdx[ x ][ y ] = regionTab[ ( yIdx << 2 ) + xIdx ] (8-272)

– Otherwise (alf_region_flag is equal to 0), the following ordered steps apply.

3. The variables varTempH[ x ][ y ], varTempV[ x ][ y ] and varTemp1[ x ][ y ] with x, y = -1..(nS)+1 is derived as

varTempH[ x ][ y ] = | ( s''[ xC+x, yC+y ] << 1 ) – s''[ xC+x-1, yC+y ] – s''[ xC+x+1, yC+y ] | (8-273)
varTempV[ x ][ y ] = | ( s''[ xC+x, yC+y ] << 1 ) – s''[ xC+x, yC+y-1 ] – s''[ xC+x, yC+y+1 ] | (8-274)
varTemp1[ x ][ y ] = varTempH[ x ][ y ] + varTempV[ x ][ y ] (8-275)

4. The variable varTemp2[ x, y ] with x, y = 0..(nS)-1 is derived as

varTemp2[ x ][ y ] = $\Sigma_i\Sigma_j$ varTemp1[ x + i ][ y + j ] with i, j = -1..1 (8-276)

5. The variables varTemp3[ x, y ], varTempH1[ x, y ] and varTempV1[ x, y ] with x, y = 0..( (nS) – 1 )>>2 are derived as

varTemp3[ x ][ y ] = ($\Sigma_i\Sigma_j$ varTemp2[ (x << 2 ) + i ][ (y << 2) + j ]) >> 4 with i, j = 0..3 (8-277)
varTempH1[ x ][ y ] = $\Sigma_i\Sigma_j$ varTempH[ (x << 2 ) + i ][ (y << 2) + j ] with i, j = 0..3 (8-278)
varTempV1[ x ][ y ] = $\Sigma_i\Sigma_j$ varTempV[ (x << 2 ) + i ][ (y << 2) + j ] with i, j = 0..3 (8-279)

6. The variable direction is derived as

– If varTempV1[ x >> 2 ][ y >> 2 ] is greater than varTempH1[ x >> 2 ][ y >> 2 ] << 1,



Deleted: 470
Deleted: 471
Deleted: 472
Deleted: <#>¶
Deleted: and s' represents the luma sample array $recPicture_L$ of the current picture.
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474
Deleted: 474

direction = 1

– Otherwise, if varTempH1[ x >> 2 ][ y >> 2 ] is greater than varTempV1[ x >> 2 ][ y >> 2 ] << 1,

direction = 2

– Otherwise,

direction = 0

7. The variable avgvar is derived as

varTab[16] = { 0, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4 } (8-280)

avgVar = Clip3( 0, 15, (varTemp3[ x >> 2 ][ y >> 2 ] * 114 ) >> (3 + BitDepthY) ) (8-281)

8. The filter index fIdx[ x, y ] with x, y = 0..(nS)-1 is derived as

fIdx[ x ][ y ] = Clip3( 0, 4, varTab[avgVar] ) + 5 * direction (8-282)

#### 8.6.3.4 Filtering process for luma samples

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit,

– a filter index array of (nS)x(nS), fIdx.

Output of this process is the filtered reconstruction of luma picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the luma location ( xC, yC ), the size of coding unit log2CUSize and the chroma component index cIdx set equal to 0, and the output is assigned to the luma sample array s''. [Ed. (WJ): s'' is now a picture-size array, but actually CU size + appropriate margin is enough]

A variable nS is set equal to ( 1 << log2CUSize ) and a variable alfTap is set equal to ( alf_length_luma_minus_5_div2 << 1 ) + 5.

Each sample of luma picture recFiltPicture$_L$[ xC + x ][ yC + y ] with x, y = 0..(nS)-1, is derived as follows.

$$recFiltPicture_L[xC+x][yC+y] = c_L[fIdx[x][y]][N] + \sum_{i=0}^{N-1}\left(s''[\,xC+x+horPos[i],\, yC+y+verPos[i]\,] * c_L[fIdx[x][y]][i]\right) \quad (8\text{-}283)$$

where N is set equal to AlfNumCoeffLuma-1 and horPos[i] and verPos[i] are specified in Table 8-15 and Table 8-16, respectively.

**Table 8-15 – Specification of horPos[ i ] according to alfTap for adaptive loop filter process of luma samples**

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alfTap = 5 | 0 | -1 | 0 | 1 | -2 | -1 | 0 | 1 | 2 | -1 | 0 | 1 | 0 | - | - | - | - | - | - | - |
| alfTap = 7 | 0 | -1 | 0 | 1 | -2 | -1 | 0 | 1 | 2 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | -2 | -1 | 0 | 1 |
| alfTap = 9 | -1 | 0 | 1 | -2 | -1 | 0 | 1 | 2 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | -4 | -3 | -2 | -1 | 0 |

Deleted: 475
Deleted: 475
Deleted: 504
Deleted: Let s' be a variable specifying luma sample array recPicture$_L$,…
Deleted: ¶
Deleted: ¶
Deleted: 476
Deleted: Table 8-14
Deleted: Table 8-15
Deleted: 8-13
Deleted: 8-12

| i | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alfTap = 5 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| alfTap = 7 | 2 | -1 | 0 | 1 | 0 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| alfTap = 9 | 1 | 2 | 3 | 4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | -2 | -1 | 0 | 1 | 2 | -1 | 0 | 1 |

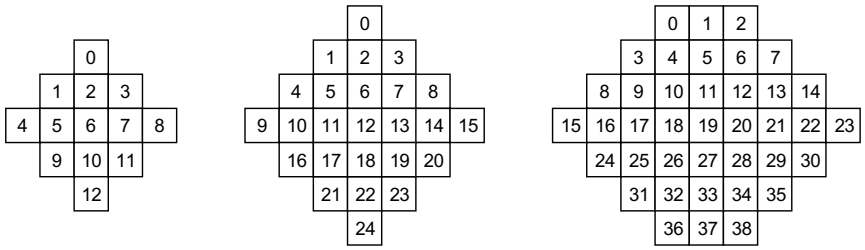
**Table 8-16 – Specification of verPos[ i ] according to alfTap for adaptive loop filter process of luma samples**

Deleted: 8-14

Deleted: 8-13

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alfTap = 5 | -2 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | | | | | | | |
| alfTap = 7 | -3 | -2 | -2 | -2 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| alfTap = 9 | -3 | -3 | -3 | -2 | -2 | -2 | -2 | -2 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 |

| i | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alfTap = 5 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| alfTap = 7 | 1 | 2 | 2 | 2 | 3 | - | - | - | - | - | - | - | - | - | - | - | - | - | - |
| alfTap = 9 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 |




(a) alfTap = 5 (a) alfTap = 7 (a) alfTap = 9

**Figure 8-8 Mapping between geometric position and luma adaptive loop filter index according to alfTap (informative)**

#### 8.6.3.5 Filtering process for chroma samples

Inputs of this process are:

– a chroma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left chroma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit.

– a variable cIdx specifying the chroma component index.

Output of this process is the filtered reconstruction of chroma picture.

The boundary padding process specified in subclause 8.6.3.1 is invoked with the chroma location ( xC, yC ), the size of coding unit log2CUSize and the chroma component index cIdx, and the output is assigned to the luma sample array s''. [Ed. (WJ): s'' is now a picture-size array, but actually CU size + appropriate margin is enough]

A variable nS is set equal to ( 1 << log2CUSize ) and a variable alfTapChroma is set equal to ( alf_length_chroma_minus_5_div2 << 1 ) + 5.

Deleted: Let s' be a variable specifying chroma sample array which is derived as follows.¶
–→ If cIdx is equal to 1, s' represents the chroma sample array $recPicture_{Cb}$ of the current picture and recFiltPicture represents the filtered reconstruction of chroma picture $recFiltPicture_{Cb}$.¶
–→ Otherwise (cIdx is equal to 2), s' represents the chroma sample array $recPicture_{Cr}$ of the current picture and recFiltPicture represents the filtered reconstruction of chroma picture $recFiltPicture_{Cr}$.¶

Filtered samples of chroma picture recFiltPicture[ xC + x ][ yC + y ] with x, y = 0..(nS)-1, are derived as follows.

$$recFiltPicture[xC + x][yC + y] = c_C[N] + \sum_{i=0}^{N-1} (s''[\, xC + x + horPos[i], yC + y + verPos[i]\,] * c_C[i]) \quad (8\text{-}284)$$

where

N = AlfNumCoeffChroma – 1, (8-285)

horPos[ i ] = ( i % alfTapChroma ) – ( alfTapChroma >> 1 ), and (8-286)

verPos[ i ] = ( i / alfTapChroma ) – ( alfTapChroma >> 1 ) (8-287)

# 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++ )                    (9-1)
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$codeNum = 2^{leadingZeroBits} - 1 + read\_bits(\, leadingZeroBits \,) \quad (9\text{-}2)$$

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 |
| … | … |



Deleted: 477
Deleted: 478
Deleted: 479
Deleted: 480
Deleted: ¶
Deleted: Table 9-1
Deleted: Table 9-1
Deleted: Table 9-1
Deleted: 9-1
Deleted: 9-1

Table 9-2 illustrates explicitly the assignment of bit strings to codeNum values.

Deleted: Table 9-2

**Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)**

Deleted: 9-2

Deleted: 9-2

| 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) [Ed. (TW): insert text]

– 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 )                                    (9-3)
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.

Deleted: Table 9-3

**Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)**

Deleted: 9-3

Deleted: 9-3

| 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 ) |

## 9.2 CAVLC parsing process for slice data

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to ce(v).

Inputs to this process are bits from the RBSP.

Outputs of this process are syntax element values.

### 9.2.1 Parsing process for VLC codes

Inputs to this process are bits from the RBSP and the parameter vlcNum specifying the VLC code. If vlcNum is equal to 14, inputs to this process additionally include a parameter cMax.

Output of this process is the variable codeNum specifying an integer value to be used when deriving syntax elements.

If vlcNum is not equal to 8 or 11, and is less than 14, the parsing process for the VLC codes 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++ )                                    (9-4)
    b = read_bits( 1 )
```

Depending on the value of vlcNum, the value of codeNum is assigned as follows.

– If vlcNum is equal to 0,

  – If leadingZeroBits is less than or equal to 6,

    codeNum = leadingZeroBits (9-5)

  – Otherwise (leadingZeroBits is greater than 6),

    numBits = leadingZeroBits – 6 (9-6)
    codeNum = 5 + (1<< numBits) + read_bits( numBits )

– Otherwise, if vlcNum is less than 5,

  – If leadingZeroBits is less than or equal to 6,

    codeNum = (leadingZeroBits << vlcNum) + read_bits( vlcNum ) (9-7)

  – Otherwise (leadingZeroBits is greater than 6),

    numBits = leadingZeroBits – 6 + vlcNum (9-8)
    codeNum = 5*(1<<vlcNum) + (1<< numBits) + read_bits( numBits)

– Otherwise, if vlcNum is less than 8,

  codeNum = (leadingZeroBits << (vlcNum - 4)) + read_bits( vlcNum - 4) (9-9)

– Otherwise, if vlcNum is equal to 8,

```
b =  read_bits(1)                                                    (9-10)
codeNum = b ? 2 - read_bits(1) : 0
```

– Otherwise, if vlcNum is equal to 9,

  – If leadingZeroBits is equal to 0,

```
b =  read_bits(1)
if (b) {
    b =  read_bits(1)
    if (b) {
        codeNum = 3 + read_bits(3)                                   (9-11)
    }
    else{
        b =  read_bits(1)
        codeNum = b ? 1 + read_bits(1) : 0
    }
}
```

  – Otherwise (leadingZeroBits is greater than 0),

```
codeNum = (leadingZeroBits << 4) +  read_bits( 4) + 11               (9-12)
```

– Otherwise, if vlcNum is equal to 10,

```
codeNum = (1<< leadingZeroBits) − 1 + read_bits( leadingZeroBits )   (9-13)
```

– Otherwise, if vlcNum is equal to 11,

```
codeNum = read_bits(3)
if( codeNum ) {
    b = read_bits(1)                                                 (9-14)
    codeNum = (codeNum << 1) + b -1
}
```

– Otherwise, if vlcNum is equal to 12,

```
codeNum = (leadingZeroBits << 6) + read_bits( 6 )                    (9-15)
```

– Otherwise, if vlcNum is equal to 13,

```
codeNum = (leadingZeroBits << 4) + read_bits( 4 )                    (9-16)
```

– Otherwise, if vlcNum is equal to 14,

```
leadingZeroBits = −1
for( b = 0; !b && leadingZeroBits < cMax; leadingZeroBits++ )        (9-17)
    b = read_bits( 1 )
codeNum = leadingZeroBits
```

– Otherwise, if vlcNum is equal to 15, codeNum is determined according to Table 9-4.

– Otherwise, if vlcNum is equal to 16

```
codeNum = read_bits(3)
if( codeNum >= 6)
    codeNum = 7 - codeNum
else if (codeNum > 0) {                                              (9-18)
    b = read_bits( 1 )
    codeNum = 13 – (codeNum << 1) - b
}
else {
    b = read_bits( 2 )
    codeNum = 15 – b
}
```

– Otherwise, if vlcNum is equal to 17

```
codeNum = read_bits(3)
if( codeNum ) {                                                      (9-19)
    b = read_bits(1)
```

```
        codeNum = 16 -  (codeNum << 1) - b
    }
```

– Otherwise, if vlcNum is equal to 18

```
    codeNum = read_bits(3)
    if( codeNum && codeNum < 4) {
        b = read_bits(1)
        codeNum = 8 -  (codeNum << 1) - b
    } else if (codeNum < 6) {
        b = read_bits(2)
        codeNum = 30 -  (codeNum << 2) - b                    (9-20)
    } else {
        b = read_bits(3)
        codeNum =  (codeNum << 3) + b
        if (codeNum < 62)
            codeNum = 76 – codeNum
        else {
            b = read_bits(1)
            codeNum = 156 - (codeNum << 1) – b
        }
    }
```

– Otherwise, if vlcNum is equal to 19

```
    codeNum = read_bits(3)
    if( codeNum && codeNum < 3) {
        b = read_bits(1)
        codeNum = 6 -  (codeNum << 1) - b
    } else {                                                  (9-21)
        b = read_bits(2)
        codeNum =  (codeNum << 2) + b
        if (codeNum < 25)
            codeNum = 29 – codeNum
        else {
            b = read_bits(1)
            codeNum = 81 - (codeNum << 1) – b
        }
    }
```

**Table 9-4 – codeNum and codeword with vlcNum equal to 15**

Deleted: 9-4

Deleted: 9-4

| codeNum | Codeword |
|---|---|
| 0 | 0 |
| 1 | 11 |
| 2 | 1001 |
| 3 | 1011 |
| 4 | 1000 |
| 5 | 10100 |
| 6 | 101010 |
| 7 | 101011 |

### 9.2.2 Initialisation process

Outputs of this process are initialised CAVLC internal variables.

The processes in subclauses 9.2.2.2 through 9.2.2.6 are invoked when starting the parsing of the slice data of a slice in subclause 7.3.4.

#### 9.2.2.1 Counter initialisation for codeword index adaptive mapping

Outputs of this process are initial values of counters that are used for parsing a syntax element.

If the number of counters used for a syntax element is N (N>0), values of the N counters are all initialized to 0. For each set of counters used for parsing a syntax element, there is an associated value, counterSum, to store the number of counter increments since its previous reset. The value of counterSum is also initialized to 0.

#### 9.2.2.2 Initialisation process for lastPosVlcNumIndex

Outputs of this process are initial values of the variable array lastPosVlcNumIndex.

The variable array lastPosVlcIndex is initialized as follows.

lastPosVlcNumIndex [ blockType ] = 0, for blockType = 0...7 (9-22)

Deleted: 21

#### 9.2.2.3 Initialisation process for lastPosTable

Outputs of this process are initial values of the variable array lastPosTable.

The variable array lastPosTable is initialized as specified in Table 9-5.

**Table 9-5 – Specification of initial values of lastPosTable[tableNum][codeNum]**

Deleted: 9-5

Deleted: 9-5

| | codeNum | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **tableNum** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 0 | 0 | 1 | 2 | 3 | 5 | 4 | 7 | 6 | 11 | 8 | 14 | 9 | 16 | 15 | 10 | 13 |
| 1 | 0 | 1 | 2 | 5 | 7 | 4 | 6 | 3 | 9 | 12 | 11 | 8 | 10 | 14 | 13 | 15 |
| 2 | 0 | 1 | 2 | 5 | 7 | 4 | 6 | 3 | 9 | 12 | 11 | 8 | 10 | 14 | 13 | 15 |

| | codeNum | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **tableNum** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| 0 | 12 | 17 | 18 | 19 | 22 | 31 | 23 | 21 | 27 | 20 | 25 | 30 | 24 | 26 | 28 | 29 |
| 1 | 16 | 17 | 21 | 22 | 25 | 18 | 28 | 23 | 30 | 26 | 20 | 19 | 27 | 24 | 29 | 31 |
| 2 | 16 | 17 | 21 | 22 | 25 | 18 | 28 | 23 | 30 | 26 | 20 | 19 | 27 | 24 | 29 | 31 |

#### 9.2.2.4 Initialisation process for splitPredPartModeTable

Outputs of this process are initial values of the variable array splitPredPartModeTable and associated counters.

The variable array splitPredPartModeTable is initialized as follows:

– For each cuDepth with a value from 0 to (Log2MaxCUSize – Log2MinCUSize), set splitPredPartModeTable[cuDepth][codeNum] equal to codeNum, with codeNum = 0…6.

For each cuDepth with a value from 0 to (Log2MaxCUSize – Log2MinCUSize), there are four counters used for codeword index adaptive mapping. These counters are initialized according to subclause 9.2.2.1.

#### 9.2.2.5 Initialisation process for intraModeTable

Outputs of this process are initial values of the variable array intraModeTable.

The variable array intraModeTable is initialized as specified in Table 9-6.

**Table 9-6 – Specification of intraModeTable[k][codeNum]**

Deleted: 9-6

Deleted: 9-6

| | codeNum | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **k** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| 0 | 0 | 15 | 11 | 10 | 13 | 7 | 9 | 4 | 14 | 2 | 3 | 6 | 8 | 5 | 12 | 1 | |
| 1 | 14 | 10 | 9 | 0 | 13 | 7 | 2 | 8 | 3 | 12 | 6 | 4 | 11 | 1 | 5 | | |
| 2 | 2 | 0 | 29 | 30 | 20 | 1 | 21 | 28 | 15 | 7 | 16 | 8 | 11 | 31 | 22 | 19 | 32 |
| 3 | 2 | 1 | 28 | 0 | 29 | 20 | 27 | 19 | 51 | 21 | 7 | 14 | 10 | 11 | 30 | 31 | 18 |

| | codeNum | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **k** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** |
| 2 | 10 | 27 | 14 | 4 | 18 | 12 | 23 | 17 | 24 | 26 | 5 | 25 | 5 | 9 | 3 | 13 | |
| 3 | 16 | 26 | 8 | 17 | 4 | 22 | 9 | 23 | 6 | 25 | 13 | 24 | 12 | 3 | 5 | | |

#### 9.2.2.6 Initialisation process for cbpSplitTransTable

Outputs of this process are initial values of the variable array cbpSplitTransTable and associated counters.

The values of the variable array cbpSplitTransTable[k][n] are initialized as specified in Table 9-7 where the values of k = 0...5 and n = 0,1. The number of counters used in each case for codeword index adaptive mapping is defined in Table 9-16. These counters are initialized according to subclause 9.2.2.1.

**Table 9-7 – Specification of initial values of cbpSplitTransTable[flagPattern-8][n][codeNum]**

Deleted: 9-7

Deleted: 9-7

| | k (n = 0) | | | | | | k (n = 1) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **codeNum** | **0** | **1** | **2** | **3** | **4** | **5** | **0** | **1** | **2** | **3** | **4** | **5** |
| 0 | 14 | 2 | 3 | 6 | 4 | 4 | 0 | 2 | 0 | 0 | 0 | 0 |
| 1 | 13 | 3 | 1 | 4 | 7 | 7 | 1 | 3 | 2 | 5 | 4 | 4 |
| 2 | 12 | 1 | 0 | 5 | 0 | 0 | 3 | 0 | 1 | 4 | 5 | 5 |
| 3 | 10 | 0 | 2 | 7 | 5 | 5 | 7 | | 3 | 7 | 6 | 6 |
| 4 | 6 | | | 0 | 6 | 6 | 2 | | | 6 | 1 | 1 |
| 5 | 11 | | | 1 | 1 | 1 | 4 | | | 2 | 7 | 7 |
| 6 | 9 | | | 2 | 2 | 2 | 5 | | | 3 | 2 | 2 |
| 7 | 8 | | | 3 | 3 | 3 | 8 | | | 1 | 3 | 3 |
| 8 | 5 | | | | | | 9 | | | | | |
| 9 | 4 | | | | | | 11 | | | | | |
| 10 | 2 | | | | | | 6 | | | | | |
| 11 | 7 | | | | | | 10 | | | | | |
| 12 | 3 | | | | | | 12 | | | | | |
| 13 | 1 | | | | | | 13 | | | | | |
| 14 | 0 | | | | | | 14 | | | | | |

### 9.2.3 Codeword index mapping update process

Inputs to this process are a VLC codeword index mapping table indexMappingTable that provides the mapping between VLC codeword indexes and syntax element values, a codeword index value codeNum, and a value counterNum that specifies the number of counters used. If counterNum is not equal to 0, inputs additionally include an array counterArray indexed from 0 to (counterNum-1) that saves the value of counters, a value counterSum that saves the number of all counter increments since its previous reset.

Outputs of this process are indexMappingTable, and if counterNum is not equal to 0 counterArray and counterSum with updated values.

The process is specified in the following ordered steps:

– Syntax element value syntaxVal is set equal to indexMappingTable [codeNum].

– If codeNum is not less than 1, codeNumPre is set equal to (codeNum-1). Otherwise, codeNumPre is set equal to codeNum. Syntax element value syntaxValPre is set equal to indexMappingTable [codeNumPre].

– If codeNum is greater or equal to counterNum, the values of indexMappingTable[codeNumPre] and indexMappingTable[codeNum] are exchanged with each other. No further step is carried out. Otherwise

  – The value of counterArray[codeNum] is increased by 1.

– If counterArray[codeNum] >= counterArray[codeNumPre], the counter values of counterArray[codeNum] and counterArray[codeNumPre] are swapped, and the values of indexMappingTable [codeNumPre] and indexMappingTable [codeNum] are also swapped.

– If counterSum is not less than 15, the value of counterSum is reset to 0. For i = 0…(counterNum-1), counterArray[i] is set equal to (counterArray[i]>>1).

– Otherwise, the value of counterSum is increased by 1.

#### 9.2.4 Parsing process for transform coefficient syntax elements

This process is invoked when parsing syntax elements with descriptor equal to ce(v) in subclause 7.3.11 and 7.3.12 and when entropy_coding_mode_flag is equal to 0.

Inputs to this process are bits from RBSP, and the variables log2TrafoSize and cIdx, and the variable arrays lastPosVlcNumIndex and lastPosTable.

Outputs of this process are CAVLC syntax elements last_pos_level_one, level_minus2_and_sign, run_level_one, and level, and the variable arrays lastPosTable and lastPosVlcNumIndex.

##### 9.2.4.1 Parsing process for last_pos_level_one

Inputs to this process are bits from RBSP and the variable arrays lastPosVlcNumIndex and lastPosTable.

Output of this process is the syntax element last_pos_level_one, the variables trOne and vlcNumLevel, and the variable arrays lastPosVlcNumIndex and lastPosTable.

The parsing process for last_pos_level_one is specified as follows.

– The variable N is set equal to (1 << log2TrafoSize) >> (cIdx > 0 ? 2 : 0).

– If N is greater than 4, last_pos_level_one is derived in the following ordered steps:

1. The variable blockType is derived as

```
blockType = (cIdx == 0 ? (PredMode==MODE_INTRA ? 0 : slice_type + 1) + (N > 8 ? 5 : 2) : cIdx – 1)   (9-23)
```

2. The variable vlcNumIndex is derived as

```
vlcNumIdx = Min(16, lastPosVlcNumIndex[blockType])   (9-24)
```

3. The variable vlcNum is derived as

```
vlcNum = lastPosVlcNumTable[blockType][vlcNumIdx]   (9-25)
```

   The values of the array lastPosVlcNumTable are shown in Table 9-8.

4. The parsing process described in subclause 9.2.1 is invoked with vlcNum as input and the variable codeNum as output.

5. The variable lastPos is derived as follows

```
if (codeNum <= N+N*N)
    if (codeNum != 0 && (codeNum & (N-1)) = = 0)
        lastPos = (codeNum >> Log2(N)) – 1   (9-26)
    else
        lastPos = codeNum – (codeNum >> Log2(N))
else
    lastPos = codeNum – N*N
```

6. The variable levelGreaterThanOneFlag is derived as

```
levelGreaterThanOneFlag = (codeNum > (N + N*N)) | | (codeNum != 0 && (codeNum & (N-1)))   (9-27)
```

7. The syntax element last_pos_level_one is derived as

```
last_pos_level_one = levelGreaterThanOneFlag ? codeNum + N : codeNum   (9-28)
```

8. The variable array lastPosVlcNumIndex[blockType] is updated as follows.

```
if ((N = = 8 ? codeNum : codeNum>>2) < lastPosVlcNumIndex[blockType])
    lastPosVlcNumIndex[blockType] -= 1   (9-29)
```

```
else if ((N = = 8 ? codeNum : codeNum>>2)  > lastPosVlcNumIndex[blockType])
    lastPosVlcNumIndex[blockType] +=  1
```

– Otherwise, (N equal to 4), last_pos_level_one is derived in the following ordered steps:

1. The parsing process described in subclause 9.2.1 is invoked with 2 as input and the variable codeNum as output.
2. The variable tableNum is derived as

```
tableNum = (PredMode==MODE_INTRA | | cIdx > 0) ? 0 : slice_type + 1
```
(9-30)

3. The syntax element last_pos_level_one is derived as

```
last_pos_level_one = lastPosTable[tableNum][codeNum]
```
(9-31)

4. The variable array lastPosTable[tableNum] is updated by invoking process 9.2.3 with lastPosTable[tableNum], codeNum and 0 as inputs.

– The variable trOne is derived as

```
trOne = ! levelGreaterThanOneFlag
```
(9-32)

– The variable vlcNumLevel is set equal to 0

**Table 9-8 – Specification of lastPosVlcNumTable[blockType][vlcNumIdx]**

| | vlcNumIdx | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **blockType** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| 0 | 10 | 10 | 10 | 10 | 2 | 2 | 2 | 7 | 9 | 9 | 9 | 9 | 9 | 4 | 4 | 4 | 4 |
| 1 | 10 | 10 | 10 | 10 | 10 | 2 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 4 | 4 | 4 | 4 |
| 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 4 | 4 | 13 |
| 3 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 13 |
| 4 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 13 |
| 5 | 10 | 10 | 10 | 4 | 4 | 4 | 4 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| 6 | 10 | 10 | 10 | 10 | 4 | 4 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| 7 | 10 | 10 | 10 | 10 | 4 | 4 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |

Deleted: 9-8

Deleted: 9-8

#### 9.2.4.2 Parsing process for level_minus2_and_sign

Input to this process are bits from slice data.

Output of this process is level_minus2_and_sign.

The syntax element level_minus2_and sign is derived by invoking the parsing process of subclause 9.2.1 with 0 as input and level_minus2_and sign as output.

#### 9.2.4.3 Parsing process for run_level_one

Inputs to this process are bits from slice data, the scan position n of the previous non-zero transform coefficient in inverse scan order, and the variable TrOne.

Outputs of this process are the syntax element run_level_one and the variable TrOne.

The value of run_level_one is derived as follows.

– The variable N is set equal to (1 << log2TrafoSize) >> (cIdx > 0 ? 2 : 0).

– The variable blockType is derived as

```
blockType = (cIdx == 0 ? (PredMode==MODE_INTRA ? 0 : slice_type + 1) + (N > 8 ? 5 : 2) : cIdx – 1)
```
(9-33)

– The variable tableIdx is derived as

```
tableIdx = (blockType = = 2 | |  blockType = = 5) ? (N <= 8 ? 0 : 1) : (N <= 8 ? 2 : 3)
```
(9-34)

– The variable maxRun is set equal to n.

– The variable maxRunIdx is set equal to Min(28, maxRun).

– The variable vlcNum is derived from tableIdx and maxRunIdx as shown in Table 9-9.

– The parsing process described in subclause 9.2.1 is invoked with vlcNum as input and the variable codeNum as output.If blockType is equal to 2 or blockType is equal to 5, the value of run_level_one is derived as follows.

  – If N is equal to 4, the variable largeOnePos is derived from trOne and maxRunIdx as shown in Table 9-10.

  – Otherwise (N is greater than 4), the variable largeOnePos is derived from trOne and maxRunIdx as shown in Table 9-11.

  – The variables levelGreaterThanOneFlag and runOfZeros are derived as follows.

```
if ( largeOnePos > 0 ) {
    if( codeNum < min(largeOnePos, maxRunIdx + 2 ) ) {
        levelGreaterThanOneFlag = 0
        runOfZeros = codeNum
    } else if( codeNum < (maxRunIdx << 1 ) + 4 – largeOnePos ) {
        if(( codeNum + largeOnePos ) & 1 ) {
            levelGreaterThanOneFlag = 0
            runOfZeros = (codeNum + largeOnePos - 1) >> 1
        } else {
            levelGreaterThanOneFlag = 1
            runOfZeros = (codeNum - largeOnePos)>>1                    (9-35)
        }
    } else {
        levelGreaterThanOneFlag = 1
        runOfZeros = codeNum - maxRunIdx – 2
    }
} else {
    if( codeNum & 1) {
        levelGreaterThanOneFlag = 0
        runOfZeros = (codeNum - 1)>>1
    } else {
        runOfZeros = codeNum >> 1
        levelGreaterThanOneFlag = ((codeNum >> 1) <= maxRunIdx)?1:0
    }
}
```

  – The syntax element run_level_one is derived as

```
run_level_one = levelGreaterThanOneFlag ? runOfZeros + maxRun : runOfZeros    (9-36)
```

– Otherwise (blockType is not equal to 2 and blockType is not equal to 5), the value of run_level_one is derived as follows.

  – The variable levelGreaterThanOneFlag is derived as

```
levelGreaterThanOneFlag  = codeNum <= maxRun + 1 ? 0 : 1                   (9-37)
```

  – If maxRun is less than 28, the variable runOfZeros is derived from maxRun and codeNum as shown in Table 9-12.

  – Otherwise (maxRun is not less than 28), the variable runOfZeros is derived as

```
runOfZeros = codeNum <= maxRun + 1 ? codeNum : codeNum – maxRun – 2        (9-38)
```

  – The syntax element run_level_one is derived as

```
run_level_one = levelGreaterThanOneFlag ? runOfZeros + n : runOfZeros      (9-39)
```

– The variable trOne is derived as

```
trOne = (trOne = = 0 | | levelGreaterThanOneFlag) ? 0 : Max(4, trOne + 1)  (9-40)
```

**Table 9-9 – Derivation of vlcNum from tableIdx and maxRunIdx**

| | maxRunIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **tableIdx** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 0 | 8 | 0 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 |
| 1 | 8 | 0 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 |

Deleted: 9-9

Deleted: 9-9

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 0 | 0 | 0 | 0 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 3 | 8 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6 | 6 |

| | maxRunIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **tableIdx** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| 0 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 |
| 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 3 | 6 |
| 2 | 5 | 5 | 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 5 |
| 3 | 6 | 6 | 6 | 6 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 6 |

**Table 9-10 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is equal to 4**

Deleted: 9-10

Deleted: 9-10

| | maxRunIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **trOne** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 2 | 3 | 4 | 5 | 6 | 5 | 6 | 7 | 7 | 7 | 7 | 7 | 6 | 4 | 2 |
| 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 3 | 3 | 3 | 2 | NA |
| 3 | 2 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | NA | NA |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | NA | NA | NA |

**Table 9-11 – Derivation of largeOnePos from tableIdx and maxRunIdx when N is greater than 4**

Deleted: 9-11

Deleted: 9-11

| | maxRunIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **trOne** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 4 | 5 | 5 | 6 | 6 | 6 | 6 | 6 |
| 1 | 2 | 4 | 4 | 6 | 6 | 8 | 8 | 10 | 11 | 13 | 15 | 13 | 14 | 15 | 16 |
| 2 | 2 | 3 | 4 | 5 | 4 | 5 | 6 | 6 | 7 | 8 | 8 | 9 | 9 | 10 | 10 |
| 3 | 2 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 6 | 6 | 7 | 7 |
| 4 | 2 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |

| | maxRunIdx | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **trOne** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** |
| 0 | 6 | 7 | 8 | 7 | 8 | 8 | 9 | 10 | 10 | 10 | 12 | 10 | 9 | 8 |
| 1 | 18 | 18 | 21 | 20 | 21 | 22 | 23 | 25 | 25 | 26 | 27 | 28 | 29 | 27 |
| 2 | 11 | 11 | 12 | 13 | 14 | 15 | 16 | 16 | 17 | 18 | 19 | 20 | 19 | 19 |
| 3 | 7 | 8 | 9 | 9 | 10 | 10 | 11 | 11 | 12 | 12 | 13 | 13 | 13 | 13 |
| 4 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 7 | 7 | 8 |

**Table 9-12 – Derivation of runOfZeros from maxRun and codeNum**

Deleted: 9-12

Deleted: 9-12

| | codeNum%(maxRun+1) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **maxRun** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| 0 | 1 | 0 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 1 | 2 | 1 | 0 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2 | 3 | 0 | 2 | 1 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 1 | 0 | 2 | 3 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 4 | 5 | 0 | 3 | 2 | 4 | 1 | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 5 | 6 | 0 | 1 | 4 | 5 | 3 | 2 | NA | NA | NA | NA | NA | NA | NA | NA |
| 6 | 7 | 1 | 0 | 2 | 4 | 3 | 6 | 5 | NA | NA | NA | NA | NA | NA | NA |
| 7 | 8 | 0 | 3 | 4 | 2 | 1 | 5 | 7 | 6 | NA | NA | NA | NA | NA | NA |
| 8 | 9 | 0 | 5 | 1 | 6 | 4 | 8 | 3 | 2 | 7 | NA | NA | NA | NA | NA |
| 9 | 10 | 0 | 1 | 6 | 7 | 2 | 9 | 5 | 4 | 3 | 8 | NA | NA | NA | NA |
| 10 | 11 | 1 | 0 | 2 | 3 | 6 | 7 | 4 | 5 | 8 | 10 | 9 | NA | NA | NA |
| 11 | 12 | 0 | 3 | 2 | 1 | 4 | 6 | 5 | 7 | 11 | 8 | 9 | 10 | NA | NA |
| 12 | 13 | 0 | 1 | 5 | 6 | 4 | 2 | 3 | 7 | 8 | 12 | 10 | 11 | 9 | NA |
| 13 | 14 | 0 | 1 | 7 | 2 | 8 | 6 | 5 | 3 | 4 | 13 | 12 | 9 | 11 | 10 |
| 14 | 15 | 0 | 1 | 2 | 8 | 3 | 9 | 7 | 6 | 4 | 5 | 14 | 10 | 13 | 12 |
| 15 | 16 | 0 | 1 | 2 | 3 | 4 | 8 | 7 | 5 | 9 | 6 | 10 | 15 | 11 | 13 |
| 16 | 17 | 0 | 3 | 1 | 2 | 4 | 5 | 8 | 9 | 7 | 6 | 10 | 11 | 12 | 16 |
| 17 | 18 | 0 | 1 | 5 | 2 | 6 | 4 | 3 | 7 | 8 | 9 | 10 | 17 | 13 | 14 |
| 18 | 19 | 0 | 1 | 7 | 8 | 2 | 6 | 3 | 9 | 4 | 5 | 10 | 18 | 15 | 16 |
| 19 | 20 | 0 | 9 | 1 | 10 | 2 | 8 | 3 | 7 | 4 | 16 | 17 | 6 | 19 | 5 |
| 20 | 21 | 0 | 1 | 10 | 2 | 11 | 3 | 9 | 5 | 4 | 17 | 20 | 18 | 8 | 7 |
| 21 | 22 | 1 | 0 | 2 | 3 | 4 | 10 | 11 | 5 | 6 | 9 | 7 | 12 | 8 | 13 |
| 22 | 23 | 0 | 3 | 4 | 2 | 1 | 5 | 7 | 6 | 10 | 11 | 8 | 9 | 12 | 13 |
| 23 | 24 | 0 | 5 | 6 | 1 | 4 | 3 | 7 | 2 | 8 | 10 | 11 | 9 | 12 | 15 |
| 24 | 25 | 0 | 7 | 1 | 8 | 6 | 5 | 2 | 9 | 10 | 4 | 11 | 3 | 12 | 17 |
| 25 | 26 | 0 | 1 | 9 | 10 | 2 | 8 | 11 | 7 | 3 | 6 | 5 | 12 | 4 | 19 |
| 26 | 27 | 0 | 1 | 11 | 2 | 10 | 12 | 3 | 9 | 7 | 8 | 4 | 6 | 20 | 13 |
| 27 | 28 | 0 | 1 | 2 | 12 | 3 | 4 | 11 | 13 | 9 | 5 | 8 | 10 | 7 | 6 |

| | **codeNum%(maxRun+1)** | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **maxRun** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** |
| 0 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 1 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 2 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 3 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 4 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 5 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 6 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 7 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 8 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 9 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 10 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 11 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 12 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 13 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 14 | 11 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 15 | 14 | 12 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 16 | 13 | 14 | 15 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 17 | 12 | 15 | 11 | 16 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| 18 | 14 | 11 | 13 | 17 | 12 | NA | NA | NA | NA | NA | NA | NA | NA | NA |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19 | 11 | 15 | 12 | 14 | 13 | 18 | NA | NA | NA | NA | NA | NA | NA | NA |
| 20 | 6 | 12 | 16 | 13 | 15 | 19 | 14 | NA | NA | NA | NA | NA | NA | NA |
| 21 | 18 | 21 | 17 | 19 | 14 | 20 | 16 | 15 | NA | NA | NA | NA | NA | NA |
| 22 | 14 | 22 | 18 | 15 | 19 | 20 | 17 | 16 | 21 | NA | NA | NA | NA | NA |
| 23 | 14 | 13 | 23 | 16 | 19 | 17 | 18 | 21 | 20 | 22 | NA | NA | NA | NA |
| 24 | 16 | 13 | 24 | 18 | 19 | 15 | 20 | 23 | 14 | 21 | 22 | NA | NA | NA |
| 25 | 13 | 18 | 20 | 14 | 17 | 25 | 21 | 16 | 23 | 24 | 15 | 22 | NA | NA |
| 26 | 5 | 14 | 26 | 21 | 19 | 18 | 25 | 15 | 16 | 17 | 24 | 22 | 23 | NA |
| 27 | 21 | 14 | 27 | 15 | 22 | 19 | 16 | 20 | 17 | 26 | 18 | 25 | 23 | 24 |

#### 9.2.4.4 Parsing process for level

Inputs to this process are bits from slice data and the variable vlcNumLevel.

Outputs of this process are the syntax element level and the variable vlcNumLevel.

– The syntax element level is determined by invoking the parsing process described in subclause 9.2.1 with vlcNumLevel as input and level as output.

– The variable vlcNumLevel derived in the following ordered steps updated as follows.

1. The variable vlcIdx is set equal to vlcIdx = Min(3,vlcNumLevel)
2. The variable levelThreshold is derived from vlcIdx as shown in Table 9-13.
3. The variable vlcNumLevel is derived as

$$vlcNumLevel = vlcNumLevel + (level > levelThreshold \;\&\&\; vlcNumLevel < 4 \;?\; 1 : 0) \quad (9\text{-}41)$$

**Table 9-13 – Derivation of levelThreshold from vlcIdx**

| vlcIdx | levelThreshold |
|---|---|
| 0 | 4 |
| 1 | 6 |
| 2 | 14 |
| 3 | 28 |

Deleted: 9-13

Deleted: 9-13

#### 9.2.4.5 Parsing process for cu_split_pred_part_mode

This process is invoked when entropy_coding_mode_flag is equal to 0 for parsing syntax element **cu_split_pred_part_mode** in subclause 7.3.5.

Inputs to this process are bits from slice data, splitPredPartModeTable.

Output of this process is the syntax element cu_split_pred_part_mode.

The value of cu_split_pred_part_mode is derived as follows.

– If log2CUSize is greater than Log2MinCUSize, the parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 6 as inputs and the variable codeNum as output.

– Otherwise, the parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 5 as inputs and the variable codeNum as output.

– The syntax element cu_split_pred_part_mode is derived as

$$cu\_split\_pred\_part\_mode = splitPredPartModeTable[Log2MaxCUSize - log2CUSize][codeNum] \quad (9\text{-}42)$$

Deleted: 41

– If log2CUSize is equal to Log2MinCUSize and cu_split_part_mode has a value of 5, PredMode and PartMode are determined as follows

1. Read one bit and assign its value to a variable b
2. If b is equal to 1, set PredMode to MODE_INTRA and PartMode to PART_2Nx2N
3. Otherwise, read the next one bit and assign its value to the variable b.

- if b is equal to 1, set PredMode to MODE_INTRA and PartMode to PART_NxN
- Otherwise, set PredMode to MODE_INTER and PartMode to PART_NxN

– The variable array splitPredPartModeTable[Log2MaxCUSize – log2CUSize] is updated by invoking process 9.2.3 with splitPredPartModeTable[Log2MaxCUSize – log2CUSize], codeNum and a value of 4 as inputs.

#### 9.2.4.6 Parsing process for rem_intra_luma_pred_mode

This process is invoked when entropy_coding_mode_flag is equal to 0 for parsing syntax element rem_intra_luma_pred_mode in subclause 7.3.7.

Inputs to this process are bits from slice data, a variable puSize specifying the size of the current prediction unit, NumMPMCand and a variable array intraModeTable.

Outputs of this process are the syntax element rem_intra_luma_pred_mode and value-updated intraModeTable.

The value of rem_intra_luma_pred_mode is derived as follows.

– Based on the input puSize, the value of a variable intraPredModeNum indicating the number of intra prediction modes for the given size of prediction unit is obtained according to Table 7-12.

Deleted: Table 7-11

– If intraPredModeNum is equal to 3, read one bit and assign its value to rem_intra_luma_pred_mode. No further step is carried out. Othewise,

– The values of variable k and vlcNum are obtained as follows
  - If intraPredModeNum is equal to 17
    - If NumMPMCand is equal to 1, set vlcNum equal to 16 and k equal to 0.
    - Otherwise, set vlcNum equal to 17 and k equal to 1.
  - If intraPredModeNum is equal to 34
    - If NumMPMCand is equal to 1, set vlcNum equal to 18 and k equal to 2.
    - Otherwise, set vlcNum equal to 19 and k equal to 3.
  - The parsing process described in subclause 9.2.1 is invoked with vlcNum as input and the variable codeNum as output.
  - The value of rem_intra_luma_pred_mode is set equal to intraModeTable[k][codeNum].
  - The value of a variable counterNum is set to 0. The variable array intraModeTable[k] is updated by invoking process in subclause 9.2.3 with intraModeTable[k], codeNum and counterNum as inputs.

#### 9.2.4.7 Parsing process for cbp_and_split_transform

This process is invoked when entropy_coding_mode_flag is equal to 0 for parsing syntax element cbp_and_split_transform in subclause 7.3.8.

Inputs to this process are bits from slice data, a variable cbpVlcNumIdx, trafoDepth and a variable array cbpSplitTransTable.

Outputs of this process are the syntax element cbp_and_split_transform and value-updated cbpVlcNumIndex and cbpSplitTransTable.

The value of cbp_and_split_transform is derived as follows.

– Obtain a 4-bit value flagPattern accoding to subclause 7.4.8.

– Derive the value of a variable k according to Table 9-16.

– Set the value of a variable n as

$$n = (PredMode == MODE\_INTRA\ ?\ 0 : 1) \tag{9-43}$$

– Obtain a value of vlcNum as follows
  - If flagPattern is equal to 8,

$$vlcNum = (n == 0\ ?\ cbpVlcNumTable[cbpVlcNumIdx]:11) \tag{9-44}$$

  - Otherwise, if flagPattern is equal to 11, 13 or 15 and n is equal to 0, set vlcNum to 15;
  - Otherwise, set vlcNum to 14.

– If vlcNum is euqal to 14, obtain the value of a variable cMax according to Table 9-15.

– The parsing process described in subclause 9.2.1 is invoked with vlcNum and cMax as inputs and the variable codeNum as output.

– Set the value of cbp_and_split_transform equal to cbpSplitTransTable[k][n][codeNum]

– If flagPattern is not equal to 10 or 12, a variable counterNum is obtained according to Table 9-16 and the variable array cbpSplitTransTable[k][n] is updated by invoking process 9.2.3 with cbpSplitTransTable[k][n], codeNum and counterNum as inputs.

– If flagPattern is euqal to 8 and n is equal to 0, the value of cbpVlcNumIdx is updated as follows

```
if(codeNum < cbpVlcNumIdx)
    cbpVlcNumIdx -= 1                                   (9-45)
else if (codeNum > cbpVlcNumIdx)
    cbpVlcNumIdx += 1
```

– If flagPattern is euqal to 15, the exact values of Cb and Cr in Table 7-14 are determined as follows.

  – The parsing process described in subclause 9.2.1 is invoked with vlcNum of 14 and cMax of 2 as inputs and the variable codeNum as output.

  – Set the value of variable b as

    b = n? (codeNum + 1) : (3 – codeNum)  (9-46)

  – The values of Cb and Cr are set equal to

    Cb = (b >> 1)  (9-47)

    Cr = (b & 0x01)  (9-48)

**Table 9-14 – Specification of cbpVlcNumTable[cbpVlcNumIdx]**

| **cbpVlcNumIdx** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **vlcNum** | 1 | 2 | 2 | 2 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 |

**Table 9-15 – Derivation of cMax from flagPattern and n**

| | **flagPattern** | | | |
|---|---|---|---|---|
| **n** | 9 | 10, 12 | 11,13,15 | 14 |
| 0 | 3 | 3 | - | 7 |
| 1 | 2 | 3 | 6 | 7 |

**Table 9-16 – Specification of cbpSplitTransTable and counterNum associated**

| **flagPattern** | 8 | 9 | 10, 12 | 11, 13, 15 | 14 (trafoDepth=0) | 14 (trafoDepth>0) |
|---|---|---|---|---|---|---|
| **k** | 0 | 1 | 2 | 3 | 4 | 5 |
| **counterNum** | 2 | 3 | 0 | 4 | 4 | 4 |

## 9.3 CABAC parsing process for slice data

[Ed. (TW) needs to be extended]

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.



Deleted: Table 7-13

Deleted: 9-14

Deleted: 9-14

Deleted: 9-15

Deleted: 9-15

Deleted: 9-16

Deleted: 9-16

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 9.3.2.10.

Deleted: 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.

### 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.

#### 9.3.1.1 Initialisation process for context variables

Outputs of this process are the initialised CABAC context variables indexed by ctxIdx.

Table 9-18 toTable 9-43 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 to 7.3.10 except for the end-of-slice flag and those related to CAVLC.

Deleted: Tables 9-5

Deleted: 9-26

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 $SliceQP_Y = 26 + pic\_init\_qp\_minus26 + slice\_qp\_delta$ (7-12). Given the two table entries ( m, n ), the initialisation is specified by the following pseudo-code process:

Deleted: 7-9

```
preCtxState = Clip3( 1, 126, ( ( m * Clip3( 0, 51, SliceQPY ) ) >> 4 ) + n )
if( preCtxState <= 63 ) {
    pStateIdx = 63 − preCtxState
    valMPS = 0                                                    (9-49)
} else {
    pStateIdx = preCtxState − 64
    valMPS = 1
}
```

Deleted: 4

In Table 9-17, the ctxIdxTable and the ctxIdx for which initialisation is needed for each of the slice types are listed. The referenced context index tables ctxIdxTable include the values of m and n needed for the initialisation.

Deleted: Table 9-4

**Table 9-17 – Association of ctxIdx and syntax elements for each slice type in the initialisation process**

| | Syntax element | ctxIdxTable | Slice Type | | |
|---|---|---|---|---|---|
| | | | I | P | B |
| slice_header() | alf_cu_flag | Table 9-18 | 0 2 | 3..5 | 6..8 |
| coding_tree() | split_coding_unit_flag | Table 9-19 | 0...2 | 3..5 | 6..8 |
| coding_unit() | skip_flag | Table 9-20 | | 0..2 | 3..5 |
| | cu_qp_delta | Table 9-21 | 0..3 | 4..7 | 8..11 |
| | pred_type | Table 9-22 | 0 | 1..4 | 5..9 |
| prediction_unit() | prev_intra_luma_pred_flag | Table 9-23 | 0 | 1 | 2 |
| | rem_intra_luma_pred_mode | Table 9-24 | 0 | 1 | 2 |
| | intra_chroma_pred_mode | Table 9-25 | 0..3 | 4..7 | 8..11 |
| | merge_flag | Table 9-26 | | 0..2 | 3..5 |
| | merge_idx | Table 9-27 | | 0..3 | 4..7 |
| | inter_pred_flag | Table 9-28 | | | 0..2 |
| | ref_idx_lc, ref_idx_l0, ref_idx_l1 | Table 9-29 | | 0..5 | 6..11 |
| | mvd_l0[ ][ ][ 0 ] | Table 9-30 | | 0..6 | 14..20 |
| | mvd_lc[ ][ ][ 0 ], mvd_l1[ ][ ][ 0 ] | Table 9-30 | | | 14..20 |
| | mvd_l0[ ][ ][ 1 ] | Table 9-30 | | 7..13 | 21..27 |
| | mvd_lc[ ][ ][ 1 ], mvd_l1[ ][ ][ 1 ] | Table 9-30 | | | 21..27 |
| | mvp_idx_lc, mvp_idx_l0, mvp_idx_l1 | Table 9-31 | | 0..1 | 2..3 |
| transform_tree() | no_residual_data_flag | Table 9-32 | | 0..3 | 4..7 |
| | split_transform_flag | Table 9-33 | 0..3 | 4..7 | 8..11 |
| | cbf_luma | Table 9-34 | 0..3 | 4..7 | 8..11 |
| | cbf_cb | Table 9-35 | 0..3 | 4..7 | 8..11 |
| | cbf_cr | Table 9-36 | 0..3 | 4..7 | 8..11 |
| residual_coding( ) | last_significant_coeff_x | Table 9-37 | 0..40 | 41..81 | 82..122 |
| | last_significant_coeff_y | Table 9-38 | 0..40 | 41..81 | 82..122 |

Deleted: 9-17
Deleted: 9-17
Formatted: Font: 8 pt
Deleted: Table 9-5
Formatted: Font: 8 pt
Deleted: Table 9-6
Formatted: Font: 8 pt
Deleted: Table 9-7
Formatted: Font: 8 pt
Deleted: Table 9-8
Formatted: Font: 8 pt
Deleted: Table 9-9
Formatted: Font: 8 pt
Deleted: Table 9-10
Formatted: Font: 8 pt
Deleted: Table 9-11
Formatted: Font: 8 pt
Deleted: Table 9-12
Formatted: Font: 8 pt
Deleted: Table 9-13
Formatted: Font: 8 pt
Deleted: Table 9-14
Formatted: Font: 8 pt
Deleted: Table 9-15
Formatted: Font: 8 pt
Deleted: Table 9-16
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-18
Formatted: Font: 8 pt
Deleted: Table 9-19
Formatted: Font: 8 pt
Deleted: Table 9-20
Formatted: Font: 8 pt
Deleted: Table 9-21
Formatted: Font: 8 pt
Deleted: Table 9-22
Formatted: Font: 8 pt
Deleted: Table 9-23
Formatted: Font: 8 pt
Deleted: Table 9-24
Formatted: Font: 8 pt
Deleted: Table 9-25

| | | | | | |
|---|---|---|---|---|---|
| | significant_coeff_flag | needs update | Table 9-39 | Table 9-40 | Table 9-41 |
| | coeff_abs_level_greater1_flag | Table 9-42 | 0..79 | 80..159 | 160..239 |
| | coeff_abs_level_greater2_flag | Table 9-43 | 0..79 | 80..159 | 160..239 |

| Initialisation | significant_coeff_flag ctxIdx | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **1** | **1** | **1** | **1** | **1** | **1** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | **1** | **1** | **1** | **1** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **3** | **3** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **4** | **4** | **4** | **4** | **4** | **4** | **4** | **4** |
| **m** | - | - | - | - | 5 | - | - | - | - | - | 0 | - | 2 | 0 | - | 0 |
| **n** | 1 | 9 | 9 | 1 | 3 | 6 | 1 | 7 | 6 | 1 | 2 | 4 | 5 | 6 | 9 | 6 |
| | **4** | **4** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **5** | **6** | **6** | **6** | **6** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | **6** | **6** | **6** | **6** | **6** | **6** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** |
| **m** | - | 0 | - | - | 1 | 2 | - | 1 | 2 | - | 6 | 5 | 2 | 0 | - | 0 |
| **n** | 8 | 6 | 7 | 7 | 7 | 5 | 8 | 3 | 5 | 8 | 1 | 4 | 5 | 6 | 8 | 6 |
| | **8** | **8** | **8** | **8** | **8** | **8** | **8** | **8** | **8** | **8** | **9** | **9** | **9** | **9** | **9** | **9** |
| **m** | - | - | - | - | - | - | - | - | 4 | - | - | 2 | 5 | 6 | - | 0 |
| **n** | 9 | 7 | 7 | 7 | 7 | 7 | 8 | 9 | 4 | 8 | 7 | 4 | 5 | 5 | 1 | 6 |
| | **9** | **9** | **9** | **9** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |
| **m** | - | - | - | - | 5 | 3 | - | - | 0 | - | - | 8 | 4 | - | - | 0 |
| **n** | 9 | 7 | 7 | 7 | 3 | 5 | 9 | 5 | 5 | 8 | 3 | 3 | 5 | 6 | 8 | 6 |

Table 9-40

NOTE 2 – ctxIdxTable equal to 0 and ctxIdx equal to 0 are associated with the end_of_slice_flag. The decoding process specified in subclause 9.3.3.2.4 applies to ctxIdxTable equal to 0 and ctxIdx equal to 0. 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 ctxIdxTable equal to 0 and ctxIdx equal to 0 are specified to be pStateIdx = 63 and valMPS = 0, where pStateIdx = 63 represents a non-adapting probability state.

**Table 9-18 – Values of variables m and n for alf_cu_flag ctxIdx**

| Initialisation variables | alf_cu_flag ctxIdx | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

**Table 9-19 – Values of variables m and n for split_coding_unit_flag ctxIdx**

| Initialisation variables | split_coding_unit_flag ctxIdx | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **m** | -7 | -10 | -10 | -14 | -6 | -6 | −14 | −7 | −10 |
| **n** | 68 | 87 | 105 | 71 | 73 | 91 | 71 | 74 | 92 |

**Formatted:** Font: 8 pt, Highlight

**Formatted:** Font: Times New Roman, 8 pt, English (UK), Highlight

**Formatted:** Font: 8 pt, Highlight

**Formatted Table**

**Deleted:** Table 9-26

**Deleted:** Page Break Table 9-28

**Deleted:** ¶ Page Break Table 9-27

**Formatted:** Font: 8 pt

**Formatted:** Keep with next

**Deleted:** Page Break Table 9-29

**Formatted:** Font: 8 pt

**Deleted:** Table 9-30

**Deleted:** 9-18

**Deleted:** 9-18

**Deleted:** 9-19

**Deleted:** 9-19

Table 9-20 – Values of variables m and n for skip_flag ctxIdx

| Initialisation variables | skip_flag ctxIdx | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| m | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 64 | 64 | 64 | 64 | 64 | 64 |

Table 9-21 – Values of variables m and n for cu_qp_delta ctxIdx

| Initialisation variables | cu_qp_delta flag ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

Table 9-22 – Values of variables m and n for pred_type

| Initialisation variables | pred_type ctxIdx | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| m | 0 | -25 | -1 | -3 | 6 | 6 | -1 | 13 | -11 | -11 |
| n | 73 | 89 | 64 | 63 | 78 | 50 | 56 | 53 | 76 | 70 |

Table 9-23 – Values of variable m and n for prev_intra_luma_pred_flag ctxIdx

| Initialisation variables | prev_intra_luma_pred_flag ctxIdx | | |
|---|---|---|---|
| | 0 | 1 | 2 |
| m | 2 | 0 | 0 |
| n | 54 | 50 | 51 |

Table 9-24 – Values of variable m and n for rem_intra_luma_pred_mode ctxIdx

| Initialisation variables | rem_intra_luma_pred_mode ctxIdx | | |
|---|---|---|---|
| | 0 | 1 | 2 |
| m | -3 | -2 | 1 |
| n | 65 | 61 | 55 |

Table 9-25 – Values of variable m and n for intra_chroma_pred_mode ctxIdx

| Initialisation variables | intra_chroma_pred_mode ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| m | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

Deleted: 9-20
Deleted: 9-20
Deleted: 9-21
Deleted: 9-21
Deleted: 9-22
Deleted: 9-22
Deleted: 9-23
Deleted: 9-23
Deleted: 9-24
Deleted: 9-24
Deleted: 9-25
Deleted: 9-25

**Table 9-26 – Values of variable m and n for merge_flag ctxIdx**

| Initialisation variables | merge_flag ctxIdx | | | | | |
|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 |

**Table 9-27 – Values of variable m and n for merge_idx ctxIdx**

| Initialisation variables | merge_idx ctxIdx | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **m** | 0 | 0 | 0 | 0 | 1 | 6 | -7 | −4 |
| **n** | 64 | 64 | 64 | 64 | 65 | 42 | 75 | 72 |

**Table 9-28 – Values of variable m and n for inter_pred_flag ctxIdx**

| Initialisation variables | inter_pred_flag ctxIdx | | |
|---|---|---|---|
| | **0** | **1** | **2** |
| **m** | -2 | -5 | -9 |
| **n** | 58 | 70 | 85 |

**Table 9-29 – Values of variable m and n for ref_idx_lc, ref_idx_l0, ref_idx_l1 ctxIdx**

| Initialisation variables | ref_idx_lc, ref_idx_l0, ref_idx_l1 ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **m** | -6 | -10 | -8 | -17 | 1 | 0 | -9 | -9 | -9 | -12 | -18 | 0 |
| **n** | 59 | 75 | 75 | 96 | 59 | 64 | 55 | 71 | 76 | 86 | 55 | 64 |

**Table 9-30 – Values of variable m and n for mvd_lc, mvd_l0, mvd_l1 ctxIdx**

| Initialisation variables | mvd_lc, mvd_l0, mvd_l1 ctxIdx | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |
| **m** | 5 | -2 | -7 | 14 | 16 | 6 | 9 | 2 | -3 | -10 | 14 | 17 | 6 | 8 |
| **n** | 61 | 79 | 89 | 39 | 49 | 60 | 71 | 63 | 78 | 94 | 37 | 46 | 60 | 73 |
| | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** |
| **m** | 1 | -4 | -10 | 11 | 14 | 4 | 7 | 1 | -4 | -10 | 11 | 15 | 5 | 8 |
| **n** | 67 | 83 | 97 | 49 | 56 | 69 | 77 | 65 | 81 | 96 | 48 | 52 | 67 | 76 |

**Table 9-31 – Values of variable m and n for mvp_idx_lc, mvp_idx_l0, mvp_idx_l1 ctxIdx**

| Initialisation variables | mvp_idx_lc, mvp_idx_l0, mvp_idx_l1 ctxIdx | | | |
|---|---|---|---|---|
| | **0** | 1 | 2 | **3** |
| **m** | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 |

Deleted: 9-26
Deleted: 9-26
Deleted: 9-27
Deleted: 9-27
Deleted: 9-28
Deleted: 9-28
Deleted: 9-29
Deleted: 9-29
Deleted: 9-30
Deleted: 9-30
Deleted: 9-31
Deleted: 9-31

**Table 9-32 – Values of variable m and n for no_residual_data_flag ctxIdx**

| Initialisation variables | no_residual_data_flag ctxIdx | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **m** | -14 | -12 | -8 | -25 | -26 | -16 | -14 | -27 |
| **n** | 72 | 86 | 78 | 128 | 88 | 91 | 87 | 132 |

**Table 9-33 – Values of variable m and n for split_transform_flag ctxIdx**

| Initialisation variables | split_transform_flag ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **m** | 0 | 12 | 22 | -2 | 0 | -28 | -30 | -34 | -11 | -31 | -42 | -47 |
| **n** | 64 | 12 | 4 | 49 | 13 | 89 | 99 | 106 | 38 | 88 | 118 | 130 |

**Table 9-34 – Values of variable m and n for cbf_luma ctxIdx**

| Initialisation variables | cbf_luma ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **m** | -22 | -5 | -16 | -16 | -18 | -41 | -29 | -23 | -11 | -32 | -19 | -16 |
| **n** | 116 | 75 | 112 | 111 | 98 | 120 | 117 | 108 | 80 | 83 | 89 | 85 |

**Table 9-35 – Values of variable m and n for cbf_cb ctxIdx**

| Initialisation variables | cbf_cb ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **m** | -35 | -12 | -9 | -10 | -46 | -42 | -11 | -19 | -22 | -48 | -7 | -37 |
| **n** | 116 | 61 | 73 | 75 | 114 | 119 | 74 | 90 | 52 | 123 | 68 | 121 |

**Table 9-36 – Values of variable m and n for cbf_cr ctxIdx**

| Initialisation variables | cbf_cr ctxIdx | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **m** | -29 | -12 | -5 | -6 | -43 | -41 | -17 | -25 | -19 | -48 | -21 | -9 |
| **n** | 104 | 59 | 65 | 67 | 107 | 118 | 86 | 101 | 45 | 123 | 94 | 73 |

Deleted: 9-32
Deleted: 9-32
Deleted: 9-33
Deleted: 9-33
Deleted: 9-34
Deleted: 9-34
Deleted: 9-35
Deleted: 9-35
Deleted: 9-36
Deleted: 9-36

**Table 9-37 – Values of variable m and n for last_significant_coefficient_x ctxIdx**

Deleted: 9-37

Deleted: 9-37

| Initialisation variables | last_significant_coeff_y ctxIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| **m** | 19 | 12 | 16 | 22 | 12 | 12 | 12 | 5 | 16 | 15 | 17 | 17 | 19 | 19 | 4 |
| **n** | 19 | 36 | 34 | 18 | 35 | 35 | 32 | 46 | 21 | 20 | 13 | 14 | 10 | 12 | 37 |
| | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| **m** | 13 | 22 | 27 | 26 | 18 | 6 | 12 | 34 | 38 | 24 | 14 | 41 | 45 | 56 | 30 |
| **n** | 22 | -4 | -19 | -12 | 6 | 27 | 10 | -33 | -42 | -15 | 7 | 7 | 1 | -9 | 22 |
| | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** |
| **m** | 14 | 23 | 19 | 25 | 29 | 29 | 19 | 10 | 12 | -2 | -37 | 15 | 14 | 15 | 25 |
| **n** | 40 | 24 | 32 | 26 | 10 | 4 | 20 | 37 | 38 | 60 | 114 | 29 | 36 | 41 | 8 |
| | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** |
| **m** | 26 | 25 | 21 | 12 | 18 | 25 | 32 | 32 | 29 | 24 | 13 | 27 | 40 | 42 | 40 |
| **n** | 4 | 10 | 16 | 34 | 21 | 1 | -12 | -15 | -11 | -5 | 12 | 6 | -32 | -39 | -39 |
| | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** |
| **m** | 45 | 43 | 45 | 38 | 15 | 11 | 11 | 26 | 28 | 21 | 15 | 27 | 29 | 23 | -1 |
| **n** | -51 | -51 | -56 | -46 | -6 | -4 | -4 | 29 | 29 | 46 | 45 | 17 | 12 | 15 | 57 |
| | **75** | **76** | **77** | **78** | **79** | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** |
| **m** | 6 | 22 | 31 | 37 | 43 | 43 | 48 | 15 | 14 | 15 | 25 | 26 | 25 | 21 | 12 |
| **n** | 65 | 17 | -4 | -25 | -44 | -46 | -44 | 29 | 36 | 41 | 8 | 4 | 10 | 16 | 34 |
| | **90** | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** |
| **m** | 18 | 25 | 32 | 32 | 29 | 24 | 13 | 27 | 40 | 42 | 40 | 45 | 43 | 45 | 38 |
| **n** | 21 | 1 | -12 | -15 | -11 | -5 | 12 | 6 | -32 | -39 | -39 | -51 | -51 | -56 | -46 |
| | **105** | **106** | **107** | **108** | **109** | **110** | **111** | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** |
| **m** | 15 | 11 | 11 | 26 | 28 | 21 | 15 | 27 | 29 | 23 | -1 | 6 | 22 | 31 | 37 |
| **n** | -6 | -4 | -4 | 29 | 29 | 46 | 45 | 17 | 12 | 15 | 57 | 65 | 17 | -4 | -25 |
| | **120** | **121** | **122** | | | | | | | | | | | | |
| **m** | 43 | 43 | 48 | | | | | | | | | | | | |
| **n** | -44 | -46 | -44 | | | | | | | | | | | | |

**Table 9-38 – Values of variable m and n for last_significant_coefficient_y ctxIdx**

Deleted: 9-38

Deleted: 9-38

| Initialisation variables | last_significant_coeff_y ctxIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| **m** | 19 | 12 | 16 | 22 | 12 | 12 | 12 | 5 | 16 | 15 | 17 | 17 | 19 | 19 | 4 |
| **n** | 19 | 36 | 34 | 18 | 35 | 35 | 32 | 46 | 21 | 20 | 13 | 14 | 10 | 12 | 37 |
| | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| **m** | 13 | 22 | 27 | 26 | 18 | 6 | 12 | 34 | 38 | 24 | 14 | 41 | 45 | 56 | 30 |
| **n** | 22 | -4 | -19 | -12 | 6 | 27 | 10 | -33 | -42 | -15 | 7 | 7 | 1 | -9 | 22 |
| | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** |
| **m** | 14 | 23 | 19 | 25 | 29 | 29 | 19 | 10 | 12 | -2 | -37 | 15 | 14 | 15 | 25 |
| **n** | 40 | 24 | 32 | 26 | 10 | 4 | 20 | 37 | 38 | 60 | 114 | 29 | 36 | 41 | 8 |
| | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** |
| **m** | 26 | 25 | 21 | 12 | 18 | 25 | 32 | 32 | 29 | 24 | 13 | 27 | 40 | 42 | 40 |
| **n** | 4 | 10 | 16 | 34 | 21 | 1 | -12 | -15 | -11 | -5 | 12 | 6 | -32 | -39 | -39 |
| | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** |
| **m** | 45 | 43 | 45 | 38 | 15 | 11 | 11 | 26 | 28 | 21 | 15 | 27 | 29 | 23 | -1 |
| **n** | -51 | -51 | -56 | -46 | -6 | -4 | -4 | 29 | 29 | 46 | 45 | 17 | 12 | 15 | 57 |
| | **75** | **76** | **77** | **78** | **79** | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** |
| **m** | 6 | 22 | 31 | 37 | 43 | 43 | 48 | 15 | 14 | 15 | 25 | 26 | 25 | 21 | 12 |
| **n** | 65 | 17 | -4 | -25 | -44 | -46 | -44 | 29 | 36 | 41 | 8 | 4 | 10 | 16 | 34 |
| | **90** | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** |
| **m** | 18 | 25 | 32 | 32 | 29 | 24 | 13 | 27 | 40 | 42 | 40 | 45 | 43 | 45 | 38 |
| **n** | 21 | 1 | -12 | -15 | -11 | -5 | 12 | 6 | -32 | -39 | -39 | -51 | -51 | -56 | -46 |
| | **105** | **106** | **107** | **108** | **109** | **110** | **111** | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** |
| **m** | 15 | 11 | 11 | 26 | 28 | 21 | 15 | 27 | 29 | 23 | -1 | 6 | 22 | 31 | 37 |
| **n** | -6 | -4 | -4 | 29 | 29 | 46 | 45 | 17 | 12 | 15 | 57 | 65 | 17 | -4 | -25 |
| | **120** | **121** | **122** | | | | | | | | | | | | |
| **m** | 43 | 43 | 48 | | | | | | | | | | | | |
| **n** | -44 | -46 | -44 | | | | | | | | | | | | |

**Table 9-39 – Values of variable m and n for significant_coeff_flag ctxIdx (A)**

Deleted: 9-39

Deleted: 9-39

| Initialisation variables | significant_coeff_flag ctxIdx | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **m** | -22 | -14 | -17 | -26 | 5 | -2 | -32 | -19 | -7 | -24 | 0 | -1 | 2 | 0 | -15 | 0 |
| **n** | 128 | 96 | 99 | 113 | 31 | 65 | 125 | 72 | 69 | 111 | 25 | 49 | 54 | 64 | 97 | 64 |
| | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **m** | -4 | 0 | -4 | -5 | 17 | 2 | -8 | 1 | 2 | -11 | 6 | 5 | 2 | 0 | -8 | 0 |
| **n** | 88 | 68 | 75 | 74 | 7 | 56 | 80 | 36 | 52 | 85 | 12 | 41 | 54 | 65 | 82 | 64 |
| | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **m** | -8 | -7 | -7 | -9 | -18 | -7 | -11 | -29 | 4 | -14 | -26 | 2 | 5 | 6 | -22 | 0 |
| **n** | 95 | 77 | 73 | 74 | 77 | 72 | 81 | 92 | 47 | 83 | 79 | 46 | 50 | 52 | 104 | 64 |
| | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** | **105** | **106** | **107** | **108** | **109** | **110** | **111** |
| **m** | -7 | -5 | -6 | -8 | 5 | 3 | -18 | -10 | 0 | -11 | -3 | 8 | 4 | -1 | -7 | 0 |
| **n** | 93 | 79 | 79 | 79 | 36 | 56 | 99 | 59 | 56 | 84 | 31 | 35 | 51 | 66 | 81 | 64 |

**Table 9-40 – Values of variable m and n for significant_coeff_flag ctxIdx (B)**

Deleted: 9-40

Deleted: 9-40

| Initialisation variables | significant_coeff_flag ctxIdx | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **m** | -22 | -14 | -17 | -26 | 5 | -2 | -32 | -19 | -7 | -24 | 0 | -1 | 2 | 0 | -15 | 0 |
| **n** | 128 | 96 | 99 | 113 | 31 | 65 | 125 | 72 | 69 | 111 | 25 | 49 | 54 | 64 | 97 | 64 |
| | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **m** | -4 | 0 | -4 | -5 | 17 | 2 | -8 | 1 | 2 | -11 | 6 | 5 | 2 | 0 | -8 | 0 |
| **n** | 88 | 68 | 75 | 74 | 7 | 56 | 80 | 36 | 52 | 85 | 12 | 41 | 54 | 65 | 82 | 64 |
| | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **m** | -8 | -7 | -7 | -9 | -18 | -7 | -11 | -29 | 4 | -14 | -26 | 2 | 5 | 6 | -22 | 0 |
| **n** | 95 | 77 | 73 | 74 | 77 | 72 | 81 | 92 | 47 | 83 | 79 | 46 | 50 | 52 | 104 | 64 |
| | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** | **105** | **106** | **107** | **108** | **109** | **110** | **111** |
| **m** | -7 | -5 | -6 | -8 | 5 | 3 | -18 | -10 | 0 | -11 | -3 | 8 | 4 | -1 | -7 | 0 |
| **n** | 93 | 79 | 79 | 79 | 36 | 56 | 99 | 59 | 56 | 84 | 31 | 35 | 51 | 66 | 81 | 64 |

**Table 9-41 – Values of variable m and n for significant_coeff_flag ctxIdx (C)**

| Initialisation variables | significant_coeff_flag ctxIdx | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** |
| **m** | -34 | -20 | -23 | 38 | 3 | -4 | -17 | -9 | -4 | -16 | -1 | 0 | -1 | -12 | -14 | 0 |
| **n** | 146 | 108 | 111 | 41 | 34 | 69 | 103 | 50 | 66 | 97 | 24 | 50 | 61 | 90 | 97 | 64 |
| | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** |
| **m** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **n** | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |
| | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** | **75** | **76** | **77** | **78** | **79** |
| **m** | -12 | -4 | -8 | -7 | 15 | 2 | -10 | 4 | -10 | -9 | -9 | 6 | 3 | -1 | -14 | 0 |
| **n** | 99 | 75 | 81 | 80 | 13 | 57 | 85 | 27 | 75 | 81 | 46 | 41 | 56 | 69 | 96 | 64 |
| | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** |
| **m** | -7 | -22 | -12 | -35 | -5 | -16 | -9 | -26 | -28 | -38 | -3 | -25 | -9 | 6 | -15 | 0 |
| **n** | 85 | 102 | 79 | 121 | 55 | 88 | 78 | 73 | 99 | 123 | 35 | 97 | 75 | 54 | 92 | 64 |
| | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** | **105** | **106** | **107** | **108** | **109** | **110** | **111** |
| **m** | -9 | -14 | -4 | -14 | 7 | 3 | -7 | -8 | -7 | -8 | -2 | 4 | 1 | -5 | -10 | 0 |
| **n** | 94 | 94 | 74 | 92 | 37 | 57 | 78 | 54 | 70 | 78 | 31 | 49 | 60 | 75 | 87 | 64 |

Deleted: 9-41

Deleted: 9-41

**Table 9-42 – Values of variable m and n for coeff_abs_level_greater1_flag ctxIdx**

Deleted: 9-42

Deleted: 9-42

| Initialisation variables | coeff_abs_level_greater1_flag ctxIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| **m** | -11 | -20 | -16 | -13 | -10 | -5 | -8 | -8 | -3 | -9 | 0 | -5 | -12 | -9 | -1 |
| **n** | 87 | 64 | 68 | 71 | 73 | 67 | 26 | 37 | 36 | 56 | 63 | 39 | 56 | 57 | 52 |
| | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| **m** | -4 | -19 | -28 | -23 | -3 | -2 | -27 | -22 | -14 | -1 | -10 | -22 | -13 | -6 | -5 |
| **n** | 72 | 73 | 88 | 85 | 59 | 72 | 97 | 89 | 77 | 58 | 86 | 74 | 63 | 57 | 63 |
| | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** |
| **m** | -4 | 16 | 9 | 5 | -7 | -6 | -13 | -12 | -18 | -14 | -5 | -8 | -22 | -35 | -3 |
| **n** | 70 | -2 | 23 | 41 | 67 | 63 | -5 | 42 | 53 | 59 | 65 | 36 | 67 | 89 | 47 |
| | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** |
| **m** | -12 | -20 | -51 | -44 | -23 | -1 | -58 | -71 | -67 | -91 | -1 | 20 | 13 | 2 | 5 |
| **n** | 77 | 66 | 113 | 109 | 84 | 64 | 127 | 143 | 134 | 187 | 64 | -3 | 21 | 46 | 48 |
| | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** |
| **m** | -4 | 29 | 1 | -6 | -9 | -6 | 23 | 3 | -5 | 5 | -12 | -2 | -15 | -11 | -3 |
| **n** | 71 | -5 | 45 | 58 | 67 | 66 | -7 | 26 | 42 | 31 | 79 | 45 | 67 | 62 | 54 |
| | **75** | **76** | **77** | **78** | **79** | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** |
| **m** | -4 | -13 | -3 | -26 | -18 | -9 | -30 | -24 | -43 | -6 | 4 | 34 | 21 | 14 | 5 |
| **n** | 70 | 68 | 100 | 91 | 84 | 83 | 103 | 90 | 122 | 66 | 59 | -11 | 10 | 25 | 44 |
| | **90** | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** |
| **m** | -4 | 40 | 19 | -11 | -6 | -11 | 12 | 17 | -6 | 11 | -11 | 1 | -26 | -28 | -2 |
| **n** | 69 | -33 | 8 | 65 | 59 | 68 | -2 | -10 | 34 | 14 | 70 | 27 | 71 | 79 | 47 |
| | **105** | **106** | **107** | **108** | **109** | **110** | **111** | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** |
| **m** | 6 | -23 | -47 | -55 | -21 | 1 | -38 | -34 | -45 | 10 | 9 | 41 | 32 | 14 | 17 |
| **n** | 47 | 67 | 107 | 117 | 83 | 59 | 82 | 77 | 95 | 25 | 46 | -31 | -17 | 19 | 18 |
| | **120** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** |
| **m** | -2 | 18 | 10 | -2 | -7 | -6 | 19 | -5 | -7 | -4 | -23 | -3 | 2 | -32 | -16 |
| **n** | 65 | 22 | 33 | 55 | 64 | 67 | 11 | 50 | 53 | 54 | 99 | 51 | 41 | 102 | 79 |
| | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** | **144** | **145** | **146** | **147** | **148** | **149** |
| **m** | -8 | -21 | -26 | -33 | -4 | -31 | -34 | -25 | -43 | -6 | 3 | 23 | 12 | 11 | 8 |
| **n** | 77 | 84 | 91 | 104 | 61 | 122 | 110 | 96 | 124 | 70 | 60 | 12 | 30 | 33 | 40 |
| | **150** | **151** | **152** | **153** | **154** | **155** | **156** | **157** | **158** | **159** | **160** | **161** | **162** | **163** | **164** |
| **m** | -2 | 40 | 0 | -5 | 7 | -39 | 31 | 1 | -35 | -32 | -15 | 16 | -43 | -75 | -8 |
| **n** | 66 | -20 | 46 | 54 | 37 | 116 | -27 | 22 | 82 | 85 | 72 | 0 | 102 | 152 | 55 |
| | **165** | **166** | **167** | **168** | **169** | **170** | **171** | **172** | **173** | **174** | **175** | **176** | **177** | **178** | **179** |
| **m** | -9 | -12 | -84 | -93 | 25 | -104 | -40 | -51 | 110 | 3 | 17 | 55 | 28 | -5 | 29 |
| **n** | 68 | 54 | 171 | 186 | 12 | 222 | 92 | 93 | -169 | 52 | 33 | -45 | 1 | 55 | -4 |

**Table 9-43 – Values of variable m and n for coeff_abs_level_greater2_flag ctxIdx**

| Initialisation variables | coeff_abs_level_greater2_flag ctxIdx | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** |
| **m** | -12 | -10 | -11 | -14 | -35 | -10 | -1 | -17 | -5 | -22 | -13 | -2 | -13 | -21 | -3 |
| **n** | 72 | 79 | 87 | 94 | 136 | 58 | 54 | 86 | 70 | 105 | 70 | 59 | 81 | 96 | 73 |
| | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** | **25** | **26** | **27** | **28** | **29** |
| **m** | -24 | -19 | -1 | 1 | -17 | -7 | -20 | -11 | -27 | -12 | -9 | -8 | -4 | -3 | -9 |
| **n** | 90 | 88 | 63 | 60 | 97 | 69 | 95 | 84 | 110 | 96 | 72 | 76 | 75 | 76 | 94 |
| | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** |
| **m** | -11 | -4 | -9 | -24 | -24 | -18 | -10 | -19 | -28 | 6 | -12 | -17 | -8 | -44 | -17 |
| **n** | 65 | 66 | 82 | 107 | 111 | 58 | 61 | 82 | 97 | 47 | 50 | 73 | 66 | 115 | 86 |
| | **45** | **46** | **47** | **48** | **49** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** |
| **m** | -26 | -4 | -43 | -58 | -51 | -51 | -85 | -47 | -95 | -65 | -2 | 7 | -2 | 3 | 4 |
| **n** | 74 | 48 | 115 | 141 | 137 | 117 | 176 | 120 | 202 | 159 | 53 | 43 | 67 | 62 | 67 |
| | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** |
| **m** | -2 | -13 | -11 | -13 | -24 | -4 | -17 | -16 | -19 | -28 | -12 | -11 | -18 | -6 | -19 |
| **n** | 49 | 81 | 82 | 88 | 112 | 44 | 77 | 82 | 89 | 110 | 64 | 70 | 88 | 67 | 97 |
| | **75** | **76** | **77** | **78** | **79** | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** |
| **m** | -17 | -27 | -18 | -13 | -17 | -11 | -15 | -10 | -10 | -12 | -8 | -7 | -6 | -8 | -9 |
| **n** | 76 | 100 | 88 | 84 | 94 | 73 | 83 | 77 | 80 | 91 | 63 | 71 | 73 | 80 | 90 |
| | **90** | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** | **100** | **101** | **102** | **103** | **104** |
| **m** | 14 | -7 | -4 | -8 | -50 | 2 | 11 | -4 | -41 | -60 | -12 | -6 | 1 | -5 | -12 |
| **n** | 20 | 68 | 66 | 76 | 148 | 22 | 15 | 49 | 117 | 149 | 49 | 55 | 44 | 58 | 73 |
| | **105** | **106** | **107** | **108** | **109** | **110** | **111** | **112** | **113** | **114** | **115** | **116** | **117** | **118** | **119** |
| **m** | 13 | 5 | -29 | 0 | -23 | -12 | 6 | 35 | 42 | 22 | 24 | 11 | 2 | -5 | 4 |
| **n** | 9 | 26 | 101 | 46 | 92 | 57 | 30 | -4 | -24 | 24 | 0 | 34 | 61 | 75 | 62 |
| | **120** | **129** | **130** | **131** | **132** | **133** | **134** | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** |
| **m** | 0 | -6 | -7 | -31 | -25 | -20 | -14 | -20 | -21 | -6 | -19 | -34 | -27 | -7 | -7 |
| **n** | 43 | 65 | 71 | 117 | 109 | 76 | 73 | 88 | 92 | 71 | 73 | 108 | 101 | 69 | 75 |
| | **135** | **136** | **137** | **138** | **139** | **140** | **141** | **142** | **143** | **144** | **145** | **146** | **147** | **148** | **149** |
| **m** | -18 | -7 | -20 | -9 | -7 | -26 | -13 | -6 | -12 | -2 | -3 | -6 | -7 | -6 | -11 |
| **n** | 77 | 64 | 91 | 75 | 78 | 98 | 81 | 69 | 83 | 70 | 50 | 66 | 73 | 75 | 91 |
| | **150** | **151** | **152** | **153** | **154** | **155** | **156** | **157** | **158** | **159** | **160** | **161** | **162** | **163** | **164** |
| **m** | 19 | 2 | 4 | -4 | -32 | 13 | -29 | -64 | -90 | 26 | -24 | 8 | -10 | -16 | -27 |
| **n** | 10 | 49 | 52 | 69 | 114 | -1 | 85 | 143 | 186 | 4 | 68 | 30 | 61 | 78 | 96 |
| | **165** | **166** | **167** | **168** | **169** | **170** | **171** | **172** | **173** | **174** | **175** | **176** | **177** | **178** | **179** |
| **m** | 31 | -12 | -15 | -68 | 31 | -8 | -32 | -36 | -53 | 36 | 26 | 19 | -22 | 5 | 8 |
| **n** | -16 | 54 | 70 | 158 | 12 | 44 | 63 | 81 | 106 | 12 | -5 | 17 | 101 | 54 | 53 |

Deleted: 9-43

Deleted: 9-43

### 9.3.1.2 Initialisation process for the arithmetic decoding engine

This process is invoked before decoding the first treeblock of a slice.

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 510 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 result in a value of codIOffset being equal to 510 or 511.

NOTE – The description of the arithmetic decoding engine in this Recommendation | International Standard utilizes 16 bit register precision. However, a minimum register precision of 9 bits is required for storing the values of the variables codIRange and

codIOffset after invocation of the arithmetic decoding process (DecodeBin) as specified in subclause 9.3.3.2. The arithmetic decoding process for a binary decision (DecodeDecision) as specified in subclause 9.3.3.2.1 and the decoding process for a binary decision before termination (DecodeTerminate) as specified in subclause 9.3.3.2.4 require a minimum register precision of 9 bits for the variables codIRange and codIOffset. The bypass decoding process for binary decisions (DecodeBypass) as specified in subclause 9.3.3.2.3 requires a minimum register precision of 10 bits for the variable codIOffset and a minimum register precision of 9 bits for the variable codIRange.

### 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 9-31 specifies the type of binarization process, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset associated with each syntax element.

The specification of the unary (U) binarization process, the truncated unary (TU) binarization process, the Rice (R) binarization process, the truncated Rice (TR), the concatenated unary / k-th order Exp-Golomb (UEGk) binarization process, the concatenated truncated Rice / k-th order Exp-Golomb (TREGk) binarization process, and the fixed-length (FL) binarization process are given in subclauses 0, to 9.3.2.6, respectively. Other binarizations are specified in subclauses 9.3.2.7 to 9.3.2.10.

Deleted: 9.3.2.1

Field Code Changed

The binarizations for the syntax element coeff_abs_level_minus3 as specified in subclause 9.3.2.10 consist of bin strings given by a concatenation of prefix and suffix bit strings. The UEGk binarization as specified in subclause 9.3.2.3, which is used for the binarization of the syntax elements mvd_lX (X = 0, 1 C) 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 table (ctxIdxTable) variable and the related context index offset (ctxIdxOffset) variable and a specific value of the maxBinIdxCtx variable as given in **Error! Reference source not found.** When two values for each of these variables are specified for one syntax element in **Error! Reference source not found.** 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.

Deleted: Table 9-27

Deleted: Table 9-27

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 **Error! Reference source not found.** 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.

Deleted: Table 9-27

– Otherwise, for each possible value of binIdx up to the specified value of maxBinIdxCtx given in **Error! Reference source not found.** a specific value of the variable ctxIdx is further specified in subclause 9.3.3. bypassFlag is set equal to 0.

Deleted: Table 9-27

The possible values of the context index ctxIdx vary depending on the value of ctxIdxTable. 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.

ctxIdxTable = 0 and ctxIdx = ctxIdxOffset = 0 are assigned to the syntax element end_of_slice_flag 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.

**Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

| Syntax element | | Type of binarization | maxBinIdx Ctx | ctxIdxTable | ctxIdxOffset |
|---|---|---|---|---|---|
| alf_cu_flag | I | FL, cMax = 1 | 0 | Table 9-18 | 0 |
| | P | | 0 | Table 9-18 | 3 |
| | B | | 0 | Table 9-18 | 6 |
| end_of_slice_flag | all | FL, cMax = 1 | 0 | 0 | 0 |
| split_coding_unit_flag | I | FL, cMax = 1 | 0 | Table 9-19 | 0 |
| | P | | 0 | Table 9-19 | 3 |
| | B | | 0 | Table 9-19 | 6 |
| skip_flag | P | FL, cMax = 1 | 0 | Table 9-20 | 0 |
| | B | | 0 | Table 9-20 | 3 |
| cu_qp_delta | I | as specified in subclause 9.3.2.7 | 2 | Table 9-21 | 0 |
| | P | | 2 | Table 9-21 | 4 |
| | B | | 2 | Table 9-21 | 8 |
| pred_type | I | as specified in subclause 9.3.2.8 | 0 | Table 9-22 | 0 |
| | P | | 3 | Table 9-22 | 1 |
| | B | | 4 | Table 9-22 | 5 |
| pcm_flag | all | FL, cMax = 1 | 0 | 0 | 0 |
| prev_intra_luma_pred_flag | I | FL, cMax = 1 | 0 | Table 9-23 | 0 |
| | P | | 0 | Table 9-23 | 1 |
| | B | | 0 | Table 9-23 | 2 |
| rem_intra_luma_pred_mode | I | as specified in subclause 9.3.2.9 | 0 | Table 9-24 | 0 |
| | P | | 0 | Table 9-24 | 1 |
| | B | | 0 | Table 9-24 | 2 |
| intra_chroma_pred_mode ( IntraPredMode < 4 ) | I | TU, cMax = 3 | 3 | Table 9-25 | 0 |
| | P | | 3 | Table 9-25 | 4 |
| | B | | 3 | Table 9-25 | 8 |
| intra_chroma_pred_mode ( 4 <= IntraPredMode < 34 ) | I | TU, cMax = 4 | 4 | Table 9-25 | 0 |
| | P | | 4 | Table 9-25 | 4 |
| | B | | 4 | Table 9-25 | 8 |
| merge_flag | P | FL, cMax = 1 | 0 | Table 9-26 | 0 |
| | B | | 0 | Table 9-26 | 3 |
| merge_idx | P | TU, cMax = NumMergeCand | 3 | Table 9-27 | 0 |
| | B | | 3 | Table 9-27 | 4 |
| inter_pred_flag | B | FL, cMax = 1 | 0 | Table 9-28 | 0 |



Deleted: 9-44
Deleted: 9-44
Deleted: Table 9-5
Formatted ... [79]
Deleted: Table 9-5
Formatted ... [80]
Deleted: Table 9-5
Formatted ... [81]
Deleted: Table 9-6
Formatted ... [82]
Deleted: Table 9-6
Formatted ... [83]
Deleted: Table 9-6
Formatted ... [84]
Deleted: Table 9-7
Formatted ... [85]
Deleted: Table 9-7
Formatted ... [86]
Deleted: Table 9-8
Formatted ... [87]
Deleted: Table 9-8
Formatted ... [88]
Deleted: Table 9-8
Formatted ... [89]
Deleted: Table 9-9
Formatted ... [90]
Deleted: Table 9-9
Formatted ... [91]
Deleted: Table 9-9
Formatted ... [92]
Deleted: Table 9-10
Formatted ... [93]
Deleted: Table 9-10
Formatted ... [94]
Deleted: Table 9-10
Formatted ... [95]
Deleted: Table 9-11
Formatted ... [96]
Deleted: Table 9-11
Formatted ... [97]
Deleted: Table 9-11
Formatted ... [98]
Deleted: Table 9-12
Formatted ... [99]
Deleted: Table 9-12
Formatted ... [100]
Deleted: Table 9-12
Formatted ... [101]
Deleted: Table 9-12
Formatted ... [102]
Deleted: Table 9-12
Formatted ... [103]
Deleted: Table 9-12
Formatted ... [104]
Deleted: Table 9-13
Formatted ... [105]
Deleted: Table 9-13
Formatted ... [106]
Deleted: Table 9-14
Formatted ... [107]
Deleted: Table 9-14
Formatted ... [108]
Deleted: Table 9-15
Formatted ... [109]

**Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

| Syntax element | | Type of binarization | maxBinIdx Ctx | ctxIdxTable | ctxIdxOffset |
|---|---|---|---|---|---|
| ref_idx_lc | B | TU, cMax = num_ref_idx_lC_active_minus1 | 1 | Table 9-29 | 6 |
| ref_idx_l0 | P | TU, cMax = num_ref_idx_l0_active_minus1 | 1 | Table 9-29 | 0 |
| | B | | 1 | Table 9-29 | 6 |
| ref_idx_l1 | B | TU, cMax = num_ref_idx_l1_active_minus1 | 1 | Table 9-29 | 6 |
| mvd_l0[ ][ ][ 0 ] | P | prefix and suffix as given by UEG3 with signedValFlag=1, uCoff=9 | prefix: 4 suffix: na | prefix: Table 9-30 suffix: na | prefix: 0 suffix: na, (uses Decode Bypass) |
| mvd_l0[ ][ ][ 1 ] | P | | prefix:4 suffix: na | prefix: Table 9-30 suffix: na | prefix: 7 suffix: na, (uses Decode Bypass) |
| mvd_l0[ ][ ][ 0 ], mvd_lc[ ][ ][ 0 ], mvd_l1[ ][ ][ 0 ] | B | | prefix: 4 suffix: na | prefix: Table 9-30 suffix: na | prefix: 15 suffix: na, (uses Decode Bypass) |
| mvd_l0[ ][ ][ 1 ], mvd_lc[ ][ ][ 1 ], mvd_l1[ ][ ][ 1 ] | B | | prefix: 4 suffix: na | prefix: Table 9-30 suffix: na | prefix: 21 suffix: na, (uses Decode Bypass) |
| mvp_idx_lc | B | TU, cMax = NumMVPCand( LcToLx ) | 1 | Table 9-31 | 2 |
| mvp_idx_l0 | P | TU, cMax = NumMVPCand( L0 ) | 1 | Table 9-31 | 0 |
| | B | | 1 | Table 9-31 | 2 |
| mvp_idx_l1 | B | TU, cMax = NumMVPCand( L1 ) | 1 | Table 9-31 | 2 |
| no_residual_data_flag | P | FL, cMax = 1 | 0 | Table 9-32 | 0 |
| | B | | 0 | Table 9-32 | 4 |
| split_transform_flag | I | FL, cMax = 1 | 0 | Table 9-33 | 0 |
| | P | | 0 | Table 9-33 | 4 |
| | B | | 0 | Table 9-33 | 8 |
| cbf_luma | I | FL, cMax = 1 | 0 | Table 9-34 | 0 |
| | P | | 0 | Table 9-34 | 4 |
| | B | | 0 | Table 9-34 | 8 |
| cbf_cb | I | FL, cMax = 1 | 0 | Table 9-35 | 0 |
| | P | | 0 | Table 9-35 | 4 |
| | B | | 0 | Table 9-35 | 8 |
| cbf_cr | I | FL, cMax = 1 | 0 | Table 9-36 | 0 |
| | P | | 0 | Table 9-36 | 4 |
| | B | | 0 | Table 9-36 | 8 |
| last_significant_coeff_x | I | TU, cMax = ( 1 << log2TrafoSize ) − 1 | 30 | Table 9-37 | 0 |
| | P | | 30 | Table 9-37 | 41 |



Deleted: 9-44
Deleted: 9-44
Formatted ... [110]
Deleted: Table 9-16
Formatted ... [111]
Deleted: Table 9-16
Formatted ... [112]
Deleted: Table 9-16
Formatted ... [113]
Deleted: Table 9-16
Formatted ... [114]
Deleted: Table 9-17
Formatted ... [115]
Deleted: Table 9-17
Formatted ... [116]
Deleted: Table 9-17
Formatted ... [117]
Deleted: Table 9-17
Deleted: Table 9-18
Formatted ... [118]
Deleted: Table 9-18
Formatted ... [119]
Deleted: Table 9-18
Formatted ... [120]
Deleted: Table 9-18
Formatted ... [121]
Deleted: Table 9-19
Formatted ... [122]
Formatted ... [123]
Deleted: Table 9-19
Formatted ... [124]
Deleted: Table 9-20
Formatted ... [125]
Deleted: Table 9-20
Formatted ... [126]
Deleted: Table 9-20
Formatted ... [127]
Deleted: Table 9-21
Formatted ... [128]
Deleted: Table 9-21
Formatted ... [129]
Deleted: Table 9-21
Formatted ... [130]
Deleted: Table 9-22
Formatted ... [131]
Deleted: Table 9-22
Formatted ... [132]
Deleted: Table 9-22
Deleted: Table 9-23
Formatted ... [133]
Deleted: Table 9-23
Formatted ... [134]
Deleted: Table 9-23
Formatted ... [135]
Deleted: Table 9-24
Formatted ... [136]
Deleted: Table 9-24
Formatted ... [137]

**Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

| Syntax element | | Type of binarization | maxBinIdxCtx | ctxIdxTable | ctxIdxOffset |
|---|---|---|---|---|---|
| | B | | 30 | Table 9-37 | 82 |

Deleted: 9-44
Deleted: 9-44
Deleted: Table 9-24
Formatted: Font: 8 pt

**Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

| Syntax element | | Type of binarization | maxBinIdxCtx | ctxIdxTable | ctxIdxOffset |
|---|---|---|---|---|---|
| last_significant_coeff_y | I | TU, cMax = ( 1 << log2TrafoSize ) − 1 | 30 | Table 9-38 | 0 |
| | P | | 30 | Table 9-38 | 41 |
| | B | | 30 | Table 9-38 | 82 |
| significant_coeff_flag | I | FL, cMax = 1 | 0 | Table 9-39 | 0 |
| | P | | 0 | Table 9-40 | 0 |
| | B | | 0 | Table 9-41 | 0 |
| coeff_abs_level_greater1_flag | I | FL, cMax = 1 | 0 | Table 9-42 | 0 |
| | P | | 0 | Table 9-42 | 60 |
| | B | | 0 | Table 9-42 | 120 |
| coeff_abs_level_greater2_flag | I | FL, cMax = 1 | 0 | Table 9-43 | 0 |
| | P | | 0 | Table 9-43 | 60 |
| | B | | 0 | Table 9-43 | 120 |

significant_coeff

| | | |
|---|---|---|
| 5 | **6** | **7** |
| 0 | 0 | 0 |
| 4 | 64 | 64 |
| **1** | **22** | **23** |
| 0 | 0 | 0 |
| 4 | 64 | 64 |
| **7** | **38** | **39** |
| 2 | -32 | -19 |
| 5 | 125 | 72 |
| **3** | **54** | **55** |
| 0 | 0 | 0 |
| 4 | 64 | 64 |
| **9** | **70** | **71** |
| 2 | -8 | 1 |
| 6 | 80 | 36 |
| **5** | **86** | **87** |
| 7 | -11 | -29 |
| 2 | 81 | 92 |
| **01** | **102** | **103** |
| 3 | -18 | -10 |
| 6 | 99 | 59 |



Deleted: 9-44
Deleted: 9-44
Deleted: Table 9-25
Formatted: Font: 8 pt
Deleted: Table 9-25
Formatted: Font: 8 pt
Deleted: Table 9-25
Formatted: Font: 8 pt
Deleted: Table 9-26
Formatted: Font: 8 pt
Formatted: Font: Times New Roman, 8 pt, English (UK)
Formatted: Table_Text, Left, Space Before: 0 pt, After: 0 pt, Line spacing: single, Don't keep with next, Don't keep lines together
Formatted Table
Deleted: ¶ Page Break Table 9-27
Formatted: Font: 8 pt
Deleted: Page Break Table 9-28
Formatted: Font: 8 pt
Formatted: Table_Text, Centered, Keep with next
Deleted: Page Break Table 9-29
Formatted: Font: 8 pt
Formatted: Table_Text, Centered, Keep with next
Deleted: Page Break Table 9-29
Formatted: Font: 8 pt
Formatted: Table_Text, Centered, Keep with next
Deleted: Page Break Table 9-29
Deleted: Table 9-30
Formatted: Font: 8 pt
Deleted: Table 9-30
Formatted: Font: 8 pt
Deleted: Table 9-30
Formatted: Font: 8 pt

**Table 9-44 – Syntax elements and associated types of binarization, maxBinIdxCtx, ctxIdxTable, and ctxIdxOffset**

| Syntax element | | Type of binarization | maxBinIdx Ctx | ctxIdxTable | ctxIdxOffset |
|---|---|---|---|---|---|
| coeff_abs_level_minus3 | all | prefix and suffix as specified in subclause 9.3.2.10 | prefix: na<br>suffix: na | prefix: na<br>suffix: na | prefix: na, (uses Decode Bypass)<br>suffix: na, (uses Decode Bypass) |
| coeff_sign_flag | all | FL, cMax = 1 | na | na | na, (uses Decode Bypass) |

[Ed. (BB): The TU binarization for last_significant_coeff_x and last_significant_coeff_y is using 0s and a terminating 1 instead of using 1s and a terminating 0.]

#### 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 9-45 illustrates the bin strings of the unary binarization for a syntax element.

**Table 9-45 – 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 0 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 Truncated Rice (TR) binarization process

Input to this process is a request for a TR binarization for a syntax element, cRiceParam and cTRMax.

Output of this process is the TR binarization of the syntax element.

A TR 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 of the value specified by synElVal >> cRiceParam with



Deleted: 9-44
Deleted: 9-44
Deleted: Table 9-35
Field Code Changed
Deleted: 9-45
Deleted: 9-45
Deleted: 9.3.2.1
Field Code Changed

cMax = cTRMax >> cRiceParam. The suffix of the TR bin string is the binary representation of synElVal – ( ( synElVal >> cRiceParam ) << cRiceParam ).

NOTE – For the input parameter cRiceParam = 0 the TR binarization is exactly the TU binarization.

#### 9.3.2.4 k-th order Exp-Golomb (EGk) binarization process

Input to this process is a request for an EGk binarization for a syntax element and signedFlag.

Output of this process is the EGk binarization of the syntax element.

The bin string of the EGk binarization process of a syntax element synVal is specified by a process equivalent to the following pseudo-code:

```
absV = Abs( synVal )
stopLoop = 0
do {
    if( absV >= ( 1 << k ) ) {
        put( 1 )
        absV = absV − ( 1 << k )
        k++
    } else {
        put( 0 )
    (9-50)
        while( k− − )
            put( ( absV >> k ) & 1 )
        stopLoop = 1
    }
} while( !stopLoop )
if( signedFlag && synVal != 0) {
    if( synVal > 0 )
        put( 0 )
    else
        put( 1 )
}
```

NOTE 1 – 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.

#### 9.3.2.5 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 variable k for a UEGk bin string is dependent on the syntax element for which a UEGk binarization is requested. **Error! Reference source not found.** specifies the associated types of binarization for syntax elements, including the value of k for syntax elements that use UEGk binarization.

NOTE 1 – For the syntax elements mvd_lc[ ][ ][ ], mvd_l0[ ][ ][ ] and mvd_l1[ ][ ][ ] a UEG3 binarization is used (k is equal to 3).

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.

Deleted: 5

Field Code Changed

Field Code Changed

Deleted: Table 9-34

– Otherwise, the bin string of the UEGk suffix part of a syntax element value synElVal is specified by invoking the EGk binarization process as specified in subclause 9.3.2.4 with k being initialised to the value that is specified in for the requested UEGk binarization process.

Deleted: Table 9-4

Formatted: Highlight

#### 9.3.2.6 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 a 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.

#### 9.3.2.7 Binarization process for cu_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.

For syntax elements mb_qp_delta, the U binarization process as specified in subclause 0 is invoked with the mapped value of the syntax element mb_qp_delta as input and the bin string as output. The assignment rule between the signed value of mb_qp_delta and its mapped value is given as specified in Table 9-3.

Deleted: 9.3.2.1

Deleted: Table 9-3

Field Code Changed

#### 9.3.2.8 Binarization process for pred_type

Input to this process is a request for a binarization for the syntax element pred_type and a variable cLog2CUSize specifying the current CU size.

Output of this process is the binarization of the syntax element.

The binarization for pred_type is given by Table 9-46 depending on slice type and the size of the coding unit.

Deleted: Table 9-6

**Table 9-46 – Binarization for pred_type**

Deleted: 9-46

Deleted: 9-46

| Slice type | Value of pred_type | PredMode | PartMode | Bin string | |
|---|---|---|---|---|---|
| | | | | cLog2CUSize > 3 | cLog2CUSize = = 3e |
| I | 0 | MODE_INTRA | PART_2Nx2N | - | 1 |
| | 1 | MODE_INTRA | PART_NxN | - | 0 |
| P | 0 | MODE_INTER | PART_2Nx2N | 01 | 01 |
| | 1 | MODE_INTER | PART_2NxN | 001 | 001 |
| | 2 | MODE_INTER | PART_Nx2N | 000 | 0001 |
| | 3 | MODE_INTER | PART_NxN | - | 0000 |
| | 4 | MODE_INTRA | PART_2Nx2N | 1 | 11 |
| | 5 | MODE_INTRA | PART_NxN | - | 10 |
| B | 0 | MODE_INTER | PART_2Nx2N | 1 | 1 |
| | 1 | MODE_INTER | PART_2NxN | 01 | 01 |
| | 2 | MODE_INTER | PART_Nx2N | 001 | 001 |
| | 3 | MODE_INTER | PART_NxN | - | 0001 |
| | 4 | MODE_INTRA | PART_2Nx2N | 000 | 0000 1 |
| | 5 | MODE_INTRA | PART_NxN | - | 0000 0 |

#### 9.3.2.9 Binarization process for rem_intra_luma_pred_mode

Input to this process is a request for the syntax element rem_intra_luma_pred_mode and cNumBins.

Output of this process is the binarization of the syntax element.

The binarization for rem_intra_luma_pred_mode is given by Table 9-47.

Deleted: Table 9-35

**Table 9-47 – Binarization for rem_intra_luma_pred_mode**

Deleted: 9-47

Deleted: 9-47

| Value of rem_intra_luma_pred_mode | Bin string |
|---|---|
| less than 32 | FL, cMax = cNumBins |
| 32 | 111110 |
| 33 | 111111 |

#### 9.3.2.10 Binarization process for coeff_abs_level_minus3

Input to this process is a request for a binarization for the syntax element coeff_abs_level_minus3[ n ].

Output of this process is the binarization of the syntax element.

The variables cLastSE and cLastRiceParam are derived as follows.

– If n is equal to 0 or all previous syntex elements coeff_abs_level_minus3[ pos ] with pos less than n are derived to be equal to 0 instead of beeing explicitly parsed, cLastSE and cLastRiceParam are set equal to 0.

– Otherwise ( n is not equal to 0 ), cLastSE is set equal to coeff_abs_level_minus3[ n − 1 ] and cLastRiceParam is set equal to the value of cRiceParam that has been derived during the invocation of this subclause for the syntax element coeff_abs_level_minus2[ n – 1 ] of the same transform block.

The variable cRiceParam is derived as follows.

– If cLastSE is equal to 0 or 1 and cLastRiceParam is equal to 0, cRiceParam is set equal to 0.

– Otherwise if cLastSE is equal to 2 or 3 and cLastRiceParam is equal to 0 or 1, cRiceParam is set equal to 1.

– Otherwise if one of the following conditions is true, cRiceParam is set equal to 2.

  – cLastSE is in the range of 4 to 12 and cLastRiceParam is equal to 0.

  – cLastSE is in the range of 4 to 10 and cLastRiceParam is equal to 1.

  – cLastSE is in the range of 4 to 9 and cLastRiceParam is equal to 2.

– Otherwise, cRiceParam is set equal to 3.

The variable cTRMax is derived from cRiceParam as given in Table 9-48.

Deleted: Table 9-36

**Table 9-48 – Specifcation of cTRMax depending on cRiceParam**

Deleted: 9-48

Deleted: 9-48

| cRiceParam | cTRMax |
|---|---|
| 0 | 7 |
| 1 | 20 |
| 2 | 42 |
| 3 | 70 |

The binarization of coeff_abs_level_minus3 consists of a prefix part and (when present) a suffix part. The prefix part of the binarization is derived by invoking the TR binarization process as specified in subclause 9.3.2.3 for the syntax element coeff_abs_level_minus3[ n ] with the variables cRiceParam and cTRMax as the inputs.

When the prefix bit string is not equal to the bin string that consists of ( cTRMax + 1 >> cRiceParam ) + cRiceParam ones, the bin string consists of a prefix bin string and a suffix bin string. The suffix bin string is derived using the EG0 binarization as specified in subclause 9.3.2.4.

### 9.3.3 Decoding process flow

Input to this process is a binarization of the requested syntax element, maxBinIdxCtx, bypassFlag, ctxIdxTable 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 9.3.2.5 and 9.3.2.10 is invoked depending on the resulting prefix bit string as specified in subclauses 9.3.2.5 and 9.3.2.10.

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, ctxIdxTable, and ctxIdxOffset, ctxIdx is derived as specified in subclause 9.3.3.1.
2. Given ctxIdxTable and 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, ctxIdxTable, and ctxIdxOffset.

Output of this process is ctxIdx.

Table 9-49 shows the assignment of ctxIdx increments (ctxIdxInc) to binIdx for all ctxIdxTable and ctxIdxOffset.

The ctxIdx to be used with a specific binIdx is specified by first determining the ctxIdxTable and ctxIdxOffset associated with the given bin string or part thereof. The ctxIdxOffset is listed in Table 9-49, the ctxIdx for a binIdx is the sum of ctxIdxOffset and ctxIdxInc, which is found in Table 9-49. When more than one value is listed in Table 9-49 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.

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 9-49 labelled with "na" correspond to values of binIdx that do not occur for the corresponding ctxIdxOffset.

**Deleted:** Table 9-37

**Deleted:** Table 9-37

**Deleted:** Table 9-37

**Deleted:** Table 9-37

**Deleted:** Table 9-37

**Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values**

| Syntax element | ctxIdxTable, ctxIdxOffset | | binIdx | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | >=4 |
| alf_cu_flag | Table 9-18 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 3 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 6 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| split_coding_unit_flag | Table 9-19 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 3 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 6 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| skip_flag | Table 9-20 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 3 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| cu_qp_delta | Table 9-21 | 0 | 0 | 2 | 3 | 3 | 3 |
| | | 4 | 0 | 2 | 3 | 3 | 3 |
| | | 8 | 0 | 2 | 3 | 3 | 3 |
| pred_type | Table 9-22 | 0 | 0 | na | na | na | na |
| | | 1 | 0 | 1 | 2 | 3 | na |
| | | 5 | 0 | 1 | 2 | 3 | 4 |
| prev_intra_luma_pred_flag | Table 9-23 | 0 | 0 | na | na | na | na |
| | | 1 | 0 | na | na | na | na |
| | | 2 | 0 | na | na | na | na |
| rem_intra_luma_pred_mode | Table 9-24 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 1 | 0 | 0 | 0 | 0 | 0 |
| | | 2 | 0 | 0 | 0 | 0 | 0 |
| intra_chroma_pred_mode | Table 9-25 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 3 | 3 | na |
| | | 4 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 3 | 3 | na |
| | | 8 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 3 | 3 | na |
| merge_flag | Table 9-26 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 3 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |
| merge_idx | Table 9-27 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.2) | 0,1,2,3 (subclause 9.3.3.1.1.2) | 0,2,3 (subclause 9.3.3.1.1.2 ) | 3 | na |
| | | 4 | 0,1,2,3 (subclause 9.3.3.1.1.2) | 0,1,2,3 (subclause 9.3.3.1.1.2) | 0,2,3 (subclause 9.3.3.1.1.2 ) | 3 | na |
| inter_pred_flag | Table 9-28 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | na | na | na | na |

Deleted: 9-49
Deleted: 9-49
Deleted: Table 9-5
Formatted: Font: 8 pt
Deleted: Table 9-6
Formatted: Font: 8 pt
Deleted: Table 9-7
Formatted: Font: 8 pt
Deleted: Table 9-8
Formatted: Font: 8 pt
Deleted: Table 9-9
Formatted: Font: 8 pt
Deleted: Table 9-10
Formatted: Font: 8 pt
Deleted: Table 9-11
Formatted: Font: 8 pt
Deleted: Table 9-12
Formatted: Font: 8 pt
Deleted: Table 9-13
Formatted: Font: 8 pt
Deleted: Table 9-14
Formatted: Font: 8 pt
Deleted: Table 9-15
Formatted: Font: 8 pt

**Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values**

| Syntax element | ctxIdxTable, ctxIdxOffset | | binIdx | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | >=4 |
| ref_idx_l0 | Table 9-29 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.1) | 4 | 4 | 4 | 4 |
| ref_idx_l0, ref_idx_l1, ref_idx_lc | Table 9-29 | 6 | 0,1,2,3 (subclause 9.3.3.1.1.1) | 4 | 4 | 4 | 4 |
| mvd_l0[ ][ ][ 0 ] | Table 9-30 | 0 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 4 | 5 | 6 |
| mvd_l0[ ][ ][ 1 ] | Table 9-30 | 7 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 4 | 5 | 6 |
| mvd_l0[ ][ ][ 0 ], mvd_lc[ ][ ][ 0 ], mvd_l1[ ][ ][ 0 ] | Table 9-30 | 15 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 4 | 5 | 6 |
| mvd_l0[ ][ ][ 1 ], mvd_lc[ ][ ][ 1 ], mvd_l1[ ][ ][ 1 ] | Table 9-30 | 21 | 0,1,2 (subclause 9.3.3.1.1.1) | 3 | 4 | 5 | 6 |
| mvp_idx_l0 | Table 9-31 | 0 | 0 | 1 | 1 | 1 | 1 |
| mvp_idx_l0, mvp_idx_l1, mvp_idx_lc | Table 9-31 | 2 | 0 | 1 | 1 | 1 | 1 |
| no_residual_data_flag | Table 9-32 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 4 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| split_transform_flag | Table 9-33 | 0 | cuDepth + trafoDepth | na | na | na | na |
| | | 4 | cuDepth + trafoDepth | na | na | na | na |
| | | 8 | cuDepth + trafoDepth | na | na | na | na |
| cbf_luma | Table 9-34 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 4 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 8 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| cbf_cb ( PredMode == MODE_INTRA ) | Table 9-35 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 4 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 8 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| cbf_cb ( PredMode != MODE_INTRA ) | Table 9-35 | 0 | trafoDepth | na | na | na | na |
| | | 4 | trafoDepth | na | na | na | na |
| | | 8 | trafoDepth | na | na | na | na |
| cbf_cr ( PredMode == MODE_INTRA ) | Table 9-36 | 0 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 4 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| | | 8 | 0,1,2,3 (subclause 9.3.3.1.1.1) | na | na | na | na |
| cbf_cr ( PredMode != MODE_INTRA ) | Table 9-36 | 0 | trafoDepth | na | na | na | na |
| | | 4 | trafoDepth | na | na | na | na |
| | | 8 | trafoDepth | na | na | na | na |

Deleted: 9-49
Deleted: 9-49
Deleted: Table 9-16
Formatted: Font: 8 pt
Deleted: Table 9-16
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-17
Formatted: Font: 8 pt
Deleted: Table 9-18
Formatted: Font: 8 pt
Deleted: Table 9-18
Formatted: Font: 8 pt
Deleted: Table 9-19
Formatted: Font: 8 pt
Deleted: Table 9-20
Formatted: Font: 8 pt
Deleted: Table 9-21
Formatted: Font: 8 pt
Deleted: Table 9-22
Formatted: Font: 8 pt
Deleted: Table 9-22
Formatted: Font: 8 pt
Deleted: Table 9-23
Formatted: Font: 8 pt
Deleted: Table 9-23
Formatted: Font: 8 pt

**Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values**

| Syntax element | ctxIdxTable, ctxIdxOffset | | binIdx | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | >=4 |
| last_significant_coeff_x | Table 9-37 | 0 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| | | 41 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| | | 82 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| last_significant_coeff_y | Table 9-38 | 0 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| | | 41 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| | | 82 | 0..40 (subclause 9.3.3.1.1.3) | | | | |
| significant_coeff_flag | Table 9-39 | 0 | 0..111 (subclause 9.3.3.1.1.4) | na | na | na | na |
| | significant_c | 0 | 0..111 (subclause 9.3.3.1.1.4) | na | na | na | na |
| | 6 7<br>0 0<br>64 64<br>22 23<br>0 0<br>64 64<br>38 39<br>-32 -19<br>125 72<br>54 55<br>0 0<br>64 64<br>70 71<br>-8 1<br>80 36<br>86 87<br>-11 -29<br>81 92<br>102 103<br>-18 -10<br>99 59 | | | | | | |
| | Table 9-40 | | | | | | |
| | Table 9-41 | 0 | 0..111 (subclause 9.3.3.1.1.4) | na | na | na | na |
| coeff_abs_level_greater1_flag | Table 9-42 | 0 | 0..59 (subclause 9.3.3.1.1.5) | na | na | na | na |
| | | 60 | 0..59 (subclause 9.3.3.1.1.5) | na | na | na | na |
| | | 120 | 0..59 (subclause 9.3.3.1.1.5) | na | na | na | na |

Deleted: 9-49
Deleted: 9-49
Deleted: Table 9-24
Formatted: Font: 8 pt
Deleted: Table 9-25
Formatted: Font: 8 pt
Deleted: Table 9-26
Formatted: Font: 8 pt
Formatted: Font: Times New Roman, 8 pt, English (UK)
Formatted Table
Deleted: ¶
Page Break
Table 9-27
Deleted: Table 9-28
Formatted: Font: 8 pt
Formatted: Font: 8 pt
Deleted: Table 9-29

**Table 9-49 – Assignment of ctxIdxInc to binIdx for all ctxIdxTable and ctxIdxOffset values**

| Syntax element | ctxIdxTable, ctxIdxOffset | | binIdx | | | | |
|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | 3 | >=4 |
| coeff_abs_level_greater2_flag | Table 9-43 | 0 | 0..59 (subclause 9.3.3.1.1.6) | na | na | na | na |
| | | 60 | 0..59 (subclause 9.3.3.1.1.6) | na | na | na | na |
| | | 120 | 0..59 (subclause 9.3.3.1.1.6) | na | na | na | na |

### 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 elements alf_cu_flag, split_coding_unit_flag, skip_flag, merge_flag, intra_chroma_pred_mode, inter_pred_flag, ref_idx_lc, ref_idx_l0, ref_idx_l1, mvd_l0, mvd_l1, mvd_lc, no_residual_data_flag, cbf_luma, cbf_cb and cbf_cr.

#### 9.3.3.1.1.1 Derivation process of ctxIdxInc using left and above syntax elements

Input to this process is the luma location ( xP, yP ) specifying the top-left luma sample of the current prediction unit relative to the top-left sample of the current picture.

Output of this process is ctxIdxInc.

Let the luma location ( xL, yL ) specify a location covered by the prediction unit to the left of the top-left luma sample of the current prediction unit with xL = xP – MinPuSize and yL = yP, the variable availableL is derived as follows. [Ed.: (BB) MinPuSize should be defined somewhere]

– If the prediction unit covering location ( xL, yL ) is available, availableL is set equal to 1.

– Otherwise (the prediction unit covering location ( xL, yL ) is not available), availableL is set equal to 0.

Let the luma location ( xA, yA ) specify a location covered by the prediction unit above the top-left luma sample of the current prediction unit with xA = xP and yA = yP – MinPuSize, the variable availableA is derived as follows. [Ed.: (BB) MinPuSize should be defined somewhere]

– If the prediction unit covering location ( xA, yA ) is available, availableA is set equal to 1.

– Otherwise (the prediction unit covering location ( xA, yA ) is not available), availableA is set equal to 0

The assignment of ctxIdxInc for the syntax elements alf_cu_flag, split_coding_unit_flag, skip_flag, merge_flag, intra_chroma_pred_mode, inter_pred_flag, ref_idx_lc, ref_idx_l0, ref_idx_l1, mvd_lc, mvd_l0, mvd_l1, no_residual_data_flag, cbf_luma, cbf_cb and cbf_cr is specified in Table 9-50 – Specifcation of ctxIdxInc using left and above syntax elements.

Deleted: 9-49
Deleted: 9-49
Deleted: Table 9-30
Formatted: Font: 8 pt
Formatted: Heading 5, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"
Deleted: Table 9-38

Table 9-50 – Specifcation of ctxIdxInc using left and above syntax elements

Deleted: 9-50

Deleted: 9-50

| Syntax element | condL | condA | ctxIdxInc |
|---|---|---|---|
| alf_cu_flag | alf_cu_flag[ xL ][ yL ] | alf_cu_flag[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) |
| split_coding_unit_flag | cuDepth[ xL ][ yL ] > cuDepth[ xP ][ yP ] | cuDepth[ xA ][ yA ] > cuDepth[ xP ][ yP ] | ( condL && availableL ) + ( condA && availableA ) |
| skip_flag | skip_flag[ xL ][ yL ] | skip_flag[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) |
| merge_flag | merge_flag[ xL ][ yL ] | merge_flag[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) |
| intra_chroma_pred_mode | IntraPredMode[ xL ][ yL ] == 4 | IntraPredMode [ xA ][ yA ] == 4 | ( condL && availableL ) + ( condA && availableA ) |
| inter_pred_flag | inter_pred_flag[ xL ][ yL ] | inter_pred_flag[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) |
| ref_idx_lc | ref_idx_lc[ xL ][ yL ] > 0 | ref_idx_lc[ xA ][ yA ] > 0 | ( condL && availableL ) + ( condA && availableA ) << 1 |
| ref_idx_l0 | ref_idx_l0[ xL ][ yL ] > 0 | ref_idx_l0[ xA ][ yA ] > 0 | ( condL && availableL ) + ( condA && availableA ) << 1 |
| ref_idx_l1 | ref_idx_l1[ xL ][ yL ] > 0 | ref_idx_l1[ xA ][ yA ] > 0 | ( condL && availableL ) + ( condA && availableA ) << 1 |
| mvd_lc | mvd_lc[ xL ][ yL ] > 16 | mvd_lc[ xA ][ yA ] > 16 | ( condL && availableL ) + ( condA && availableA ) |
| mvd_l0 | mvd_l0[ xL ][ yL ] > 16 | mvd_l0[ xA ][ yA ] > 16 | ( condL && availableL ) + ( condA && availableA ) |
| mvd_l1 | mvd_l1[ xL ][ yL ] > 16 | mvd_l1[ xA ][ yA ] > 16 | ( condL && availableL ) + ( condA && availableA ) |
| no_residual_data_flag | no_residual_data_flag[ xL ][ yL ] | no_residual_data_flag[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) << 1 |
| cbf_luma | cbf_luma[ xL ][ yL ] | cbf_luma[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) << 1 |
| cbf_cb | cbf_cb[ xL ][ yL ] | cbf_cb[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) << 1 |
| cbf_cr | cbf_cr[ xL ][ yL ] | cbf_cr[ xA ][ yA ] | ( condL && availableL ) + ( condA && availableA ) << 1 |

#### 9.3.3.1.1.2 Derivation process of ctxIdxInc for the syntax element merge_index

Input to this process is binIdx.

Output of this process is ctxIdxInc.

The variable ctxIdxInc is derived as follows.

– If binIdx is equal to 0, the following applies.
  – If availableLeft is equal to 1and availableAbove is equal to 1, ctxIdxInc is set equal to 0.
  – Otherwise if availableLeft is equal to 1 or availableAbove is equal to 1, ctxIdxInc is set equal to 2.
  – Otherwise (availableLeft is equal to 0 and availableAbove is equal to 0), ctxIdxInc is set equal to 3.
– Otherwise if binIdx is equal to 1 and NumMergeCand is greater than 2, the following applies.
  – If availableAbove is equal to 1, ctxIdxInc is set equal to 2.
  – Otherwise (availableAbove is equal to 0), ctxIdxInc is set equal to 3.
– Otherwise if binIdx is equal to 2 and NumMergeCand is greater than 3, ctxIdxInc is set equal to 3.

– Otherwise, ctxIdxInc is set equal to 0.

When ctxIdxInc is not equal to 0, ctxIdxInc is derived as follows.

ctxIdxInc = ctxIdxInc − availableCollocated (9-51)

Deleted: 6

[Ed. (BB): The flags availableLeft, availableAbove, availableCollocated need to be defined or a simpler context derivation should be used.]

**9.3.3.1.1.3 Derivation process of ctxIdxInc for the syntax elements last_significant_coeff_x and last_significant_coeff_y**

Inputs to this process are the binIdx, the color component index cIdx and the transform block size log2TrafoSize.

Output of this process is ctxIdxInc.

**Table 9-51 – Specifcation of lastCtx depending on binIdx**

Deleted: 9-51

Deleted: 9-51

| binIdx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lastCtx | 0 | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 7 |
| binIdx | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | |
| lastCtx | 7 | 7 | 7 | 8 | 8 | 8 | 8 | 9 | 9 | 9 | 9 | 10 | 10 | 10 | 10 | |

The variable lastCtx is dervied from binIdx as given by Table 9-51. For the derivation of ctxIdxInc, the following applies.

Deleted: Table 9-39

– If log2TrafoSize is less than or equal to 2, ctxIdxInc is dervied as follows.

ctxIdxInc = lastCtx (9-52)

Deleted: 7

– Otherwise if log2TrafoSize is equal to 3, ctxIdxInc is dervied as follows.

ctxIdxInc = lastCtx + 3 (9-53)

Deleted: 8

– Otherwise if log2TrafoSize is equal to 4, ctxIdxInc is dervied as follows.

ctxIdxInc = lastCtx + 8 (9-54)

Deleted: 9

– Otherwise (log2TrafoSize is greater than 4), ctxIdxInc is dervied as follows.

ctxIdxInc = lastCtx + 15 (9-55)

Deleted: 10

When cIdx is greater than 0, ctxIdxInc is modified as follows.

ctxIdxInc = ctxIdxInc + 26 (9-56)

Deleted: 11

[Ed. (BB): The context derivation assumes maximum transform sizes less than or equal to 32x32 for luma and 16x16 for chroma and minimum transform sizes greater than or equal to 4x4.]

**9.3.3.1.1.4 Derivation process of ctxIdxInc for the syntax element significant_coeff_flag**

Inputs to this process are the color component index cIdx, the current coefficient scan position ( xC , yC ) and the transform block size log2TrafoSize.

Output of this process is ctxIdxInc.

The variable sigCtx depends on the current position ( xC, yC ), the transform block size and previsously decoded bins of the syntax element significant_coeff_flag. For the derivation of sigCtx, the following applies.

– If log2TrafoSize is less than or equal to 3, sigCtx is derived as follows.

shift = log2TrafoSize == 3 ? 1 : 0
sigCtx = ( ( yC >> shift ) << 2 ) + ( xC >> shift ) (9-57)

Deleted: 12

– Otherwise if xC is less than 2 and yC is less than 2, sigCtx is derived as follows.

sigCtx = ( yC << 1 ) + xC (9-58)

Deleted: 13

– Otherwise if yC is equal to 0, sigCtx is derived as follows.

sigCtx = 4 + significant_coeff_flag[ xC-1 ][ yC ] + significant_coeff_flag[ xC-2 ][ yC ] (9-59)

– Otherwise if xC is equal to 0, sigCtx is derived as follows.

sigCtx = 4 + significant_coeff_flag[ xC ][ yC-1 ] + significant_coeff_flag[ xC ][ yC-2 ] (9-60)

– Otherwise (xC is greater than 1 and yC is greater than 1) , the following applies.

– The variable sigCtx is initialized using previously decoded bins of the syntax element significant_coeff_flag as follows.

sigCtx = significant_coeff_flag[ xC-1 ][ yC ] + significant_coeff_flag[ xC ][ yC-1 ] + significant_coeff_flag[ xC-1 ][ yC-1 ] (9-61)

– When xC is greater than 1, the following applies.

sigCtx = sigCtx + significant_coeff_flag[ xC-2 ][ yC ] (9-62)

– When yC is greater than 1, the following applies.

sigCtx = sigCtx + significant_coeff_flag[ xC ][ yC-2 ] (9-63)

– The variable sigCtx is dervied as follows.

sigCtx = 10 + min( 4, sigCtx ) (9-64)

The context index increment ctxIdxInc is derived using the color component index cIdx, the transform block size log2TrafoSize and sigCtx as follows.

– If cIdx is equal to 0, ctxIdxInc is derived as follows.

ctxIdxInc = ( ( 5 − log2TrafoSize ) * 16 ) + sigCtx (9-65)

– Otherwise (cIdx is greater than 0), ctxIdxInc is derived as follows.

ctxIdxInc = 64 + ( ( 4 − log2TrafoSize ) * 16 ) + sigCtx (9-66)

[Ed. (BB): The context derivation assumes maximum transform sizes less than or equal to 32x32 for luma and 16x16 for chroma and minimum transform sizes greater than or equal to 4x4.]

#### 9.3.3.1.1.5 Derivation process of ctxIdxInc for the syntax element coeff_abs_level_greater1_flag

Inputs to this process are the color component index cIdx, the transform block size log2TrafoSize, the 4x4 subset index i and the current coefficient scan index n within the current subset.

Output of this process is ctxIdxInc.

The variable ctxSet specifies the current context set and for its derivation the following applies.

– If n is equal to 0 or all previous syntex elements coeff_abs_level_greater1_flag[ pos ] with pos less than n are derived to be equal to 0 instead of beeing explicitly parsed, the following applies.

– If the current transform block size log2TrafoSize is equal to 2, ctxSet is derived as follows.

ctxSet = 0 (9-67)

– Otherwise if the current 4x4 subset is the first one in scanning order (i is equal to 0), ctxSet is derived as follows.

ctxSet = 5 (9-68)

– Otherwise (i is not equal to 0), the variable lastGreater2Ctx is set equal to the variable greater2Ctx that has been derived during the last invocation of subclause 9.3.3.1.1.6 for the syntax element coeff_abs_level_greater2_flag for the subset i – 1 and ctxSet is derived as follows.

ctxSet = ( ( lastGreater2Ctx + 1 ) >> 2 ) + 1 (9-69)

Deleted: 14
Deleted: 15
Deleted: 16
Deleted: 17
Deleted: 18
Deleted: 19
Deleted: 20
Deleted: 21
Deleted: 22
Deleted: 23
Deleted: 24

- The variable greater1Ctx is set equal to 1.

- Otherwise (coeff_abs_level_greater1_flag[ n ] is not the first to be parsed within the current subset i),.for the derivation of ctxSet and greater1Ctx the following applies.
  - The variable ctxSet is set equal to the variable ctxSet that has been derived during the last invocation of this subclause.
  - The variable greater1Ctx is set equal to the variable greater1Ctx that has been derived during the last invocation of this subclause.
  - When greater1Ctx is equal to 1, the variable lastGreater1Flag is set equal to the syntax element coeff_abs_level_greater1_flag that has been used during the last invocation of this subclause as follows.
    - If lastGreater1Flag is equal to 1, greater1Ctx is set equal to 0.
    - Otherwise (lastGreater1Flag is equal to 0), greater1Ctx is incremented by 1.

The context index increment ctxIdxInc is derived using the current context set ctxSet and the current context greater1Ctx as follows.

ctxIdxInc = ( ctxSet * 5 ) + min( 4, greater1Ctx ) (9-70)

Deleted: 25

When cIdx is greater than 0, ctxIdxInc is modified as follows.

ctxIdxInc = ctxIdxInc + 30 (9-71)

Deleted: 26

#### 9.3.3.1.1.6 Derivation process of ctxIdxInc for the syntax element coeff_abs_level_greater2_flag

Inputs to this process are the color component index cIdx, the transform block size log2TrafoSize, the 4x4 subset index i and the current coefficient scan index n within the current subset.

Output of this process is ctxIdxInc.

The variable ctxSet specifies the current context set and for its derivation the following applies.

- If n is equal to 0 or all previous syntex elements coeff_abs_level_greater1_flag[ pos ] with pos less than n are derived to be equal to 0 instead of beeing explicitly parsed, the following applies.
  - If the current transform block size log2TrafoSize is equal to 2, ctxSet is derived as follows.

    ctxSet = 0 (9-72)

    Deleted: 27
  - Otherwise if the current 4x4 subset is the first one in scanning order (i is equal to 0), ctxSet is derived as follows.

    ctxSet = 5 (9-73)

    Deleted: 28
  - Otherwise (i is not equal to 0), the variable lastGreater2Ctx is set equal to the variable greater2Ctx that has been derived during the last invocation of this subclause for the subset i – 1 and ctxSet is derived as follows.

    ctxSet = ( lastGreater2Ctx >> 2 ) + 1 (9-74)

    Deleted: 29
  - The variable greater2Ctx is set equal to 1.

- Otherwise (coeff_abs_level_greater1_flag[ n ] is not the first to be parsed within the current subset i), for the derivation of ctxSet and greater2Ctx the following applies.
  - The variable ctxSet is set equal to the variable ctxSet that has been derived during the last invocation of this subclause.
  - The variable greater2Ctx is set equal to the variable greater2Ctx that has been derived during the last invocation of this subclause incremented by 1.

The context index increment ctxIdxInc is derived using the current context set ctxSet and the current context greater2Ctx as follows.

ctxIdxInc = ( ctxSet * 5 ) + min( 4, greater2Ctx ) (9-75)

Deleted: 30

When cIdx is greater than 0, ctxIdxInc is modified as follows.

ctxIdxInc = ctxIdxInc + 30 (9-76)

Deleted: 31

### 9.3.3.2 Arithmetic decoding process

Inputs to this process are the bypassFlag, ctxIdxTable, and 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 9-1 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxIdxTable and the ctxIdx is passed to the arithmetic decoding process DecodeBin(ctxIdxTable, ctxIdx), which is specified as follows.

Deleted: Figure 9-1

– 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, ctxIdxTable is equal 0 and ctxIdx is equal to 0, DecodeTerminate( ) as specified in subclause 9.3.3.2.4 is invoked.

– Otherwise (bypassFlag is equal to 0, ctxIdxTable is not equal to 0 and ctxIdx is not equal to 0), DecodeDecision( ) as specified in subclause 9.3.3.2.1 is applied.

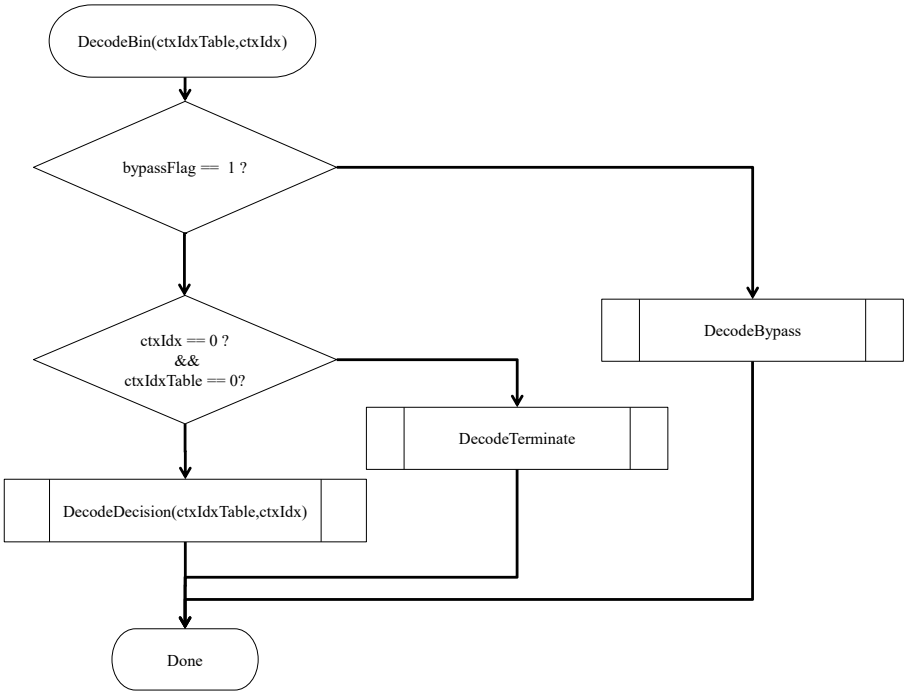



**Figure 9-1 – Overview of the arithmetic decoding process for a single bin (informative)**

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 ctxIdxTable, ctxIdx, codIRange, and codIOffset.

Outputs of this process are the decoded value binVal, and the updated variables codIRange and codIOffset.

Figure 9-2 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 ) & 3 (9-77)

   – Given qCodIRangeIdx and pStateIdx associated with ctxIdxTable and ctxIdx, the value of the variable rangeTabLPS as specified in Table 9-52 is assigned to codIRangeLPS:

     codIRangeLPS = rangeTabLPS[ pStateIdx ][ qCodIRangeIdx ] (9-78)

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 ctxIdxTable and 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 specified by the following pseudo-code:

```
if( binVal == valMPS )
    pStateIdx = transIdxMPS( pStateIdx )
else {                                                    (9-79)
    if( pStateIdx == 0 )
        valMPS = 1 – valMPS
    pStateIdx = transIdxLPS( pStateIdx )
}
```

Table 9-53 specifies the transition rules transIdxMPS( ) and transIdxLPS( ) after decoding the value of valMPS and 1 – valMPS, respectively.

Deleted: Figure 9-2
Deleted: 29
Deleted: Table 9-40
Deleted: 30
Deleted: 31
Deleted: Table 9-41

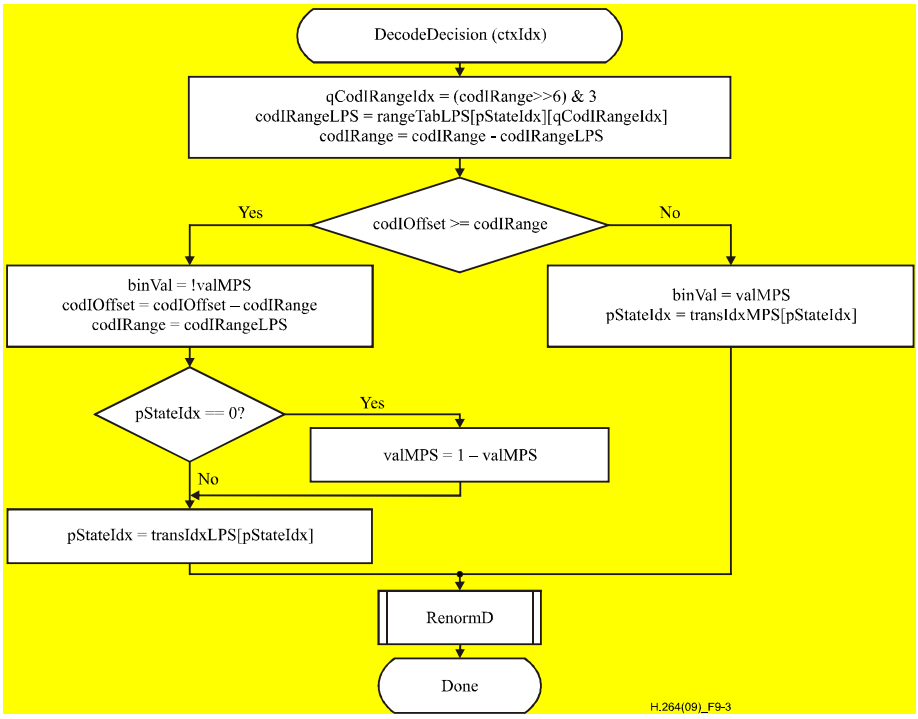



**Figure 9-2 – Flowchart for decoding a decision**

**Table 9-52 – 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 |

Deleted: 9-52

Deleted: 9-52

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **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 9-53 – State transition table**

Deleted: 9-53

Deleted: 9-53

| **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 9-3. The current value of codIRange is first compared to 256 and further steps are specified as follows.

– If codIRange is greater than or equal to 256, no renormalization is needed and the RenormD process is finished;

– Otherwise (codIRange is less than 256), 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 result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

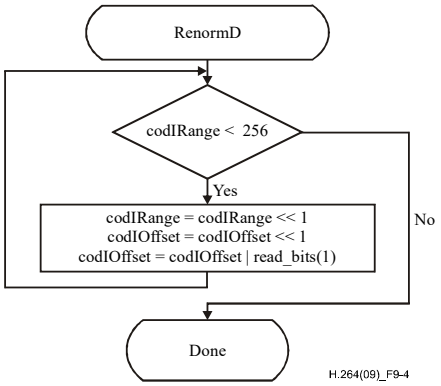



**Figure 9-3 – Flowchart of renormalization**

#### 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 9-4 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 result in a value of codIOffset being greater than or equal to codIRange upon completion of this process.

Deleted: Figure 9-3

Deleted: Figure 9-4

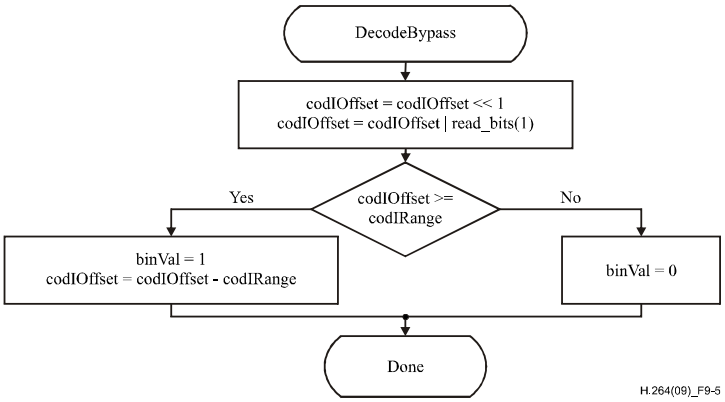



**Figure 9-4 – Flowchart of bypass decoding process**

#### 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 pcm_flag corresponding to ctxIdxTable equal to 0 and ctxIdx equal to 0. Figure 9-5 shows the flowchart of the corresponding decoding process, which is specified as follows.

**Deleted:** Figure 9-5

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(ctxIdxTable, ctxIdx) with ctxIdxTable = 0 and ctxIdx = 0. In the case where the decoded value is equal to 1, seven more bits would be read by DecodeDecision(ctxIdxTable, ctxIdx) and a decoding process would have to adjust its bitstream pointer accordingly to properly decode following syntax elements.

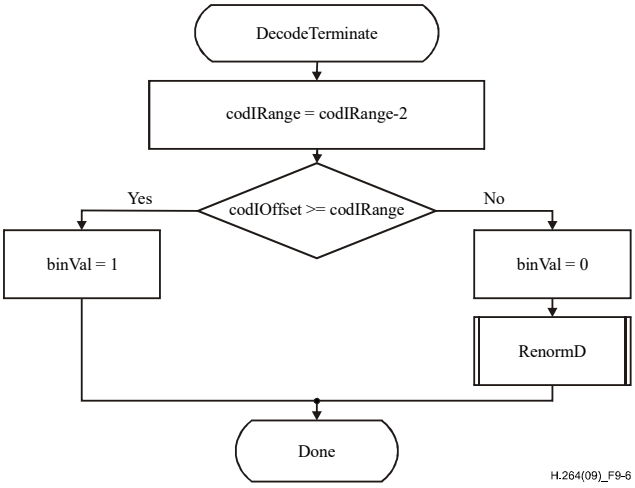



**Figure 9-5 – Flowchart of decoding a decision before termination**

### 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 coding unit of type I_PCM.

Deleted: macroblock

Outputs of this process are the values codILow, codIRange, firstBitFlag, bitsOutstanding, and BinCountsInNALunits of the arithmetic encoding engine.

In the initialisation procedure of the encoder, codILow is set equal to 0, and codIRange is set equal to 510. Furthermore, firstBitFlag is set equal to 1 and the counter bitsOutstanding is set equal to 0.

Depending on whether the current slice is the first slice of a coded picture, the following applies.

– If the current slice is the first slice of a coded picture, the counter BinCountsInNALunits is set equal to 0.

– Otherwise (the current slice is not the first slice of a coded picture), the counter BinCountsInNALunits is not modified. The value of BinCountsInNALunits is the result of encoding all the slices of a coded picture that precede the current slice in decoding order. After initialising for the first slice of a coded picture as specified in this subclause, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

NOTE – The minimum register precision required for storing the values of the variables codILow and codIRange after invocation of any of the arithmetic encoding processes specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5 is 10 bits and 9 bits, respectively. The encoding process for a binary decision (EncodeDecision) as specified in subclause 9.3.4.2 and the encoding process for a binary decision before termination (EncodeTerminate) as specified in subclause 9.3.4.5 require a minimum register precision of 10 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The bypass encoding process for binary decisions (EncodeBypass) as specified in subclause 9.3.4.4 requires a minimum register precision of 11 bits for the variable codILow and a minimum register precision of 9 bits for the variable codIRange. The precision required for the

counters bitsOutstanding and BinCountsInNALunits 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 and maxBinCountInPic denotes the maximum total number of binary decisions to encode a picture, the minimum register precision required for the variables bitsOutstanding and BinCountsInNALunits is given by Ceil( Log2( maxBinCountInSlice + 1 ) ) and Ceil( Log2( maxBinCountInPic + 1 ) ), respectively.

#### 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 BinCountsInNALunits.

Outputs of this process are the variables codIRange, codILow, and BinCountsInNALunits.

Figure 9-6 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 9-77. 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 9-52, 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 BinCountsInNALunits is incremented by 1.

Deleted: Figure 9-6

Deleted: 9-6

Deleted: Table 9-8

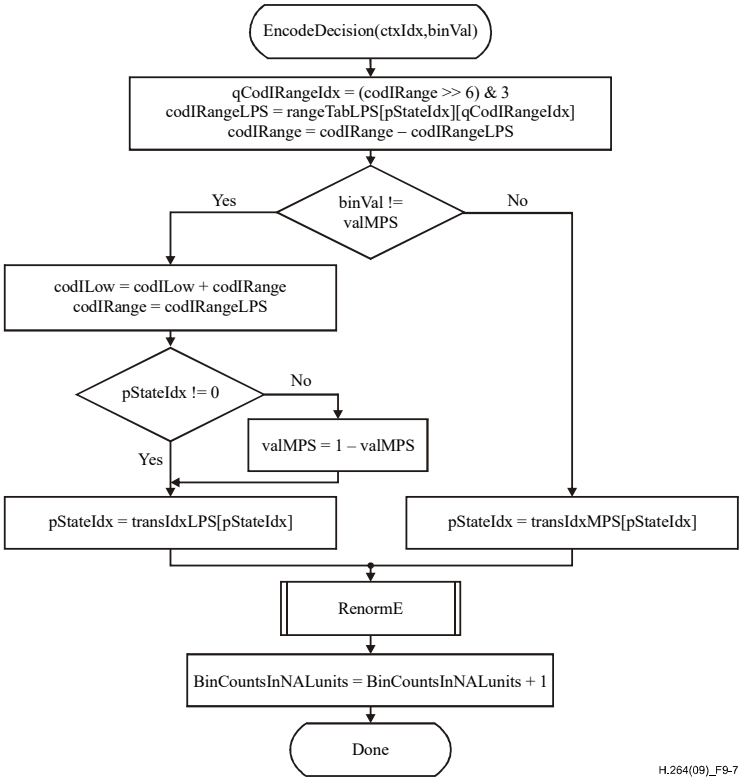



**Figure 9-6 – Flowchart for encoding a decision**

#### 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 9-7.

Deleted: Figure 9-7

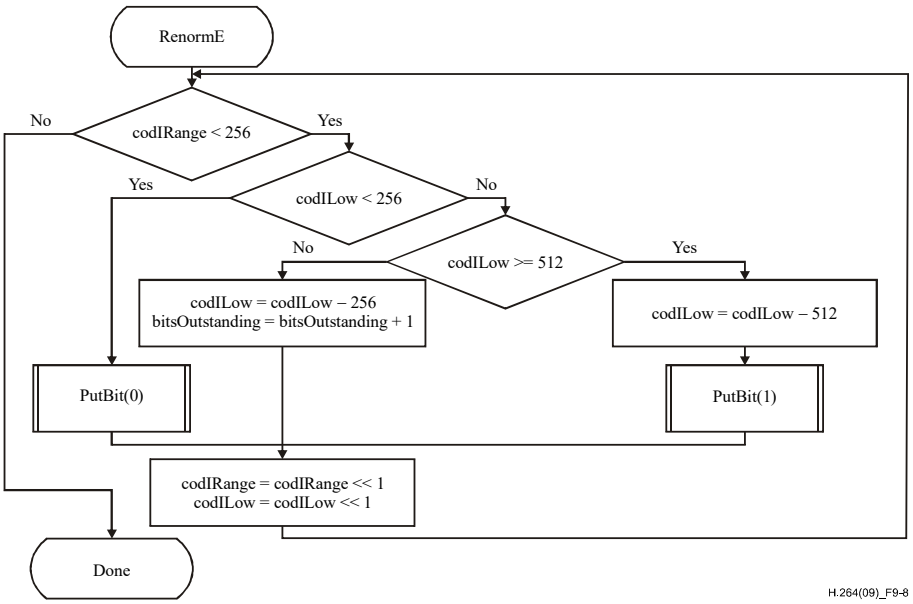



**Figure 9-7 – Flowchart of renormalization in the encoder**

The PutBit( ) procedure described in Figure 9-8 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.

Deleted: Figure 9-8

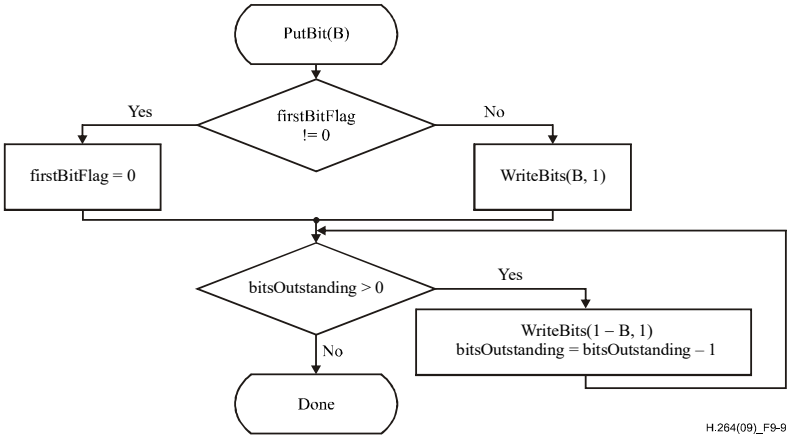



**Figure 9-8 – Flowchart of PutBit(B)**

#### 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 BinCountsInNALunits.

Output of this process is a bit written to the RBSP and the updated variables codILow, bitsOutstanding, and BinCountsInNALunits.

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 9-9.

**Deleted:** Figure 9-9

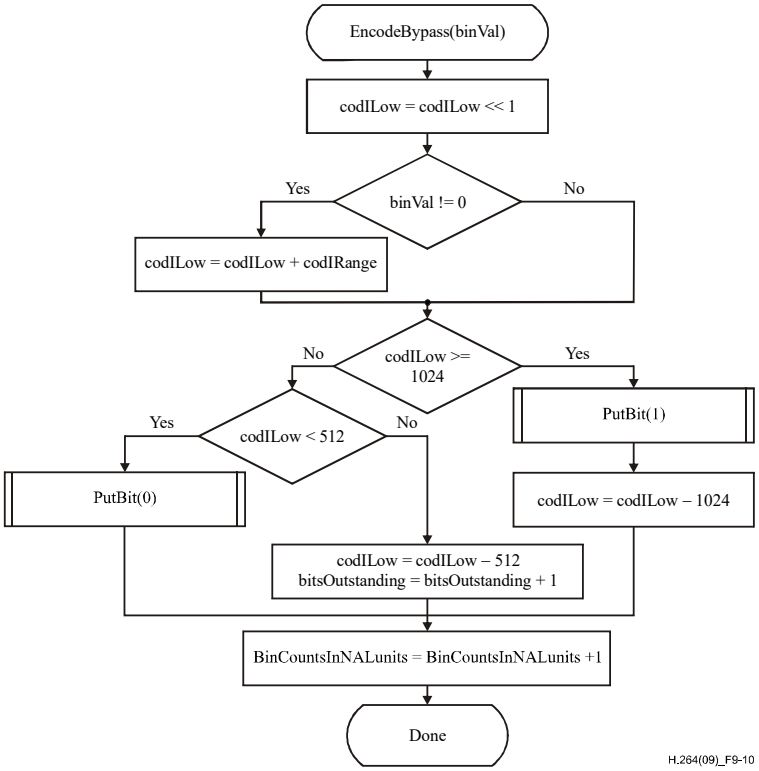



**Figure 9-9 – Flowchart of encoding bypass**

#### 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 BinCountsInNALunits.

Outputs of this process are zero or more bits written to the RBSP and the updated variables codILow, codIRange, bitsOutstanding, and BinCountsInNALunits.

This encoding routine shown in Figure 9-10 applies to encoding of the end_of_slice_flag and pcm_flag indicating the I_PCM mode both associated with ctxIdx equal to 276.

**Deleted:** Figure 9-10

**Deleted:** of the bin

**Deleted:** mb_type

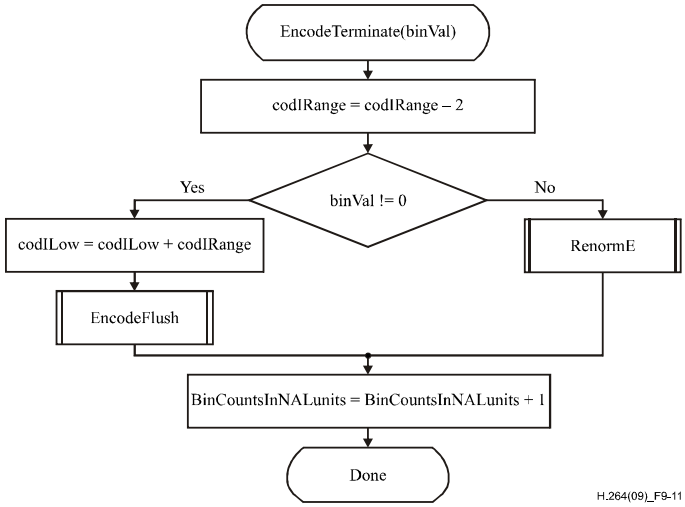



**Figure 9-10 – Flowchart of encoding a decision before termination**

When the value of binVal to encode is equal to 1, CABAC encoding is terminated and the flushing procedure shown in Figure 9-11 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.

**Deleted:** Figure 9-11

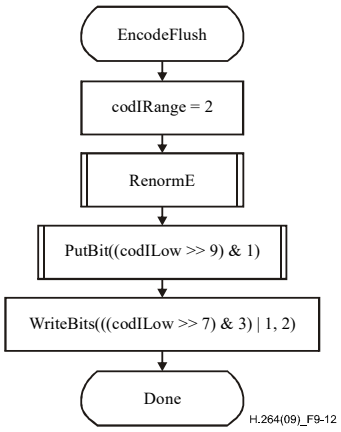



**Figure 9-11 – Flowchart of flushing at termination**

#### 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.

NOTE – The value of BinCountsInNALunits is the result of encoding all slices of a coded picture. After initialising for the first slice of a coded picture as specified in subclause 9.3.4.1, BinCountsInNALunits is incremented as specified in subclauses 9.3.4.2, 9.3.4.4, and 9.3.4.5.

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.

# Annex A

## Profiles and levels

(This annex forms an integral part of this Recommendation | International Standard)

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.

The profiles that are specified in subclause A.2 are also referred to as the profiles specified in Annex A.

### 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

All constraints for picture parameter sets that are specified are constraints for picture parameter sets that are activated in the bitstream. All constraints for sequence parameter sets that are specified in subclause A.2.1 are constraints for sequence parameter sets that are activated in the bitstream.

#### A.2.1 ABC profile

### A.3 Levels

**Table A-9-54 – Level limits [Ed. (TW) Kept for convenience]**

| Level number | Max macroblock processing rate MaxMBPS (MB/s) | Max picture size MaxFS (MBs) | Max decoded picture buffer size MaxDpbMbs (MBs) | 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 picture samples) | Min compression ratio MinCR | Max number of motion vectors per two consecutive MBs MaxMvsPer2Mb |
|---|---|---|---|---|---|---|---|---|
| **1** | 1 485 | 99 | 396 | 64 | 175 | [−64,+63.75] | 2 | - |
| **1b** | 1 485 | 99 | 396 | 128 | 350 | [−64,+63.75] | 2 | - |
| **1.1** | 3 000 | 396 | 900 | 192 | 500 | [−128,+127.75] | 2 | - |
| **1.2** | 6 000 | 396 | 2 376 | 384 | 1 000 | [−128,+127.75] | 2 | - |
| **1.3** | 11 880 | 396 | 2 376 | 768 | 2 000 | [−128,+127.75] | 2 | - |
| **2** | 11 880 | 396 | 2 376 | 2 000 | 2 000 | [−128,+127.75] | 2 | - |
| **2.1** | 19 800 | 792 | 4 752 | 4 000 | 4 000 | [−256,+255.75] | 2 | - |
| **2.2** | 20 250 | 1 620 | 8 100 | 4 000 | 4 000 | [−256,+255.75] | 2 | - |
| **3** | 40 500 | 1 620 | 8 100 | 10 000 | 10 000 | [−256,+255.75] | 2 | 32 |
| **3.1** | 108 000 | 3 600 | 18 000 | 14 000 | 14 000 | [−512,+511.75] | 4 | 16 |
| **3.2** | 216 000 | 5 120 | 20 480 | 20 000 | 20 000 | [−512,+511.75] | 4 | 16 |
| **4** | 245 760 | 8 192 | 32 768 | 20 000 | 25 000 | [−512,+511.75] | 4 | 16 |
| **4.1** | 245 760 | 8 192 | 32 768 | 50 000 | 62 500 | [−512,+511.75] | 2 | 16 |
| **4.2** | 522 240 | 8 704 | 34 816 | 50 000 | 62 500 | [−512,+511.75] | 2 | 16 |
| **5** | 589 824 | 22 080 | 110 400 | 135 000 | 135 000 | [−512,+511.75] | 2 | 16 |
| **5.1** | 983 040 | 36 864 | 184 320 | 240 000 | 240 000 | [−512,+511.75] | 2 | 16 |

Levels with non-integer level numbers in Table A-9-54 are referred to as "intermediate levels".

NOTE 4 – All levels have the same status, but some applications may choose to use only the integer-numbered levels.

Deleted: 9-54

Deleted: 9-54

Deleted: Table A-9-8

# Annex 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 &&<br>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( ) &&<br>next_bits( 24 ) != 0x000001 &&<br>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).

Deleted: 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 subsequent byte-aligned three-byte sequence equal to 0x000000,
   – A subsequent byte-aligned three-byte sequence equal to 0x000001,
   – 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.

# Annex 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-9-12 shows the types of bitstream conformance points checked by the HRD.

Deleted: Figure C-9-12

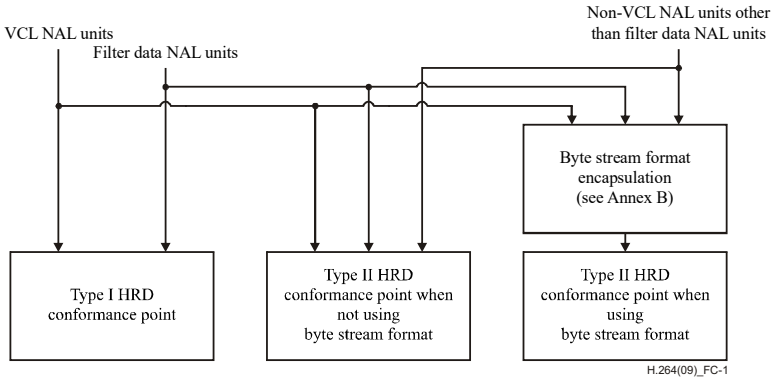



**Figure C-9-12 – Structure of byte streams and NAL unit streams for HRD conformance checks**

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 6.6, Annexes D and E.

Deleted: 7

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-9-13.

Deleted: Figure C-9-13

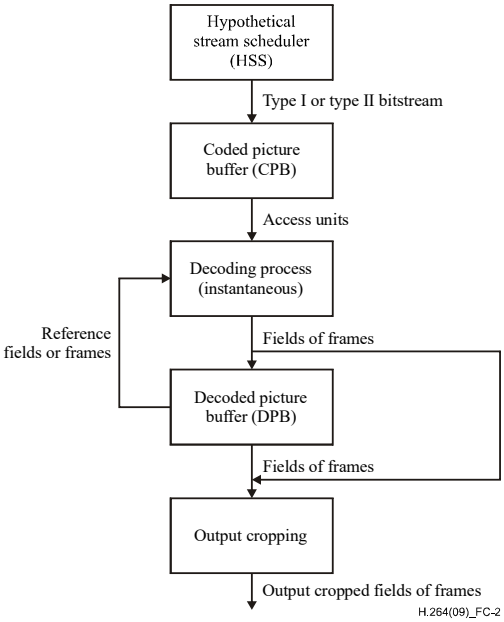



**Figure C-9-13 – HRD buffer model**

## 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-9-12.

**Deleted:** Figure C-9-12

### C.1.1 Timing of bitstream arrival

### C.1.2 Timing of coded picture removal

## C.2 Operation of the decoded picture buffer (DPB)

## C.3 Bitstream conformance

## C.4 Decoder conformance

# Annex 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 0 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 0 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

[Ed. (TW): insert table]

### D.2 SEI payload semantics

# Annex 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 0 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

[Ed. (TW): insert table]

## E.2 VUI semantics

**Page 39: [1] Deleted JCTVC-E483 6/21/11 8:28:00 PM**

**Page 40: [2] Deleted JCTVC-D070-D503 6/21/11 11:11:00 AM**

**Page 43: [3] Deleted wjhan-Samsung 6/22/11 9:29:00 PM**

**Page 44: [4] Deleted JCTVC-E483 6/21/11 8:46:00 PM**

**Page 48: [5] Deleted JCTVC-E141-E381 6/24/11 2:30:00 PM**

**Page 76: [6] Deleted JCTVC-E321 6/27/11 2:58:00 PM**

**Page 76: [7] Deleted JCTVC-E321 6/27/11 2:59:00 PM**

**Page 76: [8] Formatted JCTVC-E321 6/24/11 5:52:00 PM**

Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27"

**Page 76: [9] Formatted JCTVC-E321 6/24/11 5:52:00 PM**

Indent: Left: 0.32", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27", Tab stops: 0.59", List tab + Not at 0.27"

**Page 76: [10] Formatted JCTVC-E321 6/24/11 5:57:00 PM**

Font: English (UK)

**Page 76: [11] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [12] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [13] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [14] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [15] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [16] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

**Page 76: [17] Formatted JCTVC-E321 6/24/11 5:59:00 PM**

English (UK)

Page 76: [18] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [19] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [20] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [21] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [22] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [23] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [24] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [25] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [26] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [27] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [28] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [29] Formatted JCTVC-E321 6/24/11 5:59:00 PM

English (UK)

Page 76: [30] Formatted JCTVC-E321 6/24/11 6:00:00 PM

English (UK)

Page 76: [31] Formatted JCTVC-E321 6/24/11 6:00:00 PM

Tab stops: 0.79", Left + Not at 0.83"

**Page 76: [32] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [33] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [34] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [35] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [36] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

Font: (Asian) Korean

**Page 76: [37] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

Indent: Left: 0.52", Bulleted + Level: 1 + Aligned at: 0" + Tab after: 0.27" + Indent at: 0.27", Tab stops: 0.79", Left + Not at 0.27" + 0.55" + 0.83"

**Page 76: [38] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [39] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [40] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [41] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [42] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [43] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [44] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [45] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [46] Formatted JCTVC-E321 6/24/11 6:00:00 PM**

English (UK)

**Page 76: [47] Formatted wjhan-Samsung 6/27/11 11:26:00 AM**

Tab stops: 0.79", Left + Not at 0.55" + 0.83"

**Page 81: [48] Deleted JCTVC-E321 6/27/11 5:43:00 PM**

**Page 98: [49] Deleted JCTVC-E321 6/27/11 2:03:00 PM**

**Page 101: [50] Deleted cliff reader 1/5/26 6:11:00 PM**

**Page 101: [50] Deleted cliff reader 1/5/26 6:11:00 PM**

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [51] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM

English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [52] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [53] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [54] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [54] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [55] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [56] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [57] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM
English (UK)

**Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [58] Formatted JCTVC-E069 6/27/11 1:32:00 PM**

English (UK)

**Page 101: [59] Deleted JCTVC-E488 6/27/11 11:45:00 AM**

**Page 101: [60] Deleted wjhan-Samsung 6/27/11 11:33:00 AM**

**Page 101: [61] Deleted cliff reader 1/5/26 6:11:00 PM**

**Page 101: [61] Deleted cliff reader 1/5/26 6:11:00 PM**

**Page 104: [62] Formatted wjhan-Samsung 6/27/11 11:33:00 AM**

Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" +

**Page 104: [63] Deleted JCTVC-E488 6/27/11 11:49:00 AM**

**Page 104: [64] Deleted JCTVC-E488 6/27/11 11:49:00 AM**

**Page 104: [65] Formatted JCTVC-E266 6/23/11 11:19:00 AM**

Equation, Indent: Left: 0.59", Tab stops: 0.59", Left + 0.79", Left + 0.98", Left

**Page 104: [66] Formatted wjhan-Samsung 6/27/11 11:33:00 AM**

Indent: Left: 0.21", Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0" + Tab after: 0.28" + Indent at: 0.28", Tab stops: 0.5", Left + 0.75", Left + 1", Left + 1.18", Left + Not at 0.55" + 0.83" +

**Page 104: [67] Formatted JCTVC-E266 6/23/11 10:24:00 PM**

Normal, Indent: Left: 0", Tab stops: Not at 0.59" + 0.79" + 0.98"

**Page 131: [68] Deleted JCTVC-E125 6/22/11 5:25:00 PM**

**Page 131: [68] Deleted JCTVC-E125 6/22/11 5:25:00 PM**

**Page 131: [68] Deleted JCTVC-E125 6/22/11 5:25:00 PM**

Page 131: [69] Formatted JCTVC-E125 6/22/11 5:30:00 PM

Subscript

Page 131: [69] Formatted JCTVC-E125 6/22/11 5:30:00 PM

Subscript

Page 131: [69] Formatted JCTVC-E125 6/22/11 5:30:00 PM

Subscript

Page 131: [69] Formatted JCTVC-E125 6/22/11 5:30:00 PM

Subscript

Page 131: [70] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [70] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [70] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [70] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [71] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [71] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [71] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [72] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [72] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [72] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [73] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [73] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [73] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [73] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [74] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [74] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [75] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [75] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [75] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [75] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [76] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [76] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [76] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [76] Formatted JCTVC-E125 6/22/11 5:38:00 PM

Subscript

Page 131: [77] Deleted JCTVC-E125 6/22/11 5:25:00 PM

**Page 131: [77] Deleted** **JCTVC-E125** **6/22/11 5:25:00 PM**

**Page 131: [77] Deleted** **JCTVC-E125** **6/22/11 5:25:00 PM**

**Page 131: [78] Deleted** **JCTVC-E125** **6/22/11 5:26:00 PM**

**Page 131: [78] Deleted** **JCTVC-E125** **6/22/11 5:26:00 PM**

**Page 177: [79] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [80] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [81] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [82] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [83] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [84] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [85] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [86] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [87] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [88] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

**Page 177: [89] Formatted** **cliff reader** **1/5/26 6:13:00 PM**

Font: 8 pt

Page 177: [90] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [91] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [92] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [93] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [94] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [95] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [96] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [97] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [98] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [99] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [100] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [101] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [102] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [103] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [104] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [105] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [106] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [107] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [108] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 177: [109] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [110] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [111] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [112] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [113] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [114] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [115] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [116] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [117] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [118] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [119] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [120] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [121] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [122] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [123] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [124] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [125] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [126] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [127] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [128] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [129] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [130] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [131] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [132] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [133] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

Page 178: [134] Formatted cliff reader 1/5/26 6:13:00 PM

Font: 8 pt

**Page 178: [135] Formatted cliff reader 1/5/26 6:13:00 PM**

Font: 8 pt

**Page 178: [136] Formatted cliff reader 1/5/26 6:13:00 PM**

Font: 8 pt

**Page 178: [137] Formatted cliff reader 1/5/26 6:13:00 PM**

Font: 8 pt

# APPENDIX 2

**Joint Collaborative Team on Video Coding (JCT-VC)**
**of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**
6th Meeting: Turin, IT, 14-22 July 2011

Document: JCTVC-F158

*Title:* Resolution switching for coding efficiency and resilience

*Status:* Input Document

*Purpose:* Proposal

*Author(s) or Contact(s):* Thomas Davies Email: thdavies@cisco.com

*Source:* Cisco Systems (UK)

---

# Abstract

A method for changing the resolution of frames within a sequence without causing an IDR or new SPS to be sent is proposed. Frames may be predicted across resolutions by re-scaling reference pictures in a similar manner to H.263 Annex P. The purpose is to allow video communications to use re-scaling to adapt seamlessly to adverse network conditions. It is observed that in HEVC Intra frames are relatively more expensive than in AVC, and forcing the use of Intra frames can worsen losses and increase delays. It is reported that predicting instead of inserting an IDR frame when down-scaling gives average gains of 5.4-5.6% in Low-Delay common conditions, with 15.6-16.4% gain in Class E. When the size of intra frames is reduced to emulate realistic video conferencing, average gains of 6.9-7.4% are observed, with 20.5-21.6% gain in Class E.

# 1 Introduction

This document proposes allowing spatial resolution to vary within a video stream without triggering a new Sequence Parameter Set (SPS) and IDR frame. It is proposed instead that an encoder can signal a change of resolution and predict frames across resolutions. The method used for prediction is very close to H.263-style reference-frame resampling [1]: reference frames are simply re-scaled as needed.

The motivation for this proposal is the requirements of video communication. Here video is frequently re-scaled for a number of reasons, but especially to maximise perceptual quality in adverse or varying network conditions: video conferencing equipment often contains advanced scaling algorithms to achieve this. However, in AVC prediction across resolutions is not permitted and an IDR frame must be sent to re-initialise the stream.

Resolution is therefore often changed at times of network stress. Although IDR frames break dependencies, this is not necessarily a good thing for error resilience. The sheer size of Intra frames make them difficult to transmit in a low-delay environment. If part of an IDR frame is then lost, another IDR frame must be generated after a delay, sometimes leading to an “Intra storm” as bandwidth is consumed by repeated resynchronisation frames which cannot get through. Often it is the case that known good long-term references exist that could be used at the point of resolution change for a lower-cost recovery.

The current development of HEVC suggests that much less gain over AVC has been found for Intra frame coding than for Inter coding. This increases the disparity between Intra and Inter frames, and therefore the impact of changing resolution compared to AVC.

Although it is proposed that a stream may contain different resolutions, this is not a scalable coding proposal. However, the proposal is certainly highly compatible with future scalable extensions, and the syntax changes will provide hooks for scalable functionality.

The proposed algorithm has been implemented in HM3.1 and results are reported showing the impact of an inserted IDR frame versus cross-resolution prediction from an old reference frame.

# 2 Requirements for dynamic resolution

## *2.1 Seamless network adaption and error resilience*

Applications such as video conferencing and streaming over packets networks frequently require that the encoded stream adapt to changing network conditions, especially when bit rates are too high and data is being lost. Such applications typically have a return channel allowing the encoder to detect the errors and perform adjustments. The encoder has two main tools at its disposal: bit rate reduction and changing the resolution, either temporal or spatial. Temporal resolution changes can be effectively achieved by coding using hierarchical prediction structures. However, for best quality spatial resolution changes are needed as well as part of a well-designed encoder for video communications.

Changing spatial resolution within AVC requires an IDR frame is sent and the stream is reset. This causes significant problems. An IDR frame at reasonable quality will be much larger than an Inter picture, and will be correspondingly more complex to decode: this costs time and resource. This is a problem if the resolution change is requested by the decoder for loading reasons. It can also break low-latency buffer conditions, forcing an audio re-sync, and the end-to-end delay of the stream will increase, at least temporarily. This gives a poor user experience.

To minimize these problems, the IDR is typically sent at low quality, using a similar number of bits to a P frame, and it takes a significant time to return to full quality for the given resolution. To get low enough delay, the quality can be very low indeed and there is often a visible blurring before the image is “refocused”. In effect, the Intra frame is doing very little useful work in compression terms: it is just a method of re-starting the stream.

So there is a requirement for methods in HEVC which allow resolution to be changed, especially in challenging network conditions, with minimal impact on subjective experience.

### *2.2 Fast start*

It would be useful to have a "fast start" mode where the first frame is sent at reduced resolution and resolution is increased over the next few frames, in order to reduce delay and get to normal quality more quickly without unacceptable image blurring at the beginning.

### *2.3 Conference "compose"*

Video conferences also often have a feature whereby the person speaking is shown full-screen and other participants are shown in smaller resolution windows. To support this efficiently, often the smaller pictures are sent at lower resolution. This resolution is then increased when the participant becomes the speaker and is full-screened. Sending an intra frame at this point causes an unpleasant hiccup in the video stream. This effect can be quite noticeable and unpleasant if speakers alternate rapidly.

### *2.4 Relationship with scalable coding*

It is not proposed to add scalable coding functionality into the core standard. Likewise it is not desirable to require a scalable coding solution in order to switch resolution efficiently, since in many video conference scenarios not all participants may support scalable technology but will still need to change resolution. Instead, since resolution adaption is a core video coding technique across many applications to maximize subjective quality, there is a requirement to support dynamic changes of resolution in HEVC.

## 3 Tool description

There are a number of different ways in which changing the resolution mid-stream could be achieved. The focus in this proposal is simplicity and so the following constraints have been applied:

- the resolution may differ from the nominal resolution by a factor 0.5, applied to both dimensions at once; resolution may increase or decrease, yielding scaling ratios of 0.5 and 2.0 (but see below).
- aspect ratios, chroma formats or other parts of the video format are not changed;
- cropping areas are scaled in proportion to the picture;
- reference pictures are simply re-scaled as needed and picture prediction is as normal.

(More different resolutions could be supported, subject to limiting the number of different scalings that would need to be implemented to a manageable level.)

A Picture Resolution Index (PRI) is signaled in the PPS. This will indicate slices using this PPS will be from pictures having the resolution indicated by the index. This will set values for the width and height. This index will form part of the meta-data associated with each stored reference picture.

When a subsequent picture is decoded, reference picture lists need to be formed. If the current reference picture has index different from the reference picture, it is rescaled. Both resolutions are available for future reference – that is, if the resolution changes back, no further rescaling is needed.

If a reference frame has a different size to the current frame, temporal motion vector prediction is disabled.

It is possible that complexity could be controlled by limiting the number of scalings per frame period, to prevent several reference frames from being re-scaled at once.

## *3.1 Compatibility with SVC*

In case the PRI is never changed, the stream will correspond to a single resolution which may, in case PRI>0, be lower than the nominal resolution. An enhancement layer could therefore be added with a different PRI, with the potential for cross-layer prediction. H.263 Annex 0-style spatial scalability could be added very simply as a base-level scalability system if desired.

## *3.2 Scaling*

It is proposed to use simple, zero-phase separable down- and up-scaling filters. Note these filters are for prediction only: a decoder may also use more sophisticated scaling for display purposes.

The factor 2 down-scaling filter has zero phase and taps:

$$(-1, 9, 16, 9, -1)/32$$

Down-sampling points are at even sample positions and are co-sited. The same filter is used for luma and chroma.

2:1 up-sampling is performed by inserting zeroes at odd sample positions, and filtering with the filter

$$( -3\ 0\ 19\ 32\ 19\ 0\ -3)/32$$

Values are computed using unbiased rounding, e.g.

$$y(2n+1) = (19*(x(2n)+x(2n+2)) – 3(x(2n-2)+x(2n+4))+16)>>5$$

Edge extension is used at image boundaries.

Combined up- and down-sampling will not change phase or the position of chroma sampling points.

# 4 Modifications to bitstream syntax and semantics

Various modifications to the current Working Draft (WD) [3] are needed.

## *4.1 Sequence parameter set (SPS)*

The SPS is modified as follows:

| seq_parameter_set_rbsp( ) { | **Descriptor** |
|---|---|
| **profile_idc** | u(8) |
| **reserved_zero_8bits** /* equal to 0 */ | u(8) |
| **level_idc** | u(8) |
| **seq_parameter_set_id** | ue(v) |
| **max_temporal_layers_minus1** | u(3) |
| **sequence_pic_width_in_luma_samples** | u(16) |
| **sequence_pic_height_in_luma_samples** | u(16) |
| **bit_depth_luma_minus8** | ue(v) |
| **bit_depth_chroma_minus8** | ue(v) |
| **pcm_bit_depth_luma_minus1** | u(4) |
| **pcm_bit_depth_chroma_minus1** | u(4) |
| **log2_max_frame_num_minus4** | ue(v) |
| **pic_order_cnt_type** | ue(v) |
| if( pic_order_cnt_type == 0 ) | |
| **log2_max_pic_order_cnt_lsb_minus4** | ue(v) |
| else if( pic_order_cnt_type == 1 ) { | |
| **delta_pic_order_always_zero_flag** | u(1) |
| **offset_for_non_ref_pic** | se(v) |
| **num_ref_frames_in_pic_order_cnt_cycle** | ue(v) |
| for( i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++ ) | |
| **offset_for_ref_frame[ i ]** | se(v) |
| } | |
| **max_num_ref_frames** | ue(v) |
| **gaps_in_frame_num_value_allowed_flag** | u(1) |
| **log2_min_coding_block_size_minus3** | ue(v) |
| **log2_diff_max_min_coding_block_size** | ue(v) |
| **log2_min_transform_block_size_minus2** | ue(v) |
| **log2_diff_max_min_transform_block_size** | ue(v) |
| **log2_min_pcm_coding_block_size_minus3** | ue(v) |
| **max_transform_hierarchy_depth_inter** | ue(v) |
| **max_transform_hierarchy_depth_intra** | ue(v) |
| **chroma_pred_from_luma_enabled_flag** | u(1) |
| **loop_filter_across_slice_flag** | u(1) |
| **sample_adaptive_offset_enabled_flag** | u(1) |
| **adaptive_loop_filter_enabled_flag** | u(1) |
| **pcm_loop_filter_disable_flag** | u(1) |
| **cu_qp_delta_enabled_flag** | u(1) |
| **temporal_id_nesting_flag** | u(1) |
| rbsp_trailing_bits( ) | |
| } | |

## *4.2 Picture parameter set (PPS)*

The PPS is modified as follows:

| pic_parameter_set_rbsp( ) { | **Descriptor** |
|---|---|
| **pic_parameter_set_id** | ue(v) |
| **seq_parameter_set_id** | ue(v) |
| **picture_resolution_idx** | ue(v) |
| if (picture_resolution_idx = = 0){ | |
| pic_width_in_luma_samples = sequence_pic_width_in_luma_samples | |
| pic_height_in_luma_samples = sequence_pic_height_in_luma_samples | |
| } else if (picture_resolution_idx = = 1){ | |
| pic_width_in_luma_samples = sequence_pic_width_in_luma_samples+1)/2 | |
| pic_height_in_luma_samples = (sequence_pic_height_in_luma_samples+1)/2 | |
| } | |
| **entropy_coding_mode_flag** | u(1) |
| **num_temporal_layer_switching_point_flags** | ue(v) |
| for( i = 0; i < num_temporal_layer_switching_point_flags; i++ ) | |
| **temporal_layer_switching_point_flag**[ i ] | u(1) |
| **num_ref_idx_l0_default_active_minus1** | ue(v) |
| **num_ref_idx_l1_default_active_minus1** | ue(v) |
| **pic_init_qp_minus26** /* relative to 26 */ | se(v) |
| **constrained_intra_pred_flag** | u(1) |
| **slice_granularity** | u(2) |
| **shared_pps_info_enabled_flag** | u(1) |
| if( shared_pps_info_enabled_flag ) | |
| if( adaptive_loop_filter_enabled_flag ) | |
| alf_param( ) | |
| if( cu_qp_delta_enabled_flag ) | |
| **max_cu_qp_delta_depth** | u(4) |
| rbsp_trailing_bits( ) | |
| } | |

### 4.2.1 PPS semantics

picture_resolution_idx shall have a value between 0 and 1.

## *4.3 Slice Header*

There are no changes to the slice header syntax

## *4.4 Prediction*

The value of picture_resolution_idx present in the PPS shall be assigned to each frame whose slices reference that PPS, and made available via a function PicResolutionIdx. If a reference frame in RefPicList0 or RefPicList1 or RefPicListComb has a different resolution to the current frame, it is first rescaled to have resolution corresponding to that of the current picture.

### 4.4.1 Prediction semantics

#### 4.4.1.1 Co-located predictors

Section 8.4.2.1.6 "Derivation process for temporal luma motion vector prediction" of the WD [3] is modified as follows:

...

Outputs of this process are

– the motion vector prediction mvLXCol,

– the availability flag availableFlagLXCol.

The function RefPicOrderCnt( pic, refidx, LX ) is specified by the value of PicOrderCnt of the picture that is the reference picture RefPicListX[ refidx ] of pic with X being 0 or 1. PicOrderCnt of the reference picture shall be maintained until the picture is marked as "non-exisiting."

The function RefPicResolutionIdx ( pic, refidx, LX ) is specified by the value of PicResolutionIdx() of the picture that is the reference picture RefPicListX[ refidx ] of pic with X being 0 or 1. If RefPicResolutionIdx ( pic, refidx, LX ) is not equal to PicResolutionIdx( pic ) then availableFlagLXCol shall be false.

...

#### 4.4.1.2 Scaling

Section 8.4.2.2.1 "Reference picture selection process" in the WD [3] will be modified to require scaling when RefPicResolutionIdx is not equal to PicResolutionIdx.

# 5 Simulations

HM3.1 was modified to incorporate the changes to the PPS and SPS, and the scaling functions. In single-resolution mode, the code gave almost identical results to the HM3.1 anchors. Differences were due to the different PPS and SPS (the HM3.1 SPS does not include seq_parameter_set_id), and to padding the input pictures to a multiple of 16 by default to support the lower resolution mode.

Simulations were performed using LDB_HE, LDB_LC, LDP_HE and LDP_LC configurations [2]. A resolution change scenario was simulated at frame 20. It was assumed that frames 16-19 were lost or too badly damaged to be used as references, and hence were marked as non-reference. From frame 20 pictures were coded at 1/4 resolution. All pictures were padded to a multiple of 16 to support an 8x8 CU size at the lowest resolution.

Six configurations were simulated:

**Configuration 1 :**

– Flags: --ResSwitchFrameNum=20 --ResSwitchType=3

– IDR frame at frame 0 and frame 20

– pictures 16-19 inclusive unavailable for reference

- pictures with POC>=20 coded at 1/4 resolution:
- prediction structure as per common conditions

**Configuration 2:**

- Flags: --ResSwitchFrameNum=20 --ResSwitchType=1
- IDR frame at frame 0 and predicted frame at frame 20
- pictures 16-19 inclusive unavailable for reference
- pictures with POC>=20 coded at 1/4 resolution
- prediction structure as per common conditions

**Configuration 3:**

- Flags: --ResSwitchFrameNum=20 --ResSwitchType=3 --IntraQPOffset=6
- IDR frame at frame 0 and frame 20 with QP=nominal QP+6
- pictures 16-19 inclusive unavailable for reference
- pictures with POC>=20 coded at 1/4 resolution
- --RateGOPSize=2

**Configuration 4:**

- Flags: --ResSwitchFrameNum=20 --ResSwitchType=1 --IntraQPOffset=6
- IDR frame at frame 0 with QP=nominal QP+6 and predicted frame at frame 20
- pictures 16-19 inclusive unavailable for reference
- pictures with POC>=20 coded at 1/4 resolution
- --RateGOPSize=2

**Configuration 5:**

- Flags: --ResSwitchFrameNum=20 --ResSwitchType=3 --IntraQPOffset=12
- As Configuration 3 but with Intra QP=nominal QP+12

**Configuration 6**

- Flags: --ResSwitchFrameNum=20 --ResSwitchType=1 --IntraQPOffset=12
- As Configuration 4 but with Intra QP=nominal QP+12

Configurations 3-6 represent more realistic video conference configurations. Configurations 5 and 6 perhaps represent the most acceptable Intra frame size in delay terms. Configuration 1 was compared against configuration 2, configuration 3 against configuration 4 and configuration 5 against configuration 6.

Simulations were run on a homogenous cluster using 64 bit Intel Xeon X7560 8-core processors at 2.26GHz, running 64-bit Debian Linux. A single process was run per physical core (no hyper-threading).

## *5.1 Common conditions comparison (configs 1 and 2)*

Summary results are as follows:

| | Low delay B HE | | | Low delay B LC | | |
|---|---|---|---|---|---|---|
| | Y | U | V | Y | U | V |
| Class A | | | | | | |
| Class B | -5.2 | -10.2 | -11.2 | -5.1 | -9.9 | -10.5 |
| Class C | -2.2 | -4.2 | -4.1 | -2.1 | -4.3 | -3.7 |
| Class D | -1.7 | -5.7 | -4.3 | -1.6 | -2.0 | -2.8 |
| Class E | -16.4 | -27.7 | -33.3 | -15.6 | -31.1 | -36.6 |
| **Overall** | **-5.7** | **-10.8** | **-11.8** | **-5.4** | **-10.5** | **-11.8** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 92% | | | 92% | | |

| | Low delay P HE | | | Low delay P LC | | |
|---|---|---|---|---|---|---|
| | Y | U | V | Y | U | V |
| Class A | | | | | | |
| Class B | -4.9 | -9.8 | -11.4 | -5.1 | -10.2 | -11.3 |
| Class C | -2.1 | -4.4 | -3.9 | -2.2 | -5.1 | -4.6 |
| Class D | -1.6 | -7.2 | -5.4 | -1.7 | -3.0 | -3.1 |
| Class E | -16.5 | -29.0 | -35.3 | -16.3 | -32.8 | -38.2 |
| **Overall** | **-5.6** | **-11.4** | **-12.5** | **-5.6** | **-11.4** | **-12.6** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 92% | | | 93% | | |

For class E, the configuration 1 lower resolution IDR frame size as a multiple of mean frame size is:

| | Vidyo1 | Vidyo3 | Vidyo4 | Mean (frame periods) | Milliseconds |
|---|---|---|---|---|---|
| QP=22 | 14.9 | 13.1 | 13.9 | 14.0 | 248 |
| QP=37 | 26.9 | 24.7 | 25.1 | 25.7 | 428 |

## *5.2 Reduced intra frame quality comparison*

### 5.2.1 Intra QP + 6 (configs 3 and 4)

Summary results are as follows:

| | Low delay B HE | | | Low delay B LC | | |
|---|---|---|---|---|---|---|
| | Y | U | V | Y | U | V |
| Class A | | | | | | |
| Class B | -6.4 | -11.9 | -12.5 | -6.3 | -11.5 | -11.7 |
| Class C | -2.7 | -5.6 | -5.0 | -2.5 | -5.6 | -5.1 |
| Class D | -2.1 | -6.6 | -5.4 | -1.9 | -2.7 | -3.9 |
| Class E | -21.3 | -30.0 | -36.6 | -20.5 | -31.9 | -38.6 |
| **Overall** | **-7.2** | **-12.4** | **-13.4** | **-6.9** | **-11.7** | **-13.1** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 98% | | | 95% | | |

| | Low delay P HE | Low delay P LC |
|---|---|---|

| | Y | U | V | Y | U | V |
|---|---|---|---|---|---|---|
| Class A | | | | | | |
| Class B | -6.0 | -11.9 | -13.0 | -6.2 | -11.4 | -12.1 |
| Class C | -2.6 | -5.4 | -4.9 | -2.5 | -5.8 | -4.9 |
| Class D | -2.0 | -5.9 | -5.6 | -1.9 | -2.8 | -3.9 |
| Class E | -21.4 | -32.5 | -38.5 | -21.5 | -32.7 | -40.7 |
| **Overall** | **-7.0** | **-12.6** | **-13.9** | **-7.1** | **-11.8** | **-13.6** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 93% | | | 96% | | |

For class E, the configuration 1 lower resolution IDR frame sizes as a multiple of mean frame size are:

| | Vidyo1 | Vidyo3 | Vidyo4 | Mean (frame periods) | Milliseconds |
|---|---|---|---|---|---|
| QP=22 | 7.6 | 6.8 | 6.6 | 7.0 | 117 |
| QP=37 | 11.7 | 8.5 | 11.0 | 10.4 | 173 |

### 5.2.2 Intra QP + 12 (configs 5 and 6)

Summary results are as follows:

| | Low delay B HE | | | Low delay B LC | | |
|---|---|---|---|---|---|---|
| | Y | U | V | Y | U | V |
| Class A | | | | | | |
| Class B | -6.6 | -12.3 | -12.8 | -6.7 | -12.0 | -12.0 |
| Class C | -2.9 | -6.0 | -5.6 | -2.8 | -5.8 | -5.2 |
| Class D | -2.2 | -7.0 | -6.3 | -2.3 | -4.4 | -5.0 |
| Class E | -21.5 | -30.5 | -37.3 | -20.9 | -33.6 | -39.9 |
| **Overall** | **-7.4** | **-12.8** | **-14.0** | **-7.3** | **-12.6** | **-13.8** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 98% | | | 98% | | |

| | Low delay P HE | | | Low delay P LC | | |
|---|---|---|---|---|---|---|
| | Y | U | V | Y | U | V |
| Class A | | | | | | |
| Class B | -6.4 | -12.3 | -13.0 | -6.6 | -12.2 | -12.4 |
| Class C | -2.7 | -5.7 | -5.4 | -2.8 | -6.4 | -5.4 |
| Class D | -2.3 | -5.8 | -7.4 | -2.3 | -4.1 | -4.8 |
| Class E | -21.5 | -32.4 | -40.2 | -21.6 | -35.1 | -42.0 |
| **Overall** | **-7.3** | **-12.8** | **-14.8** | **-7.4** | **-13.0** | **-14.3** |
| Enc Time[%] | 100% | | | 100% | | |
| Dec Time[%] | 93% | | | 96% | | |

For class E, the configuration 1 lower resolution IDR frame sizes as a multiple of mean frame size are:

| | Vidyo1 | Vidyo3 | Vidyo4 | Mean (frame periods) | Milliseconds |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| QP=22 | 4.1 | 3.5 | 3.5 | 3.7 | 62 |
| QP=37 | 5.2 | 4.2 | 5.0 | 4.8 | 80 |

# 6 Recommendations

It is recommended that support for dynamic resolution be incorporated into the HEVC Working Draft and HM, and that an AHG be started to study the technology, covering issues such as filter design and the resolutions to be used.

# Patent rights declaration(s)

**Cisco Systems may have IPR relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# References

[1] ITU-T H.263 (01/2005): Video coding for low bit rate communication, Annex O "Temporal, SNR and Spatial Scalability mode" and Annex P "Reference Picture Resampling"

[2] JCTVC-E700, "Common test conditions and software reference configuration", F. Bossen (DoCoMo), 5th JCT-VC Meeting, Geneva, CH, 16-23 March, 2011

[3] JCTVC-E603_d8 "WD3: Working Draft 3 of High-Efficiency Video Coding", T, Wiegand, W.-J. Han, B. Bross, J.-R. Ohm, G. J. Sullivan (Editors), 5th JCT-VC Meeting, Geneva, CH, 16-23 March 2011