IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF NORTH CAROLINA
WESTERN DIVISION

NOKIA TECHNOLOGIES OY, **Plaintiff,**	
v.	**CIVIL ACTION NO.** 19-CV-0427
LENOVO (SHANGHAI) ELECTRONICS TECHNOLOGY CO. LTD., LENOVO GROUP, LTD., LENOVO BEIJING, LTD., LENOVO PC HK LIMITED, AND LENOVO (UNITED STATES), INC.,	**JURY TRIAL DEMANDED**
Defendants.	

ORIGINAL COMPLAINT

Plaintiff Nokia Technologies Oy ("Nokia") files this Original Complaint against Lenovo (Shanghai) Electronics Technology Co. Ltd., Lenovo Group, Ltd., Lenovo Beijing, Ltd., Lenovo PC HK Limited, and Lenovo (United States), Inc. (collectively, "Lenovo") and alleges as follows:

NATURE OF THE ACTION

1. Nokia is a leading innovator in video coding technology with one of the strongest video coding patent portfolios in the world. Nokia's patented inventions allow video to be transmitted over communications networks, such as WiFi networks, with high quality and dramatically lower bandwidth requirements, and minimize the amount of data it takes to store these videos on mobile devices, like tablets and laptops.

2. Nokia's inventions at issue in this lawsuit include claims essential to decoding video compliant with the H.264 Advanced Video Coding standard promulgated by the International Telecommunications Union ("H.264 Standard"), one of the most widely-used video

AMAZON-1019
6,950,469

decoding standards in the world. Dozens of Lenovo's products decode H.264 video, including the Lenovo IdeaCentre 310S, Lenovo V530 Tower, and Lenovo ThinkPad X1 Carbon, and thus infringe Nokia's essential patent claims asserted in this case. Lenovo has benefitted greatly from Nokia's innovations, which have enabled Lenovo products to more efficiently and effectively stream high quality video.

3. Over fifty companies have taken a license to Nokia's video decoding essential patent claims at rates that are reasonable and non-discriminatory (RAND). Yet despite receiving an offer from Nokia, Lenovo has refused to meaningfully engage in negotiations towards a license covering Nokia's H.264 essential patent claims on reasonable and non-discriminatory terms. Lenovo's refusal to negotiate a license to Nokia's patented H.264 technology in good faith has forced Nokia to institute this lawsuit.

PARTIES

4. Plaintiff Nokia Technologies Oy is a Finnish corporation with its principal place of business at Karakaari 7A, FIN-02610, Espoo, Finland.

5. Defendant Lenovo Group, Ltd. ("Lenovo Group") is a company organized under the laws of the People's Republic of China, with its principal place of business at Lincoln House, 23rd Floor, Taikoo Place, 979 King's Road, Quarry Bay, Hong Kong, China.

6. Defendant Lenovo (Shanghai) Electronics Technology Co. Ltd. is a company organized under the laws of the People's Republic of China, with its principal place of business at Wai Gao Qiao Free Trade Zone, 199 Fenju Road, No. 68 Building, Shanghai, China 200131. Lenovo (Shanghai) Electronics Technology Co. Ltd. operates as a subsidiary of Lenovo Group.

7. Lenovo Beijing, Ltd. is a company organized under the laws of the People's Republic of China, with its principal place of business at Lenovo Building, 6 Chuangye Rd,

Shangdi Haidian District, Beijing, China 100085. Lenovo Beijing, Ltd. operates as a subsidiary of Lenovo Group.

8. Lenovo PC HK Limited is a company organized under the laws of the People's Republic of China, with its principal place of business at Lincoln House, 23rd Floor, Taikoo Place, 979 King's Road, Quarry Bay, Hong Kong, China. Lenovo PC HK Limited operates as a subsidiary of Lenovo Group.

9. Lenovo (United States), Inc. is a company organized under the laws of the Delaware, with its principal place of business at 1009 Think Place, Morrisville, NC 27560. Lenovo (United States), Inc. operates as a subsidiary of Lenovo Group.

10. Together, the Lenovo Defendants design, manufacture, use, import into the United States, sell, and/or offer for sale in the United States tablets, computers, and similar products and services that practice the H.264 Standard. Lenovo's devices are marketed, offered for sale, and/or sold throughout the United States, including within this District.

JURISDICTION AND VENUE

11. This Court has subject matter jurisdiction over the patent infringement claims asserted in this case under 28 U.S.C. §§ 1331 and 1338, and over the non-patent claims under 28 U.S.C. § 1367, as they are so related to the patent infringement claims that they form part of the same case or controversy under Article III of the United States Constitution.

12. Venue is proper in this Court pursuant to 28 U.S.C. §§ 1391 and 1400(b).

13. This Court has personal jurisdiction over each of the Lenovo entities. Lenovo has continuous and systematic business contacts with the State of North Carolina that subject it to the personal jurisdiction of the Court. Lenovo, directly or through subsidiaries or intermediaries (including distributors, retailers, and others), and conducts its business extensively throughout

North Carolina, by shipping, distributing, offering for sale, selling, and advertising (including the provision of interactive web pages) its accused products and services in the State of North Carolina and the Eastern District of North Carolina. Lenovo, directly and through subsidiaries or intermediaries (including distributors, retailers, and others), has purposefully and voluntarily placed its accused products and services into this District and into the stream of commerce with the intention and expectation that they will be purchased and used by consumers in this District. These accused products and services have been and continue to be purchased and used by consumers in this District. Lenovo has committed acts of patent infringement within the State of North Carolina and, more particularly, within the Eastern District of North Carolina. Jurisdiction over Lenovo in the matter is also proper inasmuch as Lenovo has voluntarily submitted itself to the jurisdiction of the courts by commencing litigations within the State of North Carolina, by registering with the North Carolina Secretary of State's Office to do business in the State of North Carolina, and by appointing a registered agent.

14. All of the Lenovo Defendants are part of the same corporate structure and distribution chain for making, importing, offering to sell, selling, and/or using the accused products, including in the State of North Carolina generally and this District in particular. The Lenovo Defendants share the same management, common ownership, advertising platforms, facilities, distribution chains and platforms, and accused product lines and products involving related technologies. Thus, they operate as a unitary business venture.

15. The Lenovo Defendants' activities are directed and controlled by Defendant Lenovo Group, the parent company. For example, Lenovo Group's CEO, Mr. Yang Yuanqing, "formulates and recommends the strategy of the Group"—defined as Lenovo Group and all of its subsidiaries—to the Lenovo Group Board, and then "executes the strategy agreed by the Board."

According to Lenovo Group's 2018/19 Annual Report, "The Group is controlled through the Board who is responsible for steering the success of the Group by overseeing the overall strategy and directing and supervising its affairs in a responsible and effective manner." In other words, the Lenovo Group Board controls all of the Lenovo subsidiaries.

16. Lenovo Group has created a manufacturing, sales, and distribution structure that introduces the accused products into the stream of commerce with the knowledge, expectation, and intent that they will be sold and used in the United States, including in the State of North Carolina and this District. For example, at the company's annual kickoff meeting in Raleigh, North Carolina, Lenovo Group's Chairman and CEO touted its growth in North America, described Lenovo's acquisition engine in the United States as very strong, and stated: "We're very proud of our continued momentum among US consumers." In another interview, Lenovo Group's CFO talked about the company's sales in the United States and its plan to respond to potential US tariffs. Lenovo Group directed Nokia to its representatives based in its US headquarters in Morrisville, NC for negotiations over a license to Nokia's H.264 essential patent claims, including the essential claims in the Patents-in-Suit, for Lenovo Group and its affiliates. Additionally, the Lenovo Group Board of Directors held some of its 2018/19 meetings in Raleigh, North Carolina, including its third quarter Board meeting. Lenovo Group described Raleigh as a "Lenovo key operating center in US." Furthermore, according to Lenovo Group's 2018/19 Annual Report, Lenovo Group contributes to Lenovo (United States) Inc.'s employees' retirement plans.

17. Each of the Lenovo Group subsidiaries named as Defendants is a part of that distribution chain.

18. Defendant Lenovo (Shanghai) Electronics Technology Co. Ltd. applies for UL

and FCC certification for the accused products so that they can be sold in the United States, including North Carolina and this District.

19. Defendant Lenovo Beijing, Ltd. is the registered owner of the Lenovo.com website, where accused products are offered for sale and in fact sold. The website also directs customers to physical locations within North Carolina and within this District where the accused products can be purchased.

20. Defendant Lenovo PC HK Limited manufactures the accused products, including the Lenovo Smart Tab M10.[1] The accused products are designed and manufactured for the United States market, including for use within North Carolina and within this District, to comply with FCC requirements.

21. Alternatively, the facts alleged above demonstrate that Lenovo Group, Lenovo PC HK Limited, Lenovo Beijing, Ltd., and Lenovo (Shanghai) Electronics Technology Co. Ltd. are subject to personal jurisdiction in this Court at least pursuant to Rule 4(k)(2).

22. Defendant Lenovo (United States), Inc. sells the accused products in the United States and maintains its principal place of business in North Carolina and this District.

23. Venue is further proper as to Defendants Lenovo Group; Lenovo (Shanghai) Electronics Technology Co. Ltd.; Lenovo Beijing, Ltd.; and Lenovo PC HK Limited because they are each alien corporations.

[1] *See, e.g.*, the FCC test report for the Lenovo Smart Tab M10, identifying Lenovo PC HK Limited as the manufacturer.

NOKIA'S COMPLIANCE WITH THE ITU COMMON PATENT POLICY AND NOKIA'S RELEVANT DECLARATIONS

A. The International Telecommunications Union ("ITU") and H.264 Standardization Process

24. In order to encourage companies to contribute valuable innovations to the common good, as well as to enable devices manufactured by different entities to interoperate, standards-setting organizations ("SSOs") have formed to promulgate industry standards, which define communication protocols that can be adopted by different manufacturers for their devices.

25. The ITU and the International Standards Organization ("ISO") are SSOs that jointly publish a standard that is referred to as "H.264," "MPEG-4 part 10," or "Advanced Video Coding." The H.264 Standard development process was initiated by the Video Coding Experts Group ("VCEG") and finalized by the Joint Video Team ("JVT"), which was a collaborative effort between VCEG, an ITU group, and the Moving Picture Experts Group ("MPEG"), an ISO group.

26. The ITU was formed in 1865 at the International Telegraph Convention and, in 1947, became a specialized agency of the United Nations, responsible for issues that concern information and communication technologies. The ITU handles a number of different matters and thus is organized into various sectors. One of the sectors is Telecommunication Standardization or "ITU-T." The mission of ITU-T is to ensure efficient and timely production of standards related to the field of telecommunications. The standards developed by ITU-T are referred to as "Recommendations."

27. Within ITU-T, members come together in various teams or groups to propose and contribute innovative technology that best meets the aims of the organization and its members and draft the Recommendations. A goal of ITU-T is to draft Recommendations that incorporate the best available technology to ensure that the standards are the highest possible quality. The

H.264 Standard described above is explicitly detailed in the H.264 Recommendation.

28. In order to facilitate the widespread adoption of its standards, the ITU must ensure both that manufacturers have access to the standards and simultaneously ensure that patent holders have incentives to continue to innovate and contribute technologies to the standards. Without the former, standards could issue, but no one would be able to adopt them. Without the latter, there likely would be no standards at all, and manufacturers would be forced to rely on a multiplicity of proprietary technologies.

29. As it searches for the best available technical solutions, the ITU takes into account that many parts of its standards will be covered by claims in patents owned by members and third parties. In order to assist with the usage of patented technologies in standardized communication protocols, the ITU adopted a Common Patent Policy which states that "a patent embodied fully or partly in a Recommendation | Deliverable must be accessible to everybody without undue constraints." This patent policy applies to the ITU, ITU-T, ISO, and IEC.

30. The ITU has published Patent Guidelines that define the term "Patent," as used in the Common Patent Policy, to be "those claims contained in and identified by patents, utility models and other similar statutory rights based on inventions (including applications for any of these) solely to the extent that any such claims are essential to the implementation of a Recommendation | Deliverable. Essential patents are patents that would be required to implement a specific Recommendation | Deliverable" (ITU Patent Guidelines at 2). The definition of "Patent" provided by the Guidelines is also applicable to the Patent Statement and Licensing Declaration Form, which is prepared by each patent holder when declaring patent claims as essential to H.264. The ITU thus deems "essential" only patent claims that are required for implementation of a specific Recommendation.

31. The H.264 Recommendation specifies the design of decoders and specifically defines the "decoding process" as "[t]he process specified in this Recommendation | International Standard that reads a bitstream and derives decoded pictures from it." *See* ITU-T Rec. H.264, "Advanced video coding for generic audiovisual services" (03/2005) (available at https://www.itu.int/rec/T-REC-H.264-200503-S/en, last visited September 13, 2019) (H.264 Standard at 6). The H.264 Recommendation does not specify the design or operation of video encoders.

B. Nokia's Compliance with the ITU Common Patent Policy and Nokia's Relevant Declarations

32. Consistent with the Common Patent Policy, Nokia timely submitted a Patent Statement and Licensing Declaration to ITU in which it declared in good faith that it is prepared to grant licenses to the essential claims of the relevant patents on RAND terms and conditions.

33. Nokia has negotiated in good faith towards a license agreement with Lenovo consistent with its Patent Statement and Licensing Declaration and the ITU Common Patent Policy.

34. On March 18, 2019, Nokia notified Lenovo that it infringed claims in more than twenty Nokia video decoding patents, including certain of the Patents-in-Suit. Nokia attached a presentation with an overview of Nokia's relevant patents, a list of Lenovo products that infringe Nokia's patent claims, and claim charts describing that infringement. Nokia offered Lenovo a license at its established industry rate and explained that its industry rate is supported by over 50 other companies that have taken a license to use Nokia's H.264 video-coding technology.

35. Nokia reached out to Lenovo several more times, but received no response. On May 22, 2019, Nokia sent Lenovo a letter indicating that Nokia was left with no other option than to conclude that Lenovo refused to discuss a license to Nokia's essential patent claims, and

requesting that Lenovo respond by May 29, 2019. Lenovo finally responded, and agreed to a meeting in Lenovo's Chicago office on June 6, 2019 to discuss the technical merits of Nokia's assertion. Nokia sent high-level licensing executives Robert Gray and Susanna Martikainen, as well as Simon Walker, one of Nokia's lead technical experts, to discuss the essentiality of Nokia's claims, to answer any of Lenovo's technical questions about the claims, to address any response from Lenovo, and to discuss Nokia's license offer. Ms. Martikainen and Mr. Walker traveled to the meeting from Finland.

36. Unfortunately, Lenovo's representatives were unprepared for substantive discussions. Despite having received Nokia's claim charts nearly three months earlier, Lenovo presented no evidence or argument contradicting Nokia's assertion that Lenovo was infringing Nokia's essential patent claims. In addition, Lenovo did not respond to Nokia's licensing offer, would not say whether Lenovo would accept or reject Nokia's offer, and did not state whether Lenovo even intended to make a counter-offer. In good faith, Nokia stated that it would be willing to wait until a telephonic discussion between the parties scheduled for June 25 to receive Lenovo's substantive response to Nokia's March 18 licensing offer. But the day before the June 25 telephone call, Lenovo cancelled the call, while contending that Nokia's offer was unreasonable and unsupportable.

37. Lenovo rescheduled the meeting for a month later. The parties met again on July 24, 2019. At the meeting, Nokia's lead technical expert presented the claim charts that Nokia had previously sent to explain the infringement case, and was prepared to answer any questions from Lenovo. Again, Lenovo was unable to point to any evidence and made no argument contradicting Nokia's claims of essentiality or infringement. Nokia again reiterated its license offer. Lenovo did not accept or reject Nokia's offer, or make a counteroffer.

38. On July 30, 2019, the parties held a telephone conference to continue technical discussions, but they did not discuss specific license terms or conditions. Lenovo again did not provide any evidence or argument contradicting Nokia's infringement allegations.

39. On July 31, 2019, Nokia sent Lenovo an updated claim chart for one of its patents and an updated patent list.

40. On August 27, 2019, Nokia sent Lenovo an anonymized exemplary list of its licenses under its H.264 patent portfolio. The list identified each license's scope, licensed product definition, and the royalty rate. Lenovo has not commented on the list.

41. Nokia has continuously negotiated with Lenovo in good faith in compliance with the ITU Common Patent Policy and Nokia's relevant Patent Statement and Licensing Declarations. Nokia has provided Lenovo with a list of infringed patent claims, detailed proof of Lenovo's infringement, a licensing offer, and the opportunity to meet with high-level executives and technical experts for licensing discussions.

42. Nokia's license offer to Lenovo is consistent with rates agreed to by Nokia's other licensees for the same technology.

43. Lenovo's conduct in its negotiations with Nokia shows Lenovo's lack of good faith.

44. Nokia has complied with the ITU Common Patent Policy and Nokia's relevant Patent Statement and Licensing Declarations to seek the relief it requests in this case. Under such circumstances, this Complaint is necessary to put an end to Lenovo's infringing conduct.

THE NOKIA PATENTS

45. On May 12, 2009, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 7,532,808 ("the '808 Patent"), entitled "Method for Coding Motion in a Video

Sequence," to inventor Jani Lainema. Nokia owns all rights to the '808 Patent necessary to bring this action. A true and correct copy of the '808 Patent is attached hereto as Exhibit 1 and incorporated herein by reference.

46. On September 27, 2005, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 6,950,469 ("the '469 Patent"), entitled "Method for Sub-Pixel Value Interpolation," to Marta Karczewicz and Antti Olli Hallapuro. Nokia owns all rights to the '469 Patent necessary to bring this action. A true and correct copy of the '469 Patent is attached hereto as Exhibit 2 and incorporated herein by reference.

47. On October 9, 2007, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 7,280,599 ("the '599 Patent"), entitled "Method for Sub-Pixel Value Interpolation," to Marta Karczewicz and Antti Olli Hallapuro. Nokia owns all rights to the '599 Patent necessary to bring this action. A true and correct copy of the '599 Patent is attached hereto as Exhibit 3 and incorporated herein by reference.

48. On October 11, 2011, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 8,036,273 ("the '273 Patent"), entitled "Method for Sub-Pixel Value Interpolation," to Marta Karczewicz and Antti Olli Hallapuro. Nokia owns all rights to the '273 Patent necessary to bring this action. A true and correct copy of the '273 Patent is attached hereto as Exhibit 4 and incorporated herein by reference.

49. On March 27, 2012, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 8,144,764 ("the '764 Patent"), entitled "Video Coding," to Miska Hannuksela. Nokia owns all rights to the '764 Patent necessary to bring this action. A true and correct copy of the '764 Patent is attached hereto as Exhibit 5 and incorporated herein by reference.

50. On November 22, 2005, the U.S. Patent and Trademark Office duly and legally

issued U.S. Patent No. 6,968,005 ("the '005 Patent"), entitled "Video coding," to Miska Hannuksela. Nokia owns all rights to the '005 Patent necessary to bring this action. A true and correct copy of the '005 Patent is attached hereto as Exhibit 6 and incorporated herein by reference.

51. On February 15, 2005, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 6,856,701 ("the '701 Patent"), entitled "Method and System for Context-Based Adaptive Binary Arithmetic Coding," to Marta Karczewicz and Ragip Kurceren. Nokia owns all rights to the '701 Patent necessary to bring this action. A true and correct copy of the '701 Patent is attached hereto as Exhibit 7 and incorporated herein by reference.

52. On October 24, 2017, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 9,800,891 ("the '891 Patent"), entitled "Method and Associated Device for Filtering Digital Video Images," to Ossi Kalevo, Emre Aksu, and Marta Karczewicz. Nokia owns all rights to the '891 Patent necessary to bring this action. A true and correct copy of the '891 Patent is attached hereto as Exhibit 8 and incorporated herein by reference.

53. On May 25, 2010, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 7,724,818 ("the '818 Patent"), entitled "Method for Coding Sequences of Pictures," to Miska Hannuksela and Ye-Kui Wang. Nokia owns all rights to the '818 Patent necessary to bring this action. A true and correct copy of the '818 Patent is attached hereto as Exhibit 9 and incorporated herein by reference.

54. On August 28, 2007, the U.S. Patent and Trademark Office duly and legally issued U.S. Patent No. 7,263,125 ("the '125 Patent"), entitled "Method and Device for Indicating Quantizer Parameters in a Video Coding System," to Jani Lainema. Nokia owns all rights to the '125 Patent necessary to bring this action. A true and correct copy of the '125 Patent is attached

hereto as Exhibit 10 and incorporated herein by reference.

55. The ’808, ’469, ’599, ’273, ’764, ’005, ’701, ’891, ’818, and ’125 Patents are collectively referred to as the “Patents-in-Suit.”

56. Nokia exclusively owns all rights, title, and interest in the Patents-in-Suit necessary to bring this action, including the right to recover past and future damages. The Patents-in-Suit were previously owned by Nokia’s predecessor-in-interest, Nokia Corporation, and were transferred to Nokia Technologies Oy in 2015. Nokia or its predecessor-in-interest has owned all rights to the Patents-in-Suit necessary to bring this action throughout the period of Lenovo’s infringement and still owns those rights to the Patents-in-Suit. Lenovo is not currently licensed to practice the Patents-in-Suit.

57. The Patents-in-Suit are valid and enforceable.

58. Lenovo has imported into the United States, manufactured, used, marketed, offered for sale, and/or sold in the United States, H.264 devices such as tablets, laptops, and similar products that infringe the Patents-in-Suit.

59. Lenovo’s accused devices (“the Accused Products”) that infringe one or more claims of the Patents-in-Suit include Lenovo products with H.264 video capabilities, such as Lenovo laptop computers, tablets, and similar products that implement or are capable of implementing the H.264 standard.

60. Lenovo has been placed on actual notice of infringement by Nokia prior to the filing of this Complaint as to certain of the Patents-in-Suit. At a minimum, in accordance with 35 U.S.C. § 287, Lenovo has had actual notice and knowledge of Nokia’s charge of infringement as to all the Patents-in-Suit at least as early as the filing of this Original Complaint and/or the date this Original Complaint was served upon Lenovo. Despite such notice, Lenovo continues to

make, use, import into, market, offer for sale, and/or sell in the United States products that infringe the Patents-in-Suit.

GENERAL ALLEGATIONS

61. Lenovo has directly and indirectly infringed and continues to directly and indirectly infringe each of the Patents-in-Suit by engaging in acts constituting infringement under 35 U.S.C. § 271(a), (b), and/or (c), including but not necessarily limited to one or more of making, using, selling, offering to sell, and inducing and contributing to infringement by others, in this District and elsewhere in the United States, and importing into the United States, the Accused Products.

62. Lenovo's acts of infringement have caused damage to Nokia. Nokia is entitled to recover from Lenovo the damages sustained by Nokia as a result of Lenovo's wrongful acts in an amount subject to proof at trial.

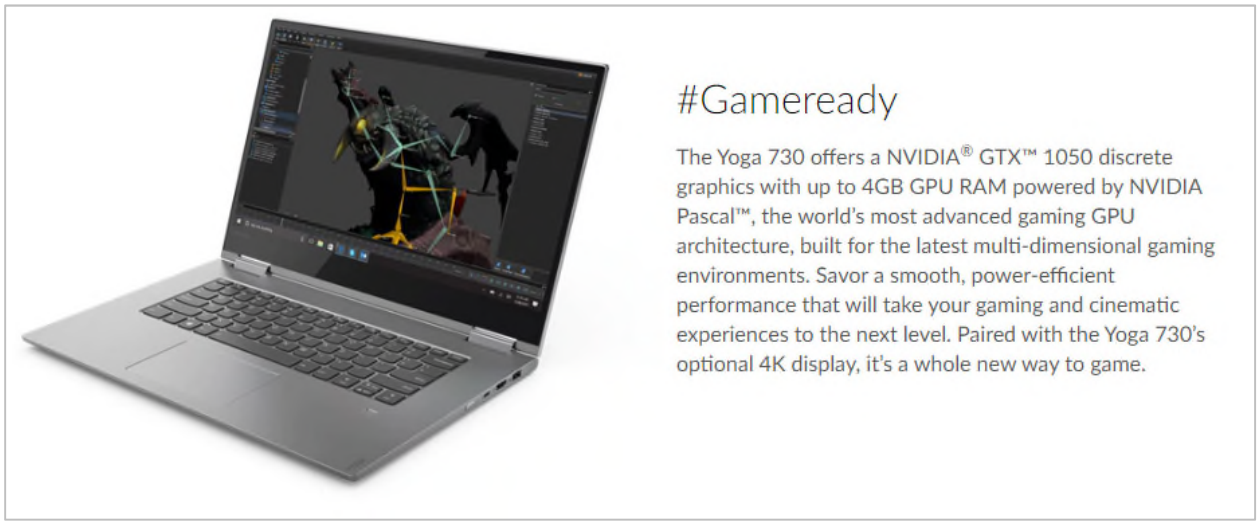
63. Lenovo's infringement of the Patents-in-Suit has been and continues to be willful. Lenovo has committed and continues to commit acts of infringement despite a high likelihood that its actions constitute infringement, and Lenovo knew or should have known that its actions constituted an unjustifiably high risk of infringement.

64. In the interest of providing detailed averments of infringement, Nokia has identified below at least one claim per patent to demonstrate infringement. However, the selection of claims should not be considered limiting, and additional claims of the Patents-in-Suit that are infringed by Lenovo will be disclosed in compliance with the Court's rules related to infringement contentions and discovery.

LENOVO'S ACCUSED PRODUCTS

A. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the '808 Patent.

65. The Accused Products infringe one or more claims of the '808 patent, including for example, claim 16. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15" Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:



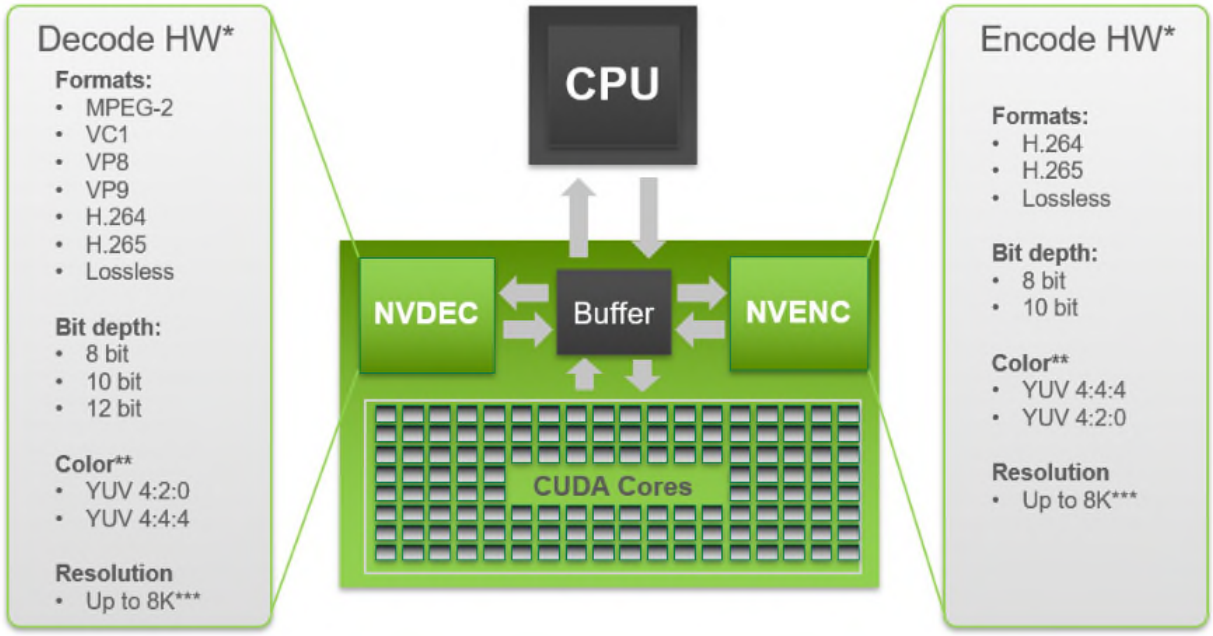
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

66. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

67. Thus, for example and as shown below, the Accused Products infringe claim 16 of the ’808 Patent by virtue of their compatibility with and practice of the H.264 Standard.[2] For example, the Accused Products comprise a video decoder for decoding an encoded video sequence, the decoder comprising a demultiplexer for receiving an indication of a skip coding mode assigned to a first segment. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3, 7.3.4, 7.4.4, and 7.4.5.

3 Definitions

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

. . .

3.75 macroblock: A 16x16 block of luma samples and two corresponding blocks of chroma samples . . .

. . .

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

. . .

Source: H.264 Standard § 3.

[2] The infringement evidence cited herein includes exemplary citations to H.264 (03/2005). The cited functionality for each claim of infringement has been included in versions of the H.264 standard since March 2005 and remains in current versions of the H.264 standard.

7.3.4 Slice data syntax

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

Source: H.264 Standard § 7.3.4.

7.4.4 Slice data semantics

. . .

mb_skip_run specifies the number of consecutive skipped macroblocks for which, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as a P macroblock type . . .

. . .

mb_skip_flag equal to 1 specifies that for the current macroblock, when decoding a P or SP slice, mb_type shall be inferred to be P_Skip and the macroblock type is collectively referred to as P macroblock type . . .

Source: H.264 Standard at § 7.4.4.

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

. . .

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

. . .

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

The following semantics are assigned to the macroblock types in Table 7-13.

. . .

– P_Skip: no further data is present for the macroblock in the bitstream.

Source: H.264 Standard at § 7.4.5.

68. The Accused Products further comprise a video decoder for decoding an encoded

video sequence, the decoder further comprising a motion compensated prediction block for assigning either a zero motion vector or a predicted non-zero motion vector for the skip coding mode for the first segment based at least in part on the motion information of a second segment neighbouring the first segment. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 8.4.1, 8.4.1.1, and 8.4.1.3.

6.4.8 Derivation processes for neighbouring macroblocks, blocks, and partitions

. . .

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

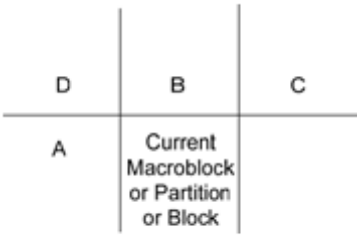


Figure 6-14 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

Source: H.264 Standard at § 6.4.8.

8.4.1 Derivation process for motion vector components and reference indices

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

Outputs of this process are

– luma motion vectors mvL0 and mvL1 as well as the chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags predFlagL0 and predFlagL1

– a sub-partition macroblock motion vector count variable subMvCnt

For the derivation of the variables mvL0 and mvL1 as well as refIdxL0 and refIdxL1, the following applies.

– If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1. mvL1 and refIdxL1 are marked as not available and predFlagL1 is set equal to 0. The sub-partition motion vector count variable subMvCnt is set equal to 1.

– Otherwise, if mb_type is equal to B_Skip or B_Direct_16x16 or sub_mb_type[mbPartIdx] is equal to B_Direct_8x8, the derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8 in B slices in subclause 8.4.1.2 is invoked with mbPartIdx and subMbPartIdx as the input and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, the sub- partition motion vector count subMvCnt, and the prediction utilization flags predFlagL0 and predFlagL1.

– Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, refIdxLX, and in Pred_LX and in the syntax elements ref_idx_lX and mvd_lX, the following applies.

. . .

Source: H.264 Standard at § 8.4.1.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0. The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0.

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

– The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

– The variable mvL0 is specified as follows.

– If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

– mbAddrA is not available

– mbAddrB is not available

– refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

– refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

– Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.

NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

Source: H.264 Standard at § 8.4.1.1.

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the reference index of the current partition refIdxLX (with X being 0 or 1),

– the variable currSubMbType.

Output of this process is the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = X (with X being 0 or 1 for refIdxLX being refIdxL0 or refIdxL1, respectively) as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX,

mvpLX = mvLXB

– MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX,

mvpLX = mvLXA

– MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16,

mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX,

mvpLX = mvLXA

MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 1, and refIdxLXC is equal to refIdxLX,

mvpLX = mvLXC

. . .

Source: H.264 Standard at § 8.4.1.3.

8.4.1.3.1 Derivation process for median luma motion vector prediction

Inputs to this process are

– the neighbouring partitions mbAddrN\mbPartIdxN\subMbPartIdxN (with N being replaced by A, B, or C),

– the motion vectors mvLXN (with N being replaced by A, B, or C) of the neighbouring partitions,

– the reference indices refIdxLXN (with N being replaced by A, B, or C) of the neighbouring partitions, and

– the reference index refIdxLX of the current partition. Output

of this process is the motion vector prediction mvpLX. The variable

mvpLX is derived as follows:

– When both partitions mbAddrB\mbPartIdxB\subMbPartIdxB and mbAddrC\mbPartIdxC\subMbPartIdxC are not available and mbAddrA\mbPartIdxA\subMbPartIdxA is available,

mvLXB = mvLXA mvLXC = mvLXA

refIdxLXB = refIdxLXA refIdxLXC =

refIdxLXA

– Depending on reference indices refIdxLXA, refIdxLXB, or refIdxLXC, the following applies.

– If one and only one of the reference indices refIdxLXA, refIdxLXB, or refIdxLXC is equal to the reference index refIdxLX of the current partition, the following applies. Let refIdxLXN be the reference index that is equal to refIdxLX, the motion vector mvLXN is assigned to the motion vector prediction mvpLX:

mvpLX = mvLXN

– Otherwise, each component of the motion vector prediction mvpLX is given by the

median of the corresponding vector components of the motion vector mvLXA, mvLXB, and mvLXC:

mvpLX[0] = Median(mvLXA[0], mvLXB[0], mvLXC[0]) mvpLX[1] =

Median(mvLXA[1], mvLXB[1], mvLXC[1])

Source: H.264 Standard at § 8.4.1.3.1.

8.4.3.2 Derivation process for motion data of neighbouring partitions

Inputs to this process are

– the macroblock partition index mbPartIdx,

– the sub-macroblock partition index subMbPartIdx,

– the current sub-macroblock type currSubMbType,

– the list suffix flag listSuffixFlag

Outputs of this process are (with N being replaced by A, B, or C)

– mbAddrN\mbPartIdxN\subMbPartIdxN specifying neighbouring partitions,

– the motion vectors mvLXN of the neighbouring partitions, and

– the reference indices refIdxLXN of the neighbouring partitions.

Variable names that include the string "LX" are interpreted with the X being equal to listSuffixFlag.

The partitions mbAddrN\mbPartIdxN\subMbPartIdxN with N being either A, B, or C are derived in the following

ordered steps.

1. Let mbAddrD\mbPartIdxD\subMbPartIdxD be variables specifying an additional neighbouring partition.

2. The process in subclause 6.4.8.5 is invoked with mbPartIdx, currSubMbType, and subMbPartIdx as input and the output is assigned to mbAddrN\mbPartIdxN\subMbPartIdxN with N being replaced by A, B, C, or D.

3. When the partition mbAddrC\mbPartIdxC\subMbPartIdxC is not available, the following applies

mbAddrC = mbAddrD mbPartIdxC =

mbPartIdxD subMbPartIdxC = subMbPartIdxD

The motion vectors mvLXN and reference indices refIdxLXN (with N being A, B, or C) are derived as follows.

– If the macroblock partition or sub-macroblock partition

mbAddrN\mbPartIdxN\subMbPartIdxN is not available or mbAddrN is coded in Intra prediction mode or predFlagLX of mbAddrN\mbPartIdxN\subMbPartIdxN is equal to 0, both components of mvLXN are set equal to 0 and refIdxLXN is set equal to –1.

– Otherwise, the following applies.

– The motion vector mvLXN and reference index refIdxLXN are set equal to MvLX[mbPartIdxN][subMbPartIdxN] and RefIdxLX[mbPartIdxN], respectively, which are the motion vector mvLX and reference index refIdxLX that have been assigned to the (sub-)macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN.

– The variables mvLXN[1] and refIdxLXN are further processed as follows.

– If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock

mvLXN[1] = mvLXN[1] / 2

refIdxLXN = refIdxLXN * 2

– Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock

mvLXN[1] = mvLXN[1] * 2

refIdxLXN = refIdxLXN / 2

– Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

Source: H.264 Standard at § 8.4.1.3.

69. The Accused Products further comprise a video decoder for decoding an encoded video sequence, the decoder further comprising a motion compensated prediction block for forming a prediction for the first segment with respect to a reference frame based at least in part on the assigned motion vector for the skip coding mode, wherein the assigned motion vector is one of the zero motion vector and the predicted non-zero motion vector. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 8.4 and 8.4.2.

8.4 Inter prediction process

This process is invoked when decoding P and B macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array predL of luma samples and when chroma_format_idc is not equal to 0 (monochrome) two 8x8 arrays predCr and predCb of chroma samples, one for each of the chroma components Cb and Cr.

The partitioning of a macroblock is specified by mb_type. Each macroblock partition is referred to by mbPartIdx. When the macroblock partitioning consists of partitions that are equal to sub-macroblocks, each sub-macroblock can be further partitioned into sub-macroblock partitions as specified by sub_mb_type. Each sub-macroblock partition is referred to by subMbPartIdx. When the macroblock partitioning does not consist of sub-macroblocks, subMbPartIdx is set equal to 0**. . .**

The Inter prediction process for a macroblock partition mbPartIdx and a sub-macroblock partition subMbPartIdx consists of the following ordered steps

. . .

1. Derivation process for motion vector components and reference indices as specified in subclause 8.4.1.

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

Outputs of this process are

– luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) the chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags predFlagL0 and predFlagL1

– the sub-macroblock partition motion vector count subMvCnt.

2. The variable MvCnt is incremented by subMvCnt.

3. Decoding process for Inter prediction samples as specified in subclause 8.4.2.

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available), and partHeightC (if available)

– luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) the chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

– prediction list utilization flags predFlagL0 and predFlagL1

Outputs of this process are

– inter prediction samples (pred); which are a (partWidth)x(partHeight) array predPartL of prediction luma samples and when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays predPartCr, and predPartCb of prediction chroma samples, one for each of the chroma components Cb and Cr.

. . .

Source: H.264 Standard at § 8.4.

8.4.2 Decoding process for Inter prediction samples

Inputs to this process are

– a macroblock partition mbPartIdx,

– a sub-macroblock partition subMbPartIdx.

– variables specifying partition width and height for luma and chroma (if available), partWidth, partHeight, partWidthC (if available) and partHeightC (if available)

– luma motion vectors mvL0 and mvL1 and when chroma_format_idc is not equal to 0 (monochrome) chroma motion vectors mvCL0 and mvCL1

– reference indices refIdxL0 and refIdxL1

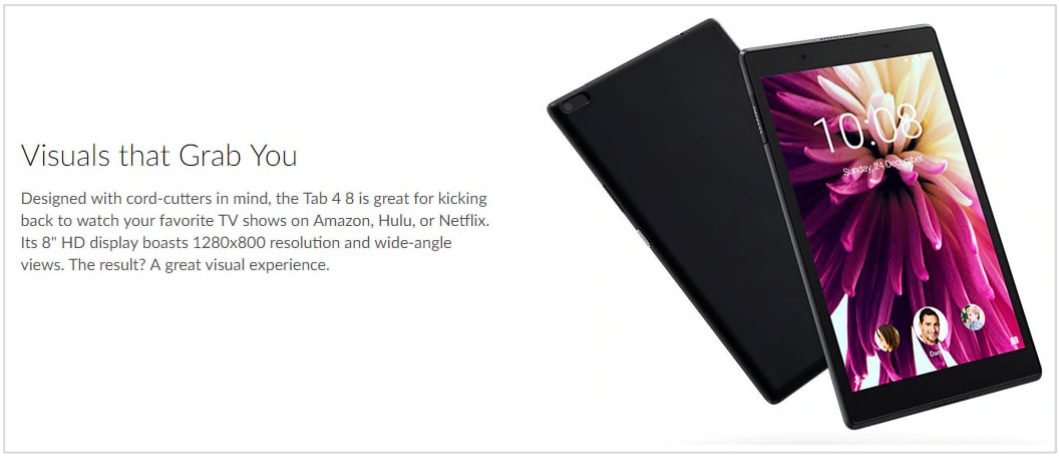
Outputs of this process are

– the Inter prediction samples predPart, which are a (partWidth)x(partHeight) array predPartL of prediction luma samples, and when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays $predPart_{Cb}$, $predPart_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr.

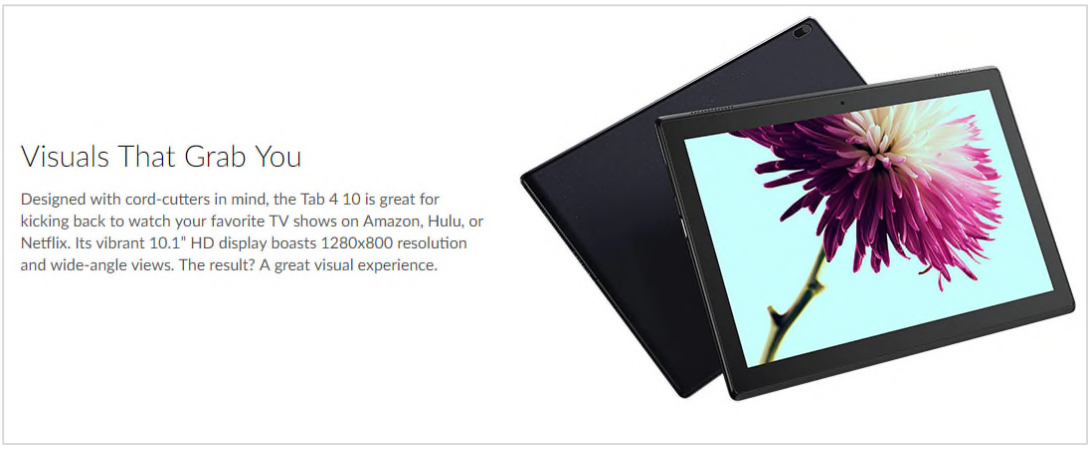
Source: H.264 Standard at § 8.4.2.

70. Thus, as described above, the Accused Products, including the Lenovo Yoga 730 15” Platinum laptop, infringe one or more claims of the ’808 Patent, including claim 16.

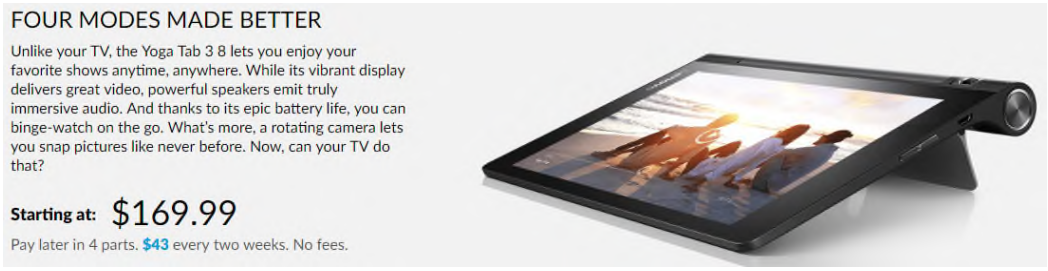
71. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’808 Patent. For example, Lenovo advertises the ability to stream video from services that use H.264 codecs on its products, claiming that some can even replace conventional televisions for streaming video:



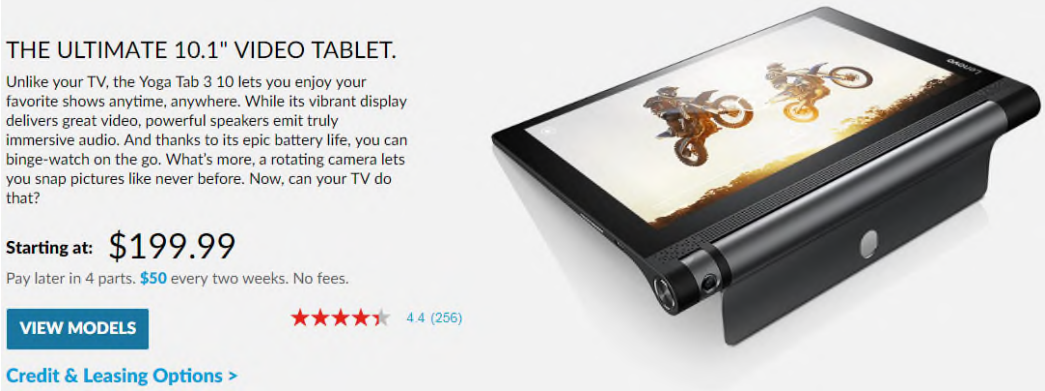
https://www.lenovo.com/us/en/tablets/android-tablets/tab-4-series/Lenovo-TB-8504/p/ZZITZTATB08 (last accessed on September 13, 2019)



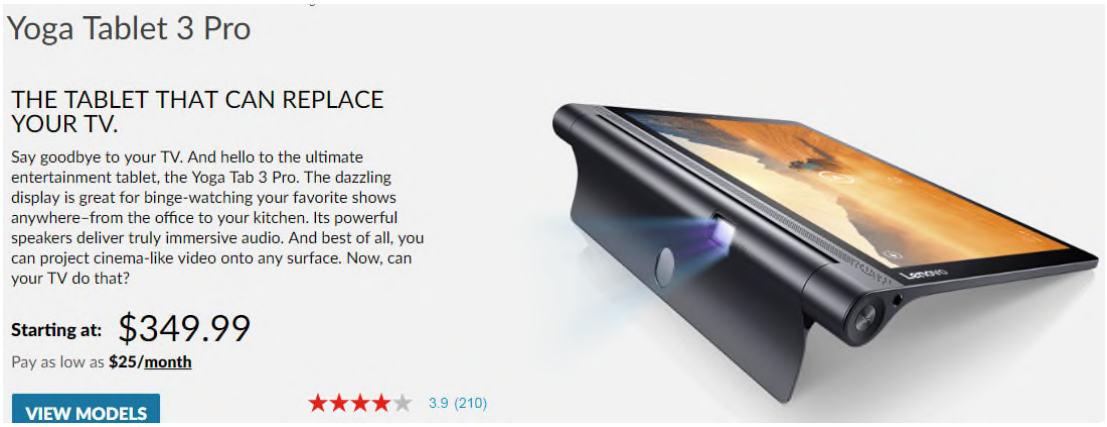
https://www.lenovo.com/us/en/tablets/android-tablets/tab-4-series/Lenovo-TB-X304/p/ZZITZTATB0X (last accessed September 13, 2019)



https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-8/p/ZZITZTBYT0F (last accessed September 13, 2019)

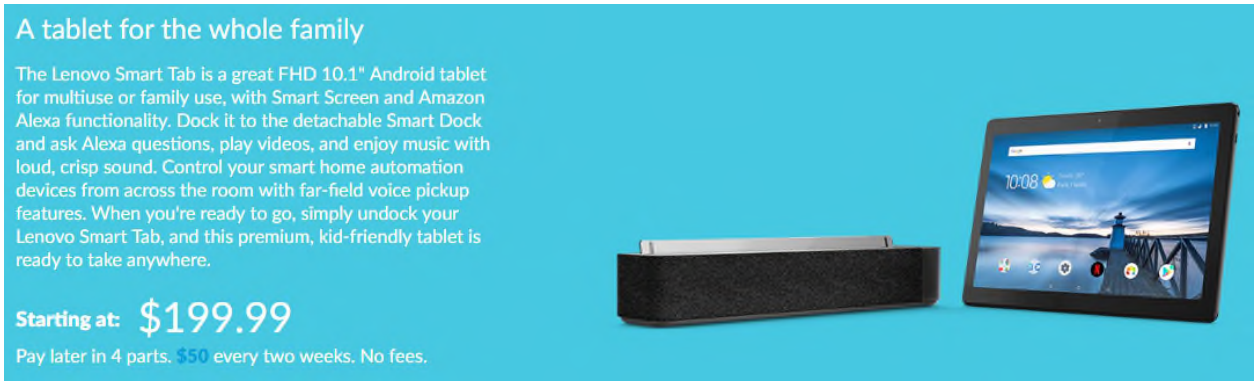


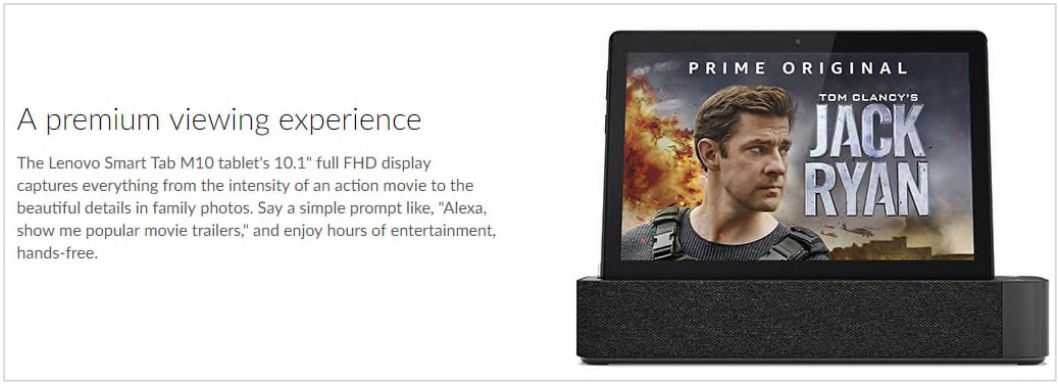
https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-10/p/ZZITZTBYT2F (last accessed September 13, 2019)



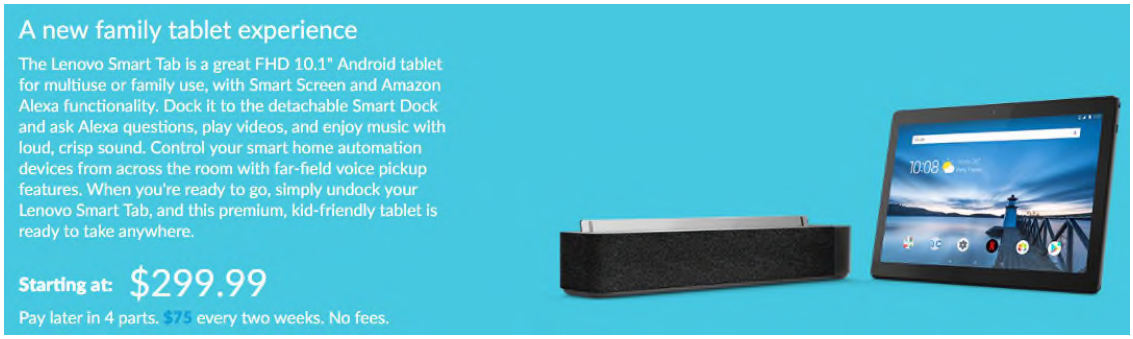
https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-Pro/p/ZZITZTBYT1F (last accessed September 13, 2019)

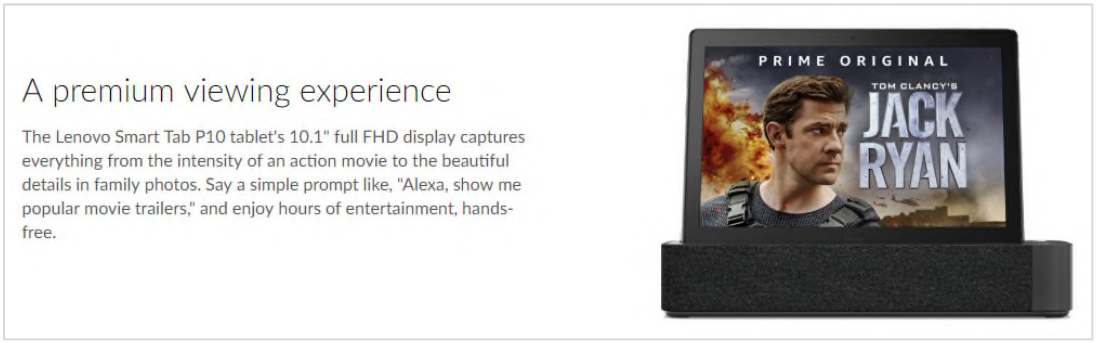
https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-E10/p/ZZITZTATB9X (last accessed September 13, 2019)





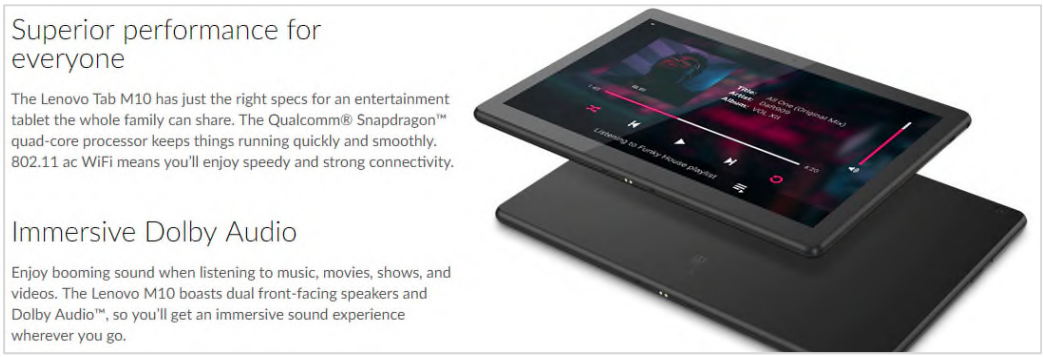
https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-M10/p/ZZITZTATBAX (last accessed September 13, 2019)





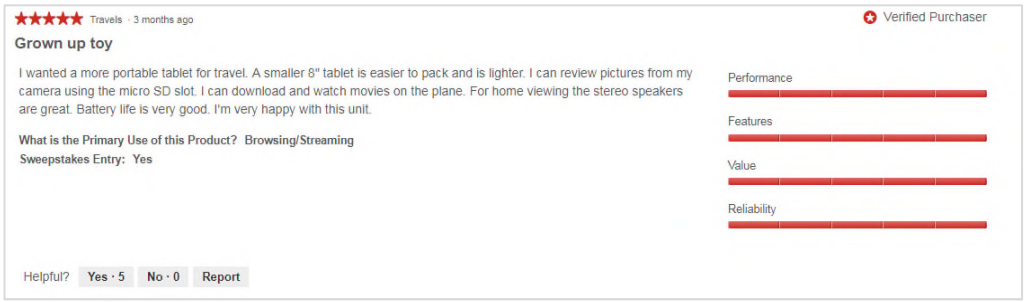
https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Tab-P10/p/ZZITZTATB7X (last accessed September 13, 2019)

https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-E8/p/ZZITZTATB48 (last accessed September 13, 2019)

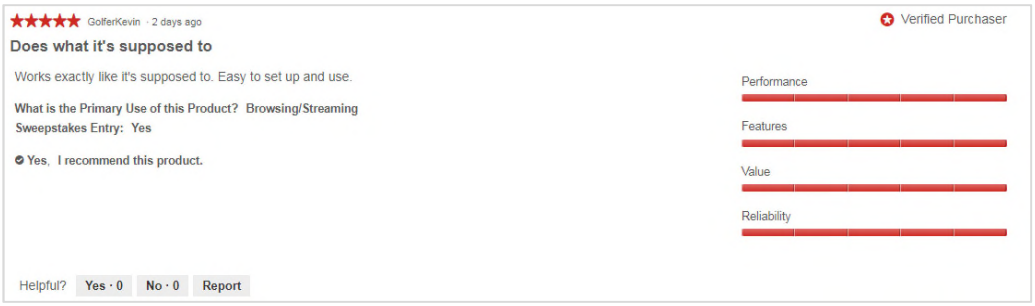


https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Tab-M10/p/ZZITZTATBBX (last accessed September 13, 2019)

72. Additionally, Lenovo publishes customer reviews of the Accused Products that show its customers actually use the Accused Products to stream video and recommending the Accused Products to others:

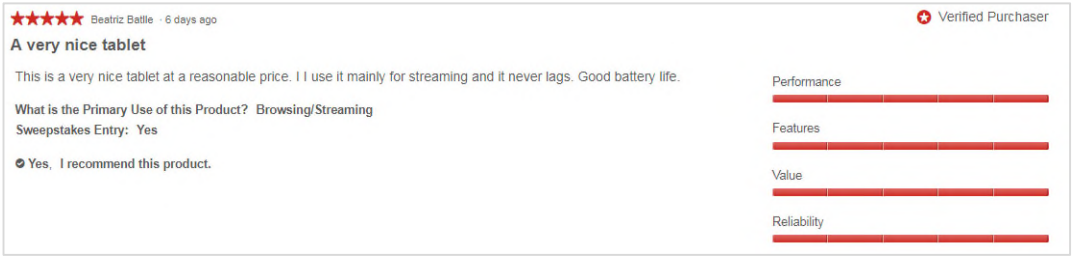


https://www.lenovo.com/us/en/tablets/android-tablets/tab-4-series/Lenovo-TB-8504/p/ZZITZTATB08 (last accessed on September 13, 2019)

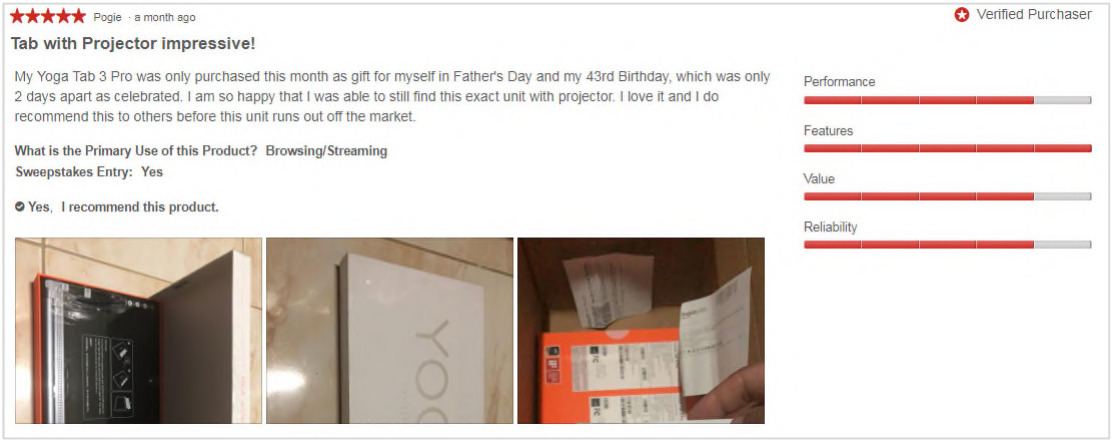


https://www.lenovo.com/us/en/tablets/android-tablets/tab-4-series/Lenovo-TB-X304/p/ZZITZTATB0X (last accessed September 13, 2019)

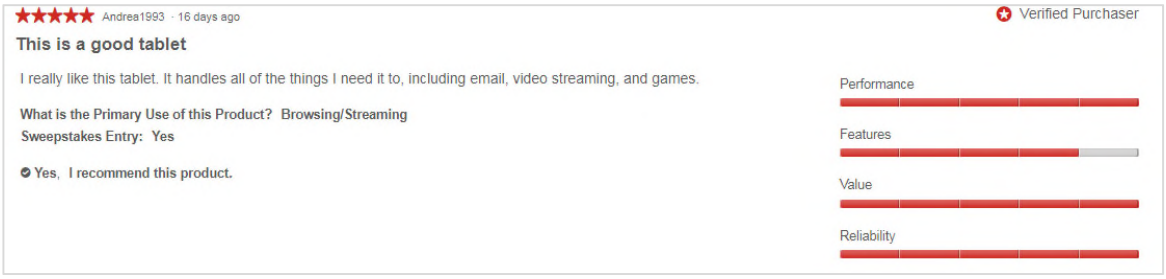
https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-8/p/ZZITZTBYT0F (last accessed September 13, 2019)



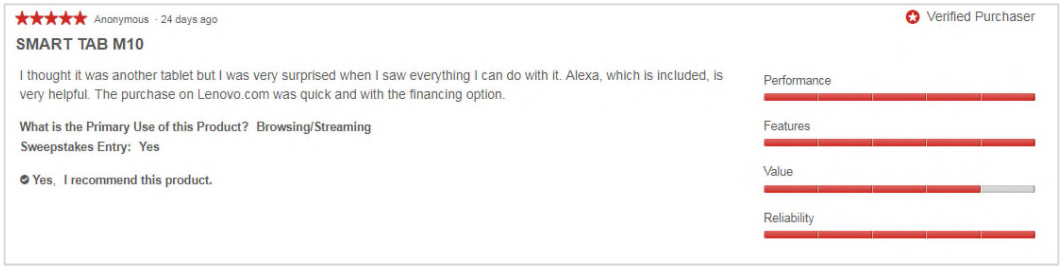
https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-10/p/ZZITZTBYT2F (last accessed September 13, 2019)



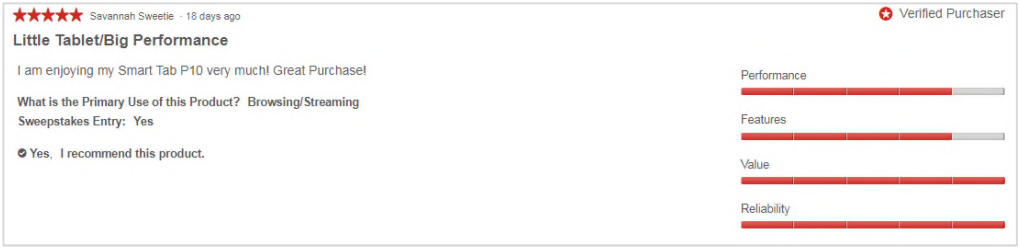
https://www.lenovo.com/us/en/tablets/android-tablets/yoga-tab-3-series/Yoga-Tab-3-Pro/p/ZZITZTBYT1F (last accessed September 13, 2019)



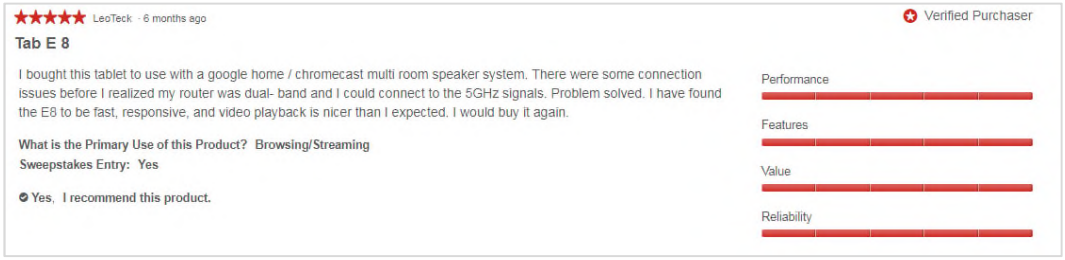
https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-E10/p/ZZITZTATB9X (last accessed September 13, 2019)



https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-M10/p/ZZITZTATBAX (last accessed July 23, 2019)



https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Tab-P10/p/ZZITZTATB7X (last accessed September 13, 2019)

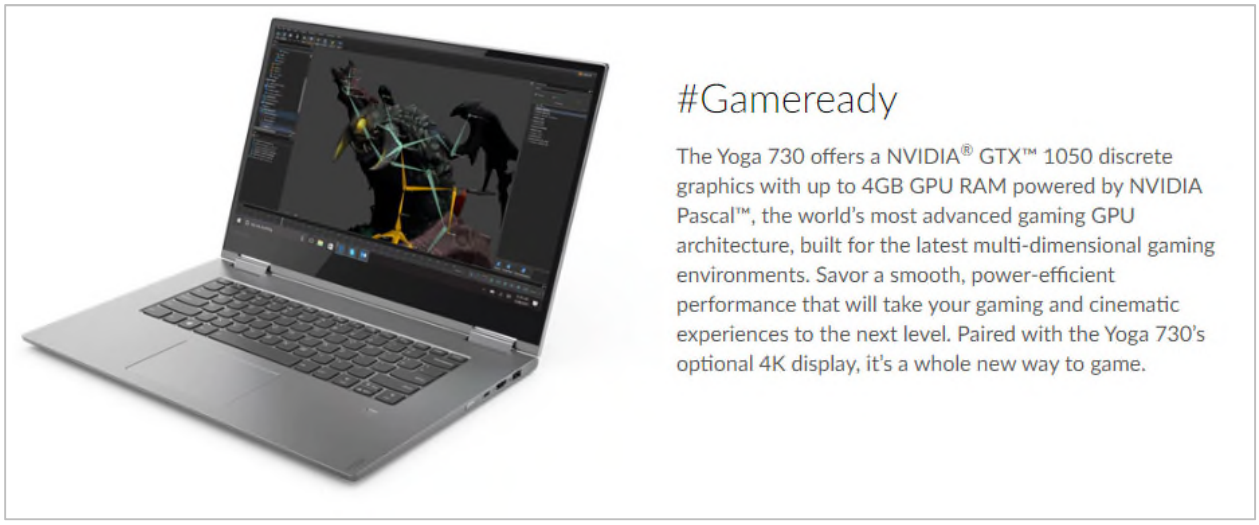


https://www.lenovo.com/us/en/tablets/android-tablets/lenovo-tab-series/Lenovo-Tab-E8/p/ZZITZTATB48 (last accessed September 13, 2019)

B. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the '469 Patent.

73. The Accused Products infringe one or more claims of the '469 patent, including for example, claim 27. Each of the Accused Products is compliant with the H.264 Standard. For

example, as shown below, the Lenovo Yoga 730 15” Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:



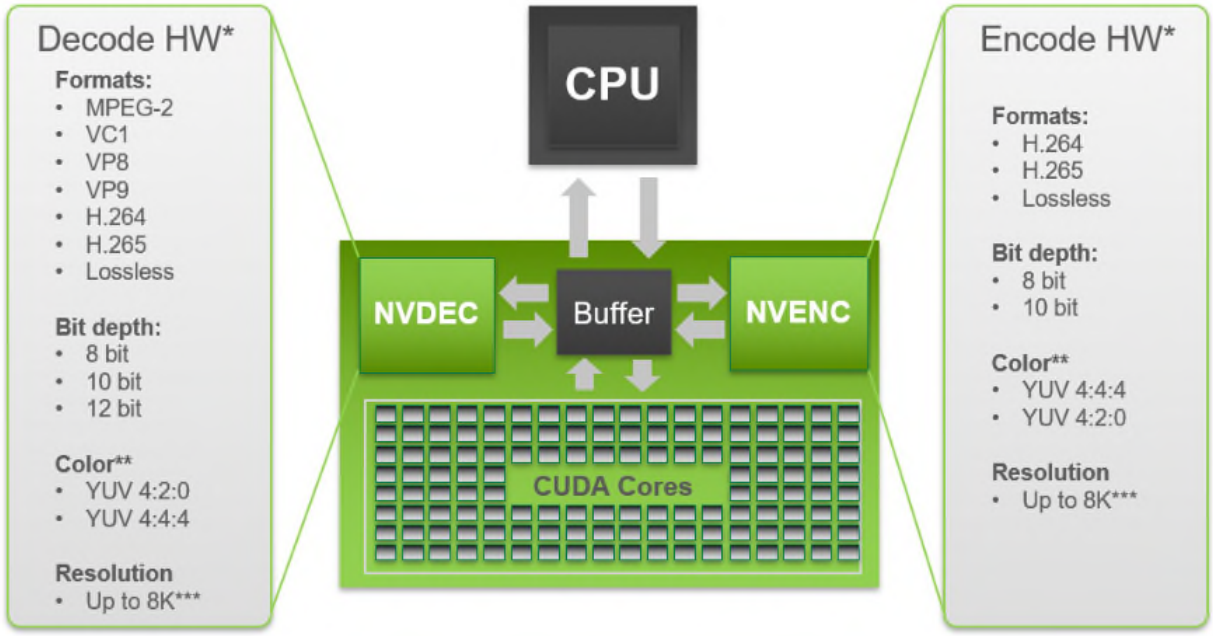
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

74. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

75. Thus, for example and as shown below, the Accused Products infringe claim 27 of the ’469 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products comprise a communications terminal including an H.264 decoder, which comprises a video coder for coding an image comprising pixels arranged in rows and columns and represented by values having a specified dynamic range, the pixels in the rows residing at unit horizontal locations and the pixels in the columns residing at unit vertical locations. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 6.2, 7.3, 7.4, and 8.4.2.2.

6.2 Source, decoded, and output picture formats

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

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

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

- Luma (Y) only (monochrome), with or without an auxiliary array.
- Luma and two Chroma (YCbCr or YCgCo), with or without an auxiliary array.
- Green, Blue and Red (GBR, also known as RGB), with or without an auxiliary array.

Source: H.264 Standard at § 6.2.

7.4.2.1 Sequence parameter set RBSP semantics

. . .

bit_depth_luma_minus8 specifies the bit depth of the samples of the luma array . . . as specified by

$$BitDepth_Y = 8 + bit_depth_luma_minus8 \quad (7\text{-}1)$$

. . .

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

bit_depth_chroma_minus8 specifies the bit depth of the samples of the chroma arrays . . . as specified by

$$BitDepth_C = 8 + bit_depth_chroma_minus8 \quad (7\text{-}3)$$

. . .

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

. . .

Source: H.264 Standard at § 7.4.2.1.

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 100 || profile_idc == 110 || profile_idc == 122 || profile_idc == 144) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc == 3)		
residual_colour_transform_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)

. . .

Source: H.264 Standard at § 7.3.2.1.

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

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

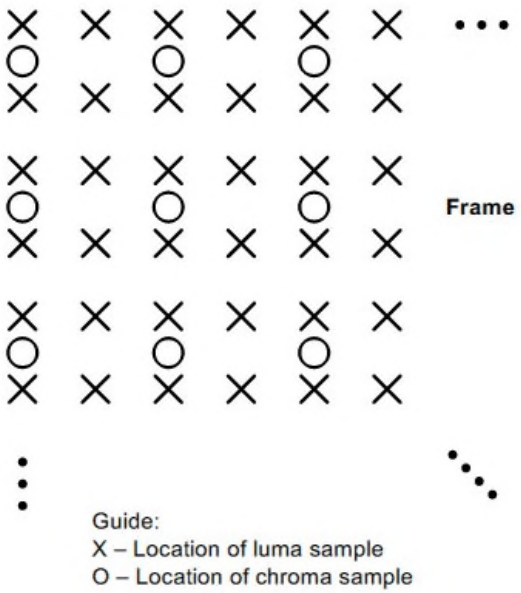


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

Source: H.264 Standard at § 6.2.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

- a luma location in full-sample units ($xInt_L$, $yInt_L$),
- a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and
- the luma sample array of the selected reference picture $refPicLX_L$

Output of this process is a predicted luma sample value $predPartLX_L[x_L, y_L]$.

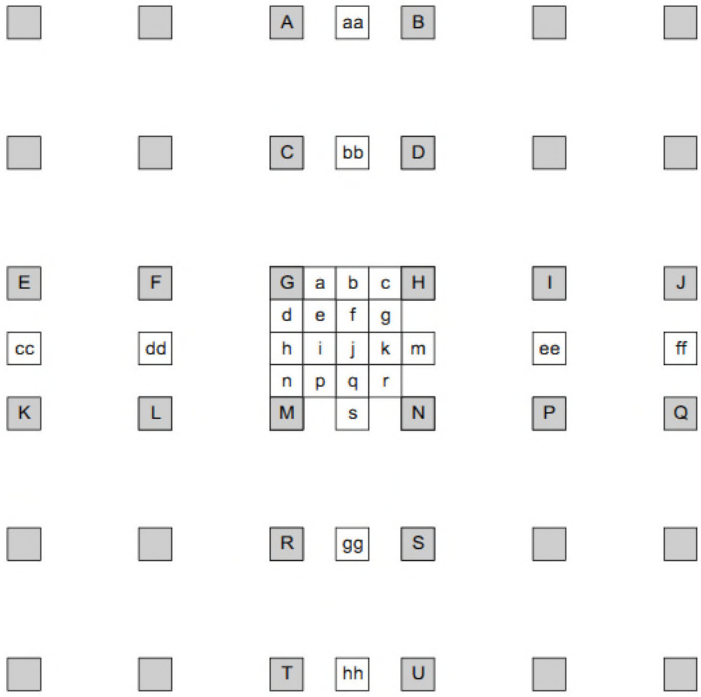


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

Source: H.264 Standard at § 8.4.2.2.1.

76. The Accused Products further comprise the video coder comprising an interpolator adapted to generate values for sub-pixels at fractional horizontal and vertical locations, the fractional horizontal and vertical locations being defined according to $½^x$, where x is a positive integer having a maximum value N. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.

8.4.2.2 Fractional sample interpolation process

Inputs to this process are

- the current partition given by its partition index mbPartIdx and its sub-macroblock partition index subMbPartIdx,
- the width and height partWidth, partHeight of this partition in luma-sample units,
- a luma motion vector mvLX given in quarter-luma-sample units,
- a chroma motion vector mvCLX given in eighth-chroma-sample units, and
- the selected reference picture sample arrays refPicLX$_L$, refPicLX$_{Cb}$, and refPicLX$_{Cb}$

Outputs of this process are

- a (partWidth)x(partHeight) array predPartLX$_L$ of prediction luma sample values and
- when chroma_format_idc is not equal to 0 (monochrome) two (partWidthC)x(partHeightC) arrays predPartLX$_{Cb}$, and predPartLX$_{Cr}$ of prediction chroma sample values.

Source: H.264 Standard at § 8.4.2.2.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

- a luma location in full-sample units (xInt$_L$, yInt$_L$),
- a luma location offset in fractional-sample units (xFrac$_L$, yFrac$_L$), and
- the luma sample array of the selected reference picture refPicLX$_L$

Output of this process is a predicted luma sample value predPartLX$_L$[x$_L$, y$_L$].

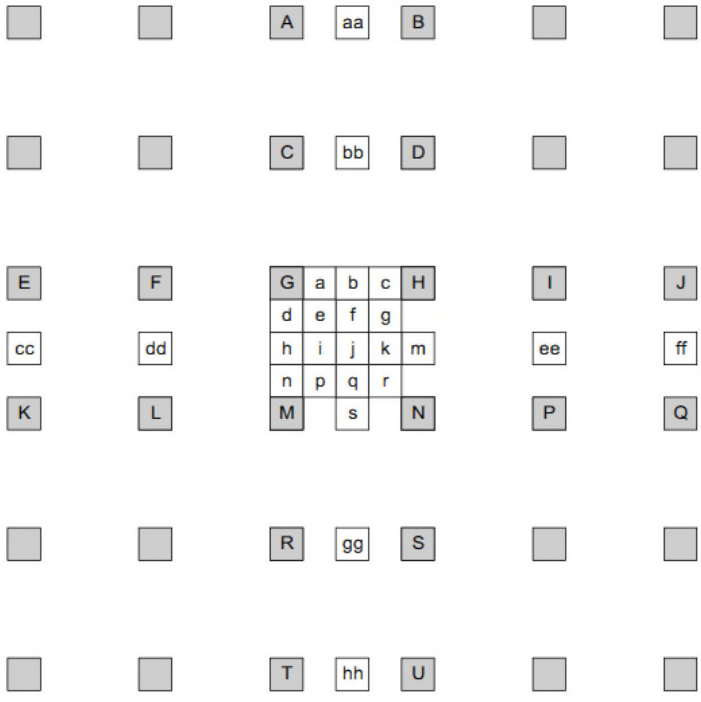


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

Source: H.264 Standard at § 8.4.2.2.1.

Table 8-11 – Differential full-sample luma locations

Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P	Q	R	S	T	U
xDZ_L	0	1	0	1	-2	-1	0	1	2	3	-2	-1	0	1	2	3	0	1	0	1
yDZ_L	-2	-2	-1	-1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	3	3

Source: H.264 Standard at § 8.4.2.2.1.

Table 8-12 – Assignment of the luma prediction sample $predPartLX_L[x_L, y_L]$

$xFrac_L$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
$yFrac_L$	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
$predPartLX_L[x_L, y_L]$	G	d	h	n	a	e	i	p	b	f	j	q	c	g	k	r

Source: H.264 Standard at § 8.4.2.2.1.

77. The Accused Products further comprise the interpolator being adapted to (a) interpolate values for sub-pixels at $½^{N-1}$ unit horizontal and unit vertical locations, and unit horizontal and $½^{N-1}$ unit vertical locations directly using weighted sums of pixels residing at unit horizontal and unit vertical locations. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to Figure 8-4 and § 8.4.2.2.1.

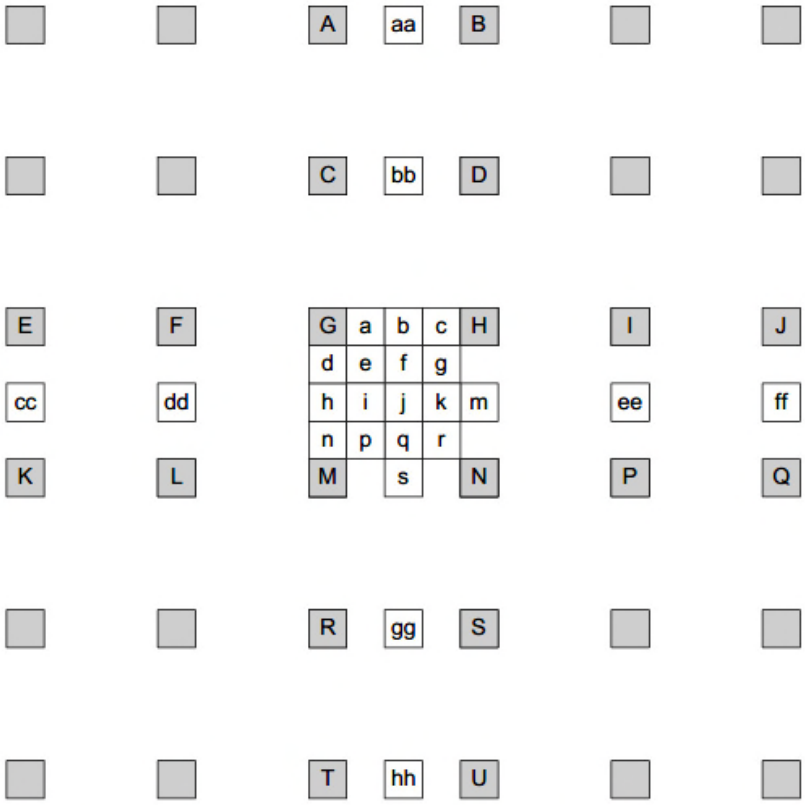


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation.

Source: H.264 Standard at § 8.4.2.2.1.

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}241)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}242)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (8\text{-}243)$$

Source: H.264 Standard at § 8.4.2.2.1.

78. The Accused Products further comprise the interpolator being adapted to (b) interpolate values for sub-pixels at $½^{N-1}$ unit horizontal and $½^{N-1}$ unit vertical locations directly

using a choice of a first weighted sum of values for sub-pixels residing at $½^{N-1}$ unit horizontal and unit vertical locations and a second weighted sum of values for sub-pixels residing at unit horizontal and $½^{N-1}$ unit vertical locations, the first and second weighted sums of values being calculated according to step (a). As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}241)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}242)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (8\text{-}243)$$

Source: H.264 Standard at § 8.4.2.2.1.

79. The Accused Products further comprise the interpolator being adapted to (c) interpolate a value for a sub-pixel situated at a $½^N$ unit horizontal and $½^N$ unit vertical location by taking a weighted average of the value of a first sub-pixel or pixel situated at a $½^{N-m}$ unit horizontal and $½^{N-n}$ unit vertical location and the value of a second sub-pixel or pixel located at a $½^{N-p}$ unit horizontal and $½^{N-q}$ unit vertical location, variables m, n, p and q taking integer values in the range 1 to N such that the first and second sub-pixels or pixels are located diagonally with respect to the sub-pixel at $½^N$ unit horizontal and $½^N$ vertical location. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}254)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}255)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}256)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}257)$$

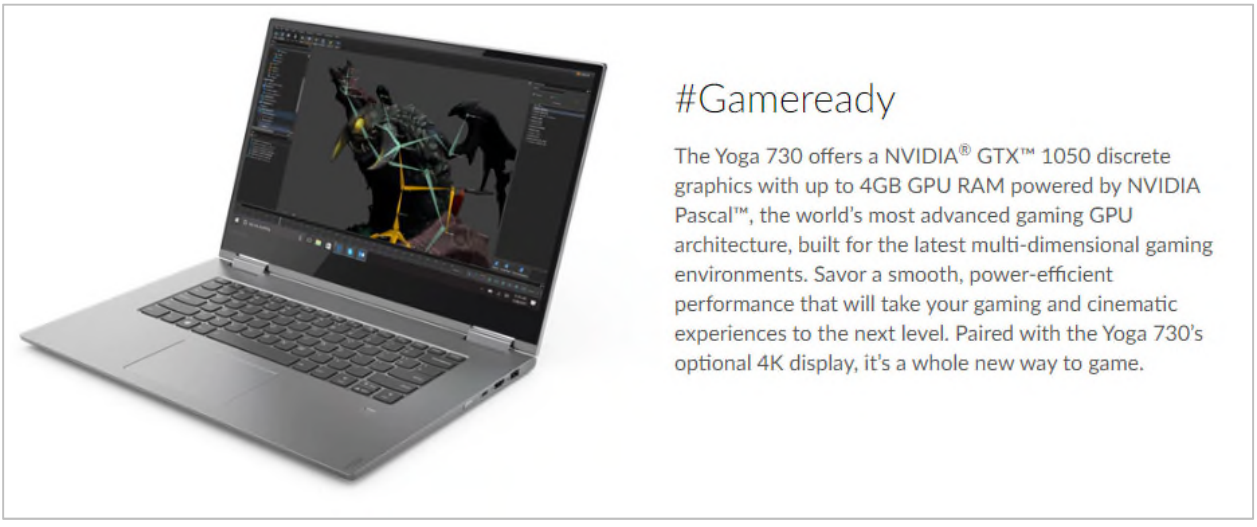
Source: H.264 Standard at § 8.4.2.2.1.

80. Thus, as described above, the Accused Products, including the Lenovo Yoga 730 15" Platinum laptop, infringe one or more claims of the '469 Patent, including claim 27.

81. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the '469 patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

C. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the '599 Patent.

82. The Accused Products infringe one or more claims of the '599 patent, including for example, claim 22. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15" Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:



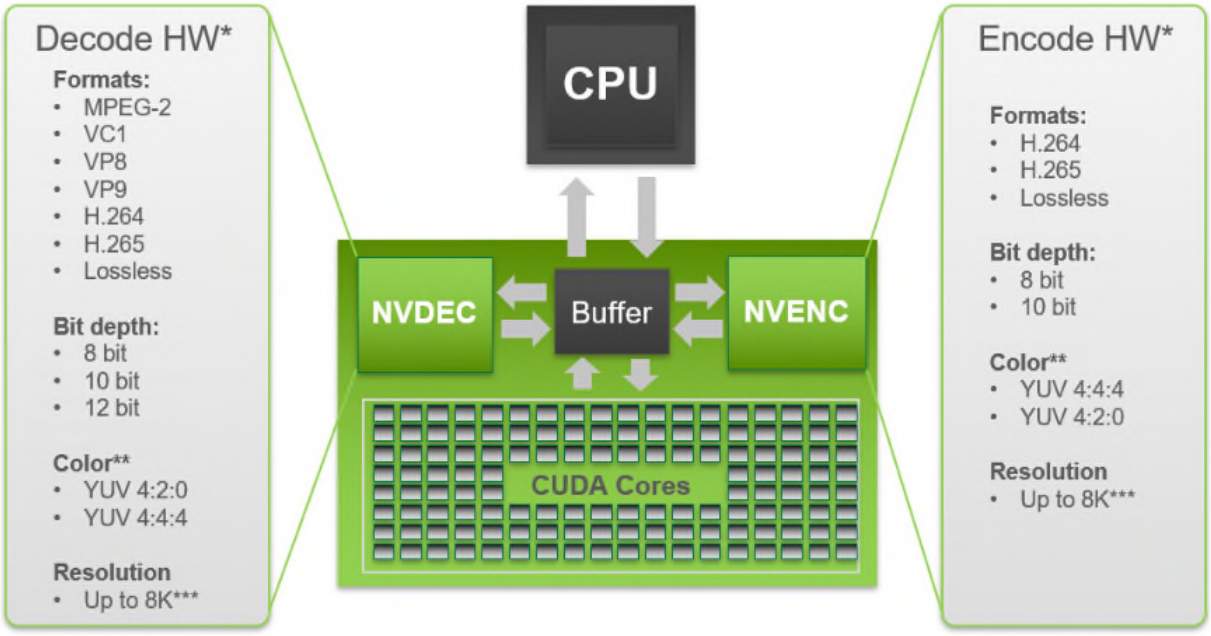
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

83. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

84. Thus, for example and as shown below, the Accused Products infringe claim 22 of the ’599 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products comprise an apparatus for sub-pixel value interpolation, the apparatus being operable to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being arranged in rows and columns, the pixel and sub-pixel locations being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation. This structure and functionality is described in the H.264 Standard, including but not limited to §§ 6.2, 8.4.2.2 and 8.4.2.2.1.

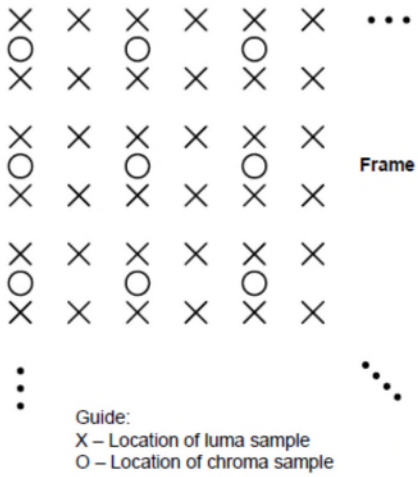


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

Source: H.264 Standard at § 6.1.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

- a luma location in full-sample units ($xInt_L$, $yInt_L$),
- a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and
- the luma sample array of the selected reference picture $refPicLX_L$

Output of this process is a predicted luma sample value $predPartLX_L[x_L, y_L]$.

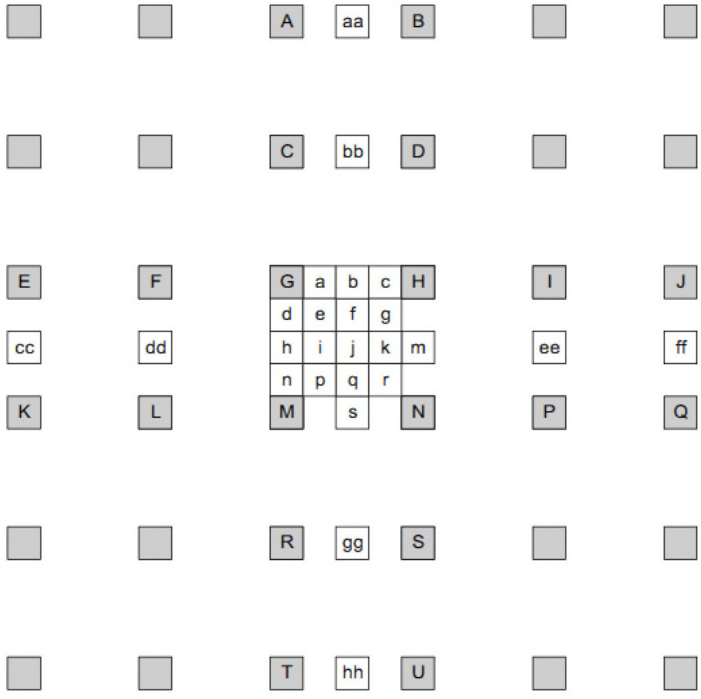


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

Source: H.264 Standard at § 8.4.2.2.1.

85. The Accused Products also comprise circuitry operable to interpolate a sub-pixel value for a sub-pixel having co-ordinates with odd values of both K and L, according to a predetermined choice of a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at co-ordinates 1/2, 1/2, and a weighted average of the values of a pair of diagonally-opposed sub-pixels having co-ordinates with even values of both K and L, including zero, situated within a quadrant of the rectangular bounded region defined by corner pixels having co-ordinates 1/2, 1/2 and the nearest neighbouring pixel. This structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}254)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}255)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}256)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}257)$$

Source: H.264 Standard at § 8.4.2.2.1.

86. The Accused Products also comprise circuitry operable to interpolate sub-pixel values for sub-pixels having co-ordinates with K equal to an even value and L equal zero and sub-pixels having co-ordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having co-ordinates with odd values of both K and L, using weighted sums of the values of pixels located in rows and columns respectively. This structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as b_1 by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as h_1 by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}237)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}238)$$

The final prediction values b and h are derived using:

$$b = \mathrm{Clip1}_Y((b_1 + 16) >> 5) \quad (8\text{-}239)$$
$$h = \mathrm{Clip1}_Y((h_1 + 16) >> 5) \quad (8\text{-}240)$$

Source: H.264 Standard at § 8.4.2.2.1.

87. The Accused Products also comprise circuitry operable to interpolate sub-pixel values for sub-pixels having co-ordinates with even values of both K and L, used in the interpolation of sub-pixel values for the sub-pixels having co-ordinates with odd values of both K and L, using a predetermined choice of either a weighted sum of the values of sub-pixels having co-ordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding co-ordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having co-ordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding co-ordinates in immediately adjacent bounded rectangular regions. This structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}241)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}242)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = \mathrm{Clip1}_Y((j_1 + 512) >> 10) \quad (8\text{-}243)$$

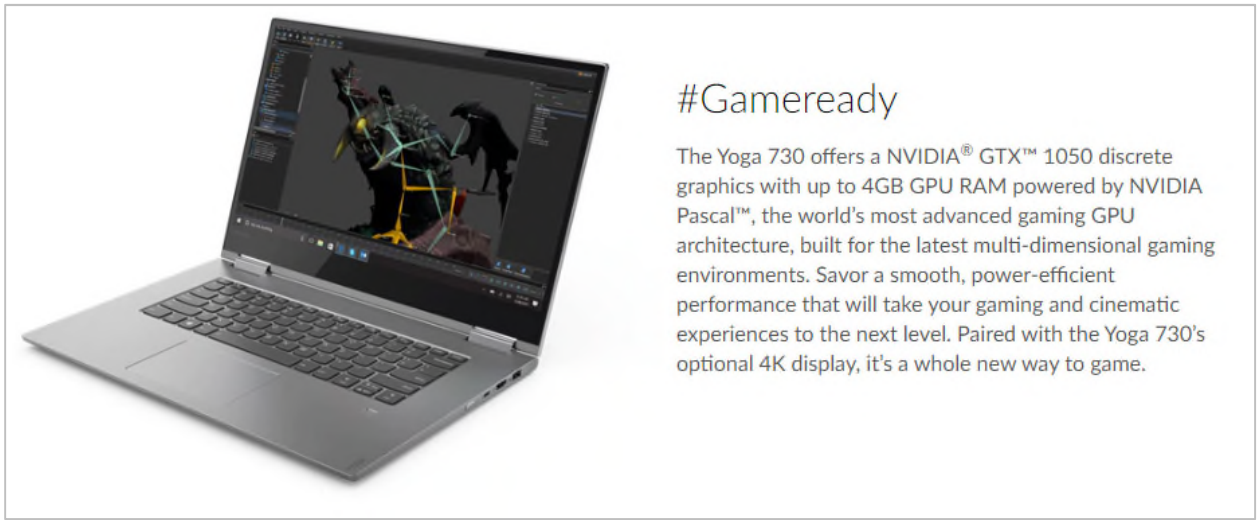
Source: H.264 Standard at § 8.4.2.2.1.

88. Thus, as described above the Accused Products, including the Lenovo Yoga 730 15" Platinum laptop, infringe one or more claims of the '599 Patent, including claim 22.

89. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the '599 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

D. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the '273 Patent.

90. The Accused Products infringe one or more claims of the '273 patent, including for example, claim 33. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15" Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:



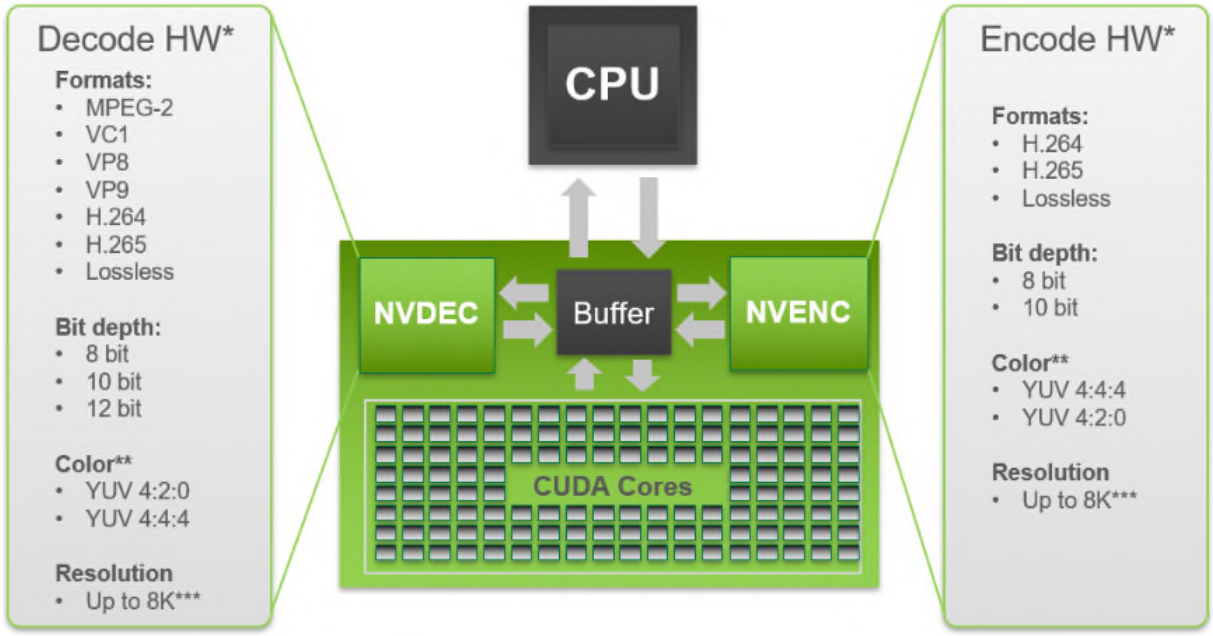
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

91. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

92. Thus, for example and as shown below, the Accused Products infringe claim 33 of the ’273 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products comprise an interpolator for sub-pixel value interpolation, the interpolator being configured to determine values for sub-pixels situated within a rectangular bounded region defined by four corner pixels with no intermediate pixels between the corners, the pixels and sub-pixels being configured for display in rows and columns, pixel and sub-pixel locations in the rows and columns being representable mathematically within the rectangular bounded region using the co-ordinate notation $K/2^N$, $L/2^N$, K and L being positive integers having respective values between zero and 2^N, N being a positive integer greater than one and representing a particular degree of sub-pixel value interpolation. This structure and functionality is described in the H.264 Standard, including but not limited to §§ 6.2, 8.4.2.2 and 8.4.2.2.1.

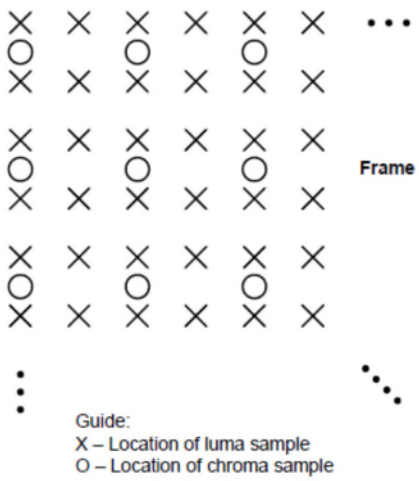


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

Source: H.264 Standard at § 6.1.

8.4.2.2.1 Luma sample interpolation process

Inputs to this process are

- a luma location in full-sample units ($xInt_L$, $yInt_L$),
- a luma location offset in fractional-sample units ($xFrac_L$, $yFrac_L$), and
- the luma sample array of the selected reference picture $refPicLX_L$

Output of this process is a predicted luma sample value $predPartLX_L[x_L, y_L]$.

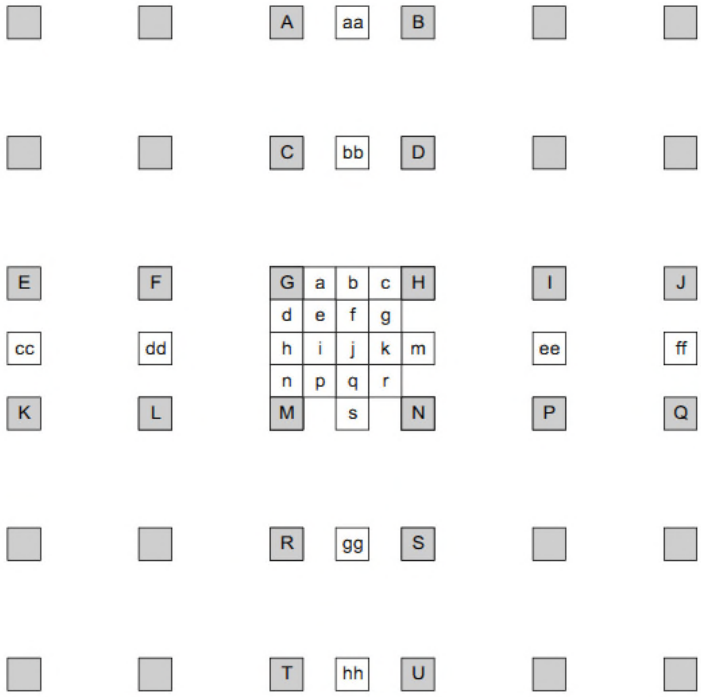


Figure 8-4 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation

Source: H.264 Standard at § 8.4.2.2.1.

93. The Accused Products also comprise an interpolator configured to interpolate a sub-pixel value for a sub-pixel having coordinates with odd values of K and L, according to a predetermined choice of either a weighted average of the value of a nearest-neighbouring pixel and the value of the sub-pixel situated at coordinates ½, ½, or a weighted average of the values of a pair of diagonally-opposed sub-pixels having coordinates with even values of K and L, including zero, situated within a quadrant of the rectangular bounded region, the quadrant being defined by the sub-pixel having coordinates ½, ½ and the nearest neighbouring pixel. This

structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at quarter sample positions labelled as e, g, p, and r are derived by averaging with upward rounding of the two nearest samples at half sample positions in the diagonal direction using

$$e = (b + h + 1) >> 1 \quad (8\text{-}254)$$
$$g = (b + m + 1) >> 1 \quad (8\text{-}255)$$
$$p = (h + s + 1) >> 1 \quad (8\text{-}256)$$
$$r = (m + s + 1) >> 1. \quad (8\text{-}257)$$

Source: H.264 Standard at § 8.4.2.2.1.

94. The Accused Products also comprise an interpolator configured to interpolate sub-pixel values for sub-pixels having coordinates with K equal to an even value and L equal to zero and sub-pixels having coordinates with K equal to zero and L equal to an even value, used in the interpolation of the sub-pixels having coordinates with odd values of K and L, using weighted sums of the values of pixels located in rows and columns respectively. This structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at half sample positions labelled b are derived by first calculating intermediate values denoted as b_1 by applying the 6-tap filter to the nearest integer position samples in the horizontal direction. The samples at half sample positions labelled h are derived by first calculating intermediate values denoted as h_1 by applying the 6-tap filter to the nearest integer position samples in the vertical direction:

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (8\text{-}237)$$
$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (8\text{-}238)$$

The final prediction values b and h are derived using:

$$b = Clip1_Y((b_1 + 16) >> 5) \quad (8\text{-}239)$$
$$h = Clip1_Y((h_1 + 16) >> 5) \quad (8\text{-}240)$$

Source: H.264 Standard at § 8.4.2.2.1.

95. The Accused Products also comprise an interpolator configured to interpolate sub-pixel values for sub-pixels having coordinates with even values of K and L, used in the interpolation of sub-pixel values for the sub-pixels having coordinates with odd values of K and

L, using a predetermined choice of either a weighted sum of the values of sub-pixels having coordinates with K equal to an even value and L equal to zero and the values of sub-pixels having corresponding coordinates in immediately adjacent rectangular bounded regions, or a weighted sum of the values of sub-pixels having coordinates with K equal to zero and L equal to an even value and the values of sub-pixels having corresponding coordinates in immediately adjacent rectangular bounded regions. This structure and functionality is described in the H.264 Standard, including but not limited to § 8.4.2.2.1.

– The samples at half sample position labelled as j are derived by first calculating intermediate value denoted as j_1 by applying the 6-tap filter to the intermediate values of the closest half sample positions in either the horizontal or vertical direction because these yield an equal result.

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff, \text{ or} \quad (8\text{-}241)$$
$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (8\text{-}242)$$

where intermediate values denoted as aa, bb, gg, s_1 and hh are derived by applying the 6-tap filter horizontally in the same manner as the derivation of b_1 and intermediate values denoted as cc, dd, ee, m_1 and ff are derived by applying the 6-tap filter vertically in the same manner as the derivation of h_1. The final prediction value j are derived using:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (8\text{-}243)$$

Source: H.264 Standard at § 8.4.2.2.1.

96. Thus, as described above the Accused Products, including the Lenovo Yoga 730 15” Platinum laptop, infringe one or more claims of the ’273 Patent, including claim 33.

97. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’273 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

E. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the ’764 Patent.

98. The Accused Products infringe one or more claims of the ’764 patent, including for example, claim 46. Each of the Accused Products is compliant with the H.264 Standard. For

example, as shown below, the Lenovo X1 Carbon laptop supports the H.264 video format. Product testing confirms that the Accused Products support the H.264 standard. For example, Plaintiff used an exemplary video sequence encoded in an H.264 bitstream. The H.264 bitstream played back successfully on the Lenovo product, showing that the Lenovo product infringes the asserted claims.

99. Each of the Accused Products infringes claims of the ’764 Patent based on its implementation of H.264. For example and as shown below, the Accused Products infringe claim 46 of the ’764 Patent, in part, by virtue of their compatibility with and practice of the H.264 Standard and specific implementation as shown through testing. For example, the Accused Products comprise an apparatus for decoding an encoded video signal representing a sequence of pictures to form a decoded video signal. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.

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

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

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

3.37 **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.

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

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

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

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

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

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

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

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

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

Source: H.264 Standard at § 3.

100. The Accused Products further comprise such an apparatus wherein the apparatus is configured to examine decoded reference pictures to identify a difference in respective sequence indicator values assigned to consecutively encoded reference pictures. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 8, 8.1, 7.3.1, 7.4.1, 7.3.3, and 7.4.3.

8 Decoding process

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

This clause describes the decoding process, given syntax elements and upper-case variables from clause 7.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Each picture referred to in this clause is a primary picture. Each slice referred to in this clause is a slice of a primary picture. Each slice data partition referred to in this clause is a slice data partition of a primary picture.

An overview of the decoding process is given as follows.

– The decoding of NAL units is specified in subclause 8.1.

– The processes in subclause 8.2 specify decoding processes using syntax elements in the slice layer and above.

. . .

– **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 clause 8.2.5.2 prior to the decoding of any slices of the current picture.**

Source: H.264 Standard at § 8.

8.1 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.2 describes the decoding process for NAL units with nal_unit_type equal to 1 through 5.

Subclauses 8.3 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1, 2, and 5.

Source: H.264 Standard at § 8.1.

7.3.1 NAL unit syntax

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

Source: H.264 Standard at § 7.3.1.

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

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

Source: H.264 Standard at § 7.4.1.

7.3.3 Slice header syntax

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)

Source: H.264 Standard at § 7.3.3.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows.

- If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.
- Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows.
  - If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.
  - Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows.

- If the current picture is an IDR picture, frame_num shall be equal to 0.
- Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true.
  - the current picture and the preceding reference picture belong to consecutive access units in decoding order
  - the current picture and the preceding reference picture are reference fields having opposite parity
  - one or more of the following conditions is true
    - the preceding reference picture is an IDR picture
    - the preceding reference picture includes a memory_management_control_operation syntax element equal to 5

      NOTE 1 – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture

When the value of frame_num is not equal to PrevRefFrameNum, the following applies.

- There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum != frame_num )                    (7-21)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

- The value of frame_num is constrained as follows.
  - If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.

Source: H.264 Standard at § 7.4.3.

101. The Accused Products further comprise such an apparatus wherein the apparatus

is configured to compare the identified difference in sequence indicator values on the basis of an independent numbering scheme in which consecutive reference pictures in encoding order are assigned sequence indicator values that differ with respect to each other by a predetermined amount, independent of one or more of the number of non-reference pictures encoded between consecutive reference pictures, and the number of non-coded pictures between consecutive reference pictures. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 7.4.3 and 7.4.2.1.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows.

- If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.
- Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows.
  - If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.
  - Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows.

- If the current picture is an IDR picture, frame_num shall be equal to 0.
- Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true.
  - the current picture and the preceding reference picture belong to consecutive access units in decoding order
  - the current picture and the preceding reference picture are reference fields having opposite parity
  - one or more of the following conditions is true
    - the preceding reference picture is an IDR picture
    - the preceding reference picture includes a memory_management_control_operation syntax element equal to 5

      NOTE 1 – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture

When the value of frame_num is not equal to PrevRefFrameNum, the following applies.

- There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum != frame_num )                    (7-21)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

- The value of frame_num is constrained as follows.
  - If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.

Source: H.264 Standard at § 7.4.3.

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

Source: H.264 Standard at § 7.4.2.1.

102. The Accused Products further comprise such an apparatus wherein the apparatus is configured to detect corruption or loss of a reference picture if said identified difference in sequence indicator values is more than said predetermined amount. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 8.2.5.2.

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE 1 – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this Recommendation | International Standard are historical.

NOTE 2 – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

Source: H.264 Standard at § 8.2.5.2.

103. Product testing confirms the Accused Products practice this limitation. For example, Plaintiff performed a test on a Lenovo X1 Carbon laptop. Plaintiff used an exemplary video sequence encoded in an H.264 bitstream and simulated a loss of reference pictures by means of a packet loss software, *i.e.* reference pictures were erased from the bitstream to obtain a test bitstream. The resulting H.264 bitstream was played back successfully by the Lenovo product, despite reference pictures required for successful/continued decoding of the bitstream being missing from the test bitstream. From this test, it can be concluded that the infringing decoder makes use of the features described in Note 2 in clause 8.2.5.2 of the standard and thus detects corruption and/or a loss of a reference picture if the values for frame_num of consecutive reference frames do not differ by the predetermined amount one. Otherwise, it would not have been able to take the necessary measures to successfully play back a bitstream with missing reference pictures.

104. Thus, as described above, the Accused Products, including the Lenovo X1 Carbon laptop, infringe one or more claims of the ’764 Patent, including claim 46.

105. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’764 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

F. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the ’005 Patent.

106. The Accused Products infringe one or more claims of the ’005 patent, including for example, claim 9. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo X1 Carbon laptop supports the H.264 video format. Product testing confirms that the Accused Products support the H.264 standard. For example, Plaintiff used an exemplary video sequence encoded in an H.264 bitstream. The H.264 bitstream played back successfully on the Lenovo product, showing that the Lenovo product supports the H.264 standard.

107. Each of the Accused Products infringes claims of the ’005 Patent based on its implementation of H.264. For example and as shown below, the Accused Products infringe claim 9 of the ’005 Patent, in part, by virtue of their compatibility with and practice of the H.264 Standard and specific implementation as shown through testing.

108. For example, the Accused Products comprise a video decoder for decoding an encoded video signal representing a sequence of pictures to form a decoded video signal, the encoded video signal comprising temporally independent INTRA pictures and temporally predicted pictures, wherein the INTRA pictures and at least some of the temporally predicted pictures form reference pictures for the temporal prediction of other pictures. This structure and functionality is described in the H.264 Standard, including but not limited to §§ 3 and 7.4,

and Table 7-6.

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

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

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

3.37 **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.

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

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

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

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

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

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

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

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

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

Source: H.264 Standard at § 3.

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

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

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

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

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

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

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

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

Source: H.264 Standard at § 3.

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

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

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

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

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

Source: H.264 Standard at § 7.4.1.

109. The Accused Products further comprise a video decoder for decoding an encoded video signal, the encoded video signal further comprising a sequence indicator having an independent numbering scheme such that consecutive reference pictures in encoding order are assigned sequence indicator values that differ with respect to each other by a predetermined amount independent of the number of non-reference pictures encoded between successive reference pictures. This structure and functionality is described in the H.264 Standard, including but not limited to § 7.4.

frame_num is used as an identifier for pictures and shall be represented by log2_max_frame_num_minus4 + 4 bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows.

- If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.
- Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows.
  - If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.
  - Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows.

- If the current picture is an IDR picture, frame_num shall be equal to 0.
- Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true.
  - the current picture and the preceding reference picture belong to consecutive access units in decoding order
  - the current picture and the preceding reference picture are reference fields having opposite parity
  - one or more of the following conditions is true
    - the preceding reference picture is an IDR picture
    - the preceding reference picture includes a memory_management_control_operation syntax element equal to 5

      NOTE 1 – When the preceding reference picture includes a memory_management_control_operation syntax element equal to 5, PrevRefFrameNum is equal to 0.
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have frame_num equal to PrevRefFrameNum
    - there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture

When the value of frame_num is not equal to PrevRefFrameNum, the following applies.

- There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of frame_num equal to any value taken on by the variable UnusedShortTermFrameNum in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum  !=  frame_num )                    (7-21)
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

- The value of frame_num is constrained as follows.
  - If gaps_in_frame_num_value_allowed_flag is equal to 0, the value of frame_num for the current picture shall be equal to (PrevRefFrameNum + 1) % MaxFrameNum.

Source: H.264 Standard at § 7.4.3.

110. The Accused Products further comprise a video decoder for decoding, comprising

an input for receiving the encoded video signal and being arranged to decode received encoded pictures, to examine each decoded picture that forms a reference picture to identify the sequence indicator value assigned to the reference picture and to compare the sequence indicator values assigned to consecutively decoded reference pictures to detect loss of a reference picture. This functionality is described in the H.264 Standard, including but not limited to §§ 8 and 8.2.5.2.

– When the frame_num of the current picture is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process for gaps in frame_num is performed according to subclause 8.2.5.2 prior to the decoding of any slices of the current picture.

Source: H.264 Standard at § 8.

8.2.5.2 Decoding process for gaps in frame_num

This process is invoked when frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum.

NOTE 1 – Although this process is specified as a subclause within subclause 8.2.5 (which defines a process that is invoked only when nal_ref_idc is not equal to 0), this process may also be invoked when nal_ref_idc is equal to 0 (as specified in clause 8). The reasons for the location of this subclause within the structure of this Recommendation | International Standard are historical.

NOTE 2 – This process can only be invoked for a conforming bitstream when gaps_in_frame_num_value_allowed_flag is equal to 1. When gaps_in_frame_num_value_allowed_flag is equal to 0 and frame_num is not equal to PrevRefFrameNum and is not equal to (PrevRefFrameNum + 1) % MaxFrameNum, the decoding process should infer an unintentional loss of pictures.

Source: H.264 Standard at § 8.2.5.2.

111. Product testing confirms the Accused Products practice this limitation. For example, Plaintiff performed a test on a Lenovo X1 Carbon laptop. Plaintiff used an exemplary video sequence encoded in an H.264 bitstream and simulated a loss of reference pictures by means of a packet loss software, *i.e.* reference pictures were erased from the bitstream to obtain a test bitstream. The resulting H.264 bitstream was played back successfully by the Lenovo product, despite reference pictures required for successful/continued decoding of the bitstream being missing from the test bitstream. From this test, it can be concluded that the infringing decoder makes use of the features described in Note 2 in clause 8.2.5.2 of the standard and thus detects corruption and/or a loss of a reference picture if the values for frame_num of consecutive reference frames do not differ by the predetermined amount one. Otherwise, it would not have

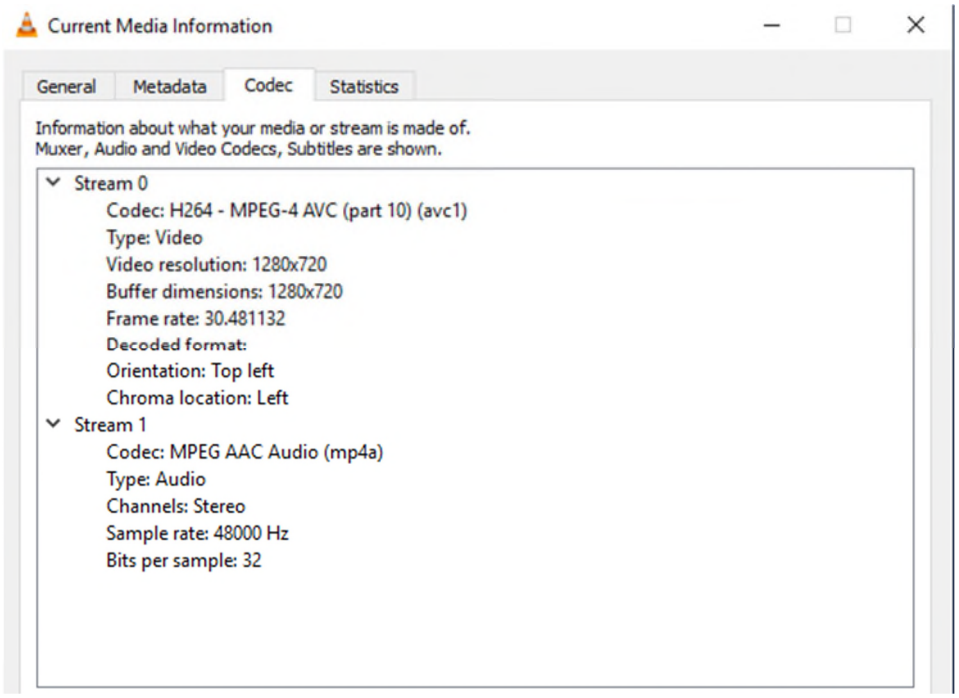
been able to take the necessary measures to successfully play back a bitstream with missing reference pictures.

112. Thus, as described above the Accused Products, including the Lenovo X1 Carbon laptop, infringe one or more claims of the ’005 Patent, including claim 9.

113. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’005 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

G. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the ’701 Patent.

114. The Accused Products infringe one or more claims of the ’701 patent, including for example, claim 18. Product testing confirms that the Accused Products can encode an H.264 bitstream. For example, Plaintiff recorded an H.264 video using the default settings on a Lenovo X1 Carbon laptop. The screenshot below shows that the video encoded on the Lenovo X1 Carbon laptop is an H.264 video.



115. Thus, for example and as shown below, the Accused Products include an H.264 encoder that infringes claim 18 of the '701 Patent. For example, the Accused Products comprise a system for image coding comprising an input for receiving an image as a plurality of blocks having a plurality of pixels, each pixel having a pixel value. This is shown for example in the reference encoder source code. (*See, e.g.*, ITU-T H.264 – Reference software for ITU-T H.264 advanced video coding (03/2005), at JM10.1a\lencod\src\lencod.c; JM10.1a\lencod\src\image.c; JM10.1a\lencod\src\cabac.c; and JM10.1a\lencod\src\macroblock.c.). On information and belief, the Accused Products operate consistently with the reference encoder. The reference encoder is not part of the H.264 standard.

116. Moreover, though the H.264 standard does not describe encoders, it does show that an H.264 bitstream comprises a plurality of blocks having a plurality of pixels, each pixel having a pixel value.

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

3.84 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.

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

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

Source: H.264 Standard at § 3.

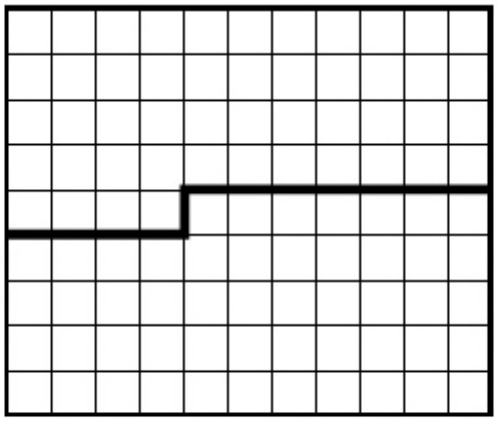


Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices

Source: H.264 Standard at Figure 6-7.

6.3 Spatial subdivision of pictures and slices

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

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

Source: H.264 Standard at § 6.3.

117. Further, the Accused Products comprise a transform coder for performing a transform coding operation on a block of pixels to produce a corresponding block of transform coefficient values. This is shown for example in the reference encoder source code. (*See, e.g.*, ITU-T H.264 – Reference software for ITU-T H.264 advanced video coding (03/2005), at JM10.1a\lencod\src\lencod.c; JM10.1a\lencod\src\image.c; JM10.1a\lencod\src\cabac.c; and JM10.1a\lencod\src\macroblock.c.). On information and belief, the Accused Products operate consistently with the reference encoder. The reference encoder is not part of the H.264 standard.

118. Moreover, though the H.264 standard does not describe encoders, it does show that an H.264 bitstream comprises a block of transform coefficient values representing blocks of

pixels.

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

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

Source: H.264 Standard at § 3.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.
2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

Source: H.264 Standard at § 8.5.1.

8.5.10 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

Depending on the values of qpprime_y_zero_transform_bypass_flag and QP'_Y, the following applies.

– If qpprime_y_zero_transform_bypass_flag is equal to 1 and QP'_Y is equal to 0, the output r is derived as

$r_{ij} = c_{ij}$ with i, j = 0..3 (8-333)

– Otherwise (qpprime_y_zero_transform_bypass_flag is equal to 0 or QP'_Y is not equal to 0), the following text of this process specifies the output.

The variable bitDepth is derived as follows.

– If the input array c relates to a luma residual block, bitDepth is set equal to $BitDepth_Y$.

– Otherwise (the input array c relates to a chroma residual block), bitDepth is set equal to $BitDepth'_C$.

The bitstream shall not contain data that results in any element c_{ij} of c with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)}-1$, inclusive.

The variable sMbFlag is derived as follows.

- If mb_type is equal to SI or the macroblock prediction mode is equal to Inter in an SP slice, sMbFlag is set equal to 1,

- Otherwise (mb_type not equal to SI and the macroblock prediction mode is not equal to Inter in an SP slice), sMbFlag is set equal to 0.

The variable qP is derived as follows.

– If the input array c relates to a luma residual block and sMbFlag is equal to 0

$qP = QP'_Y$ (8-334)

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1

$qP = QS_Y$ (8-335)

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0

$qP = QP'_C$ (8-336)

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$qP = QS_C$ (8-337)

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows.

– If all of the following conditions are true

– i is equal to 0

– j is equal to 0

– c relates to a luma residual block coded using Intra_16x16 prediction mode or c relates to a chroma residual block

the variable d_{00} is derived by

$d_{00} = c_{00}$ (8-338)

– Otherwise, the following applies.

– If qP is greater than or equal to 24, the scaled result is derived as follows

$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) << (qP / 6 - 4)$, with i, j = 0..3 except as noted above (8-339)

– Otherwise (qP is less than 24), the scaled result is derived as follows

$d_{ij} = (c * \text{LevelScale}(qP \% 6, i, j) + 2^{3-qp/6}) >> (4 - qP / 6)$, with i, j 0..3 except as noted above (8-340)

The bitstream shall not contain data that results in any element d_{ij} of d with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} – 1$, inclusive.

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

First, each (horizontal) row of scaled transform coefficients is transformed using a one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$e_{i0} = d_{i0} + d_{i2}$, with i = 0..3 (8-341)

$e_{i1} = d_{i0} – d_{i2}$, with i = 0..3 (8-342)

$e_{i2} = (d_{i1} >> 1) – d_{i3}$, with i = 0..3 (8-343)

$e_{i3} = d_{i1} + (d_{i3} >> 1)$, with i = 0..3 (8-344)

The bitstream shall not contain data that results in any element e_{ij} of e with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + \text{bitDepth})}$ to $2^{(7 + \text{bitDepth})} – 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$f_{i0} = e_{i0} + e_{i3}$, with i = 0..3 (8-345)

$f_{i1} = e_{i1} + e_{i2}$, with i = 0..3 (8-346)

$f_{i2} = e_{i1} – e_{i2}$, with i = 0..3 (8-347)

$f_{i3} = e_{i0} – e_{i3}$, with i = 0..3 (8-348)

The bitstream shall not contain data that results in any element f_{ij} of f with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

Then, each (vertical) column of the resulting matrix is transformed using the same one-dimensional inverse transform as follows.

A set of intermediate values is computed as follows.

$g_{0j} = f_{0j} + f_{2j}$, with j = 0..3 (8-349)

$g_{1j} = f_{0j} - f_{2j}$, with j = 0..3 (8-350)

$g_{2j} = (f_{1j} >> 1) - f_{3j}$, with j = 0..3 (8-351)

$g_{3j} = f_{1j} + (f_{3j} >> 1)$, with j = 0..3 (8-352)

The bitstream shall not contain data that results in any element g_{ij} of g with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 1$, inclusive.

Then, the transformed result is computed from these intermediate values as follows.

$h_{0j} = g_{0j} + g_{3j}$, with j = 0..3 (8-353)

$h_{1j} = g_{1j} + g_{2j}$, with j = 0..3 (8-354)

$h_{2j} = g_{1j} - g_{2j}$, with j = 0..3 (8-355)

$h_{3j} = g_{0j} - g_{3j}$, with j = 0..3 (8-356)

The bitstream shall not contain data that results in any element h_{ij} of h with i, j = 0..3 that exceeds the range of integer values from $-2^{(7 + bitDepth)}$ to $2^{(7 + bitDepth)} - 33$, inclusive.

After performing both the one-dimensional horizontal and the one-dimensional vertical inverse transforms to produce an array of transformed samples, the final constructed residual sample values is derived as

$r_{ij} = (h_{ij} + 2^5) >> 6$ with i, j = 0..3 (8-357)

Source: H.264 Standard at § 8.5.10.

119. Further, the Accused Products comprise a scanner for scanning the block of transform coefficient values in a given scanning order to produce a scanned array of coefficient values arranged according to the scanning order. This is shown for example in the reference encoder source code. (*See, e.g.*, ITU-T H.264 – Reference software for ITU-T H.264 advanced video coding (03/2005), at JM10.1a\lencod\src\lencod.c; JM10.1a\lencod\src\image.c; JM10.1a\lencod\src\cabac.c; and JM10.1a\lencod\src\macroblock.c.). On information and belief, the Accused Products operate consistently with the reference encoder. The reference encoder is not part of the H.264 standard.

120. Moreover, though the H.264 standard does not describe encoders, it does show that an H.264 bitstream comprises a scanned array of coefficient values arranged according to

the scanning order.

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

Source: H.264 Standard at § 3.

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.

Source: H.264 Standard at § 8.5.1.

8.5.5 Inverse scanning process for transform coefficients

Input to this process is a list of 16 values.

Output of this process is a variable c containing a two-dimensional array of 4x4 values. In the case of transform coefficients, these 4x4 values represent levels assigned to locations in the transform block. In the case of applying the inverse scanning process to a scaling list, the output variable c contains a two-dimensional array representing a 4x4 scaling matrix.

The inverse scanning process for transform coefficients maps the sequence of transform coefficient levels to the transform coefficient level positions. Table 8-13 specifies the two mappings: inverse zig-zag scan and inverse field scan. The inverse zig-zag scan is used for transform coefficients in frame macroblocks and the inverse field scan is used for transform coefficients in field macroblocks.

The inverse scanning process for scaling lists maps the sequence of scaling list entries to the positions in the corresponding scaling matrix. For this mapping, the inverse zig-zag scan is used.

Figure 8-8 illustrates the scans.

Source: H.264 Standard at § 8.5.5.

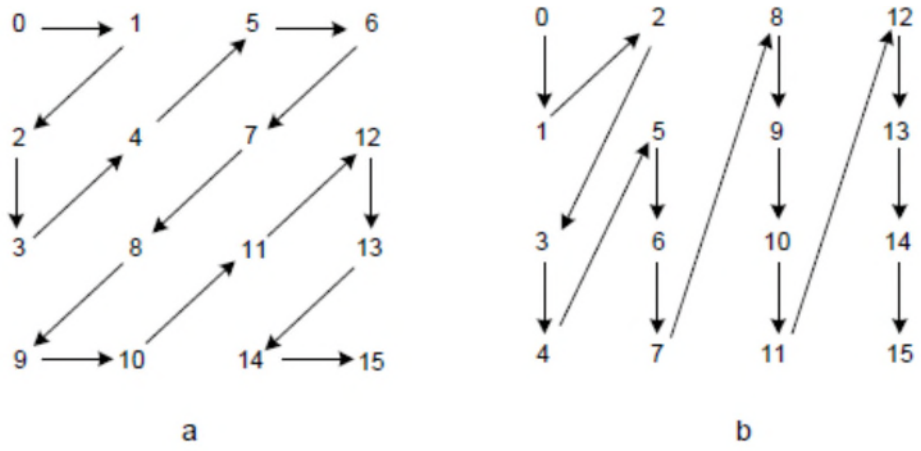


Figure 8-8 – 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative)

Source: H.264 Standard at § 8.5.5.

121. Further, the Accused Products comprise a run-level coder for representing the coefficient values in the scanned array by a plurality of number pairs, said number pairs having a first number and a second number. This is shown for example in the reference encoder source code. (*See, e.g.*, ITU-T H.264 – Reference software for ITU-T H.264 advanced video coding (03/2005), at JM10.1a\lencod\src\lencod.c; JM10.1a\lencod\src\image.c; JM10.1a\lencod\src\cabac.c; and JM10.1a\lencod\src\macroblock.c.). On information and belief, the Accused Products operate consistently with the reference encoder. The reference encoder is not part of the H.264 standard.

122. Moreover, though the H.264 standard does not describe encoders, it does show that an H.264 bitstream comprises a plurality of run level number pairs for representing the coefficient values in the scanned array, said number pairs having a first number and a second number.

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

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

Source: H.264 Standard at § 3.

7.3.5.3.2 Residual block CABAC syntax

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

Source: H.264 Standard at § 7.3.5.3.2.

significant_coeff_flag[i **]** specifies whether the transform coefficient level at scanning position i is non-zero as follows.

- If significant_coeff_flag[i] is equal to 0, the transform coefficient level at scanning position i is set equal to 0;
- Otherwise (significant_coeff_flag[i] is equal to 1), the transform coefficient level at scanning position i has a non-zero value.

. . .

coeff_abs_level_minus1[i **]** is the absolute value of a transform coefficient level minus 1. The value of coeff_abs_level_minus1 is constrained by the limits in subclause 8.5.

Source: H.264 Standard at § 7.4.5.3.2.

9.3 CABAC parsing process for slice data

This process is invoked when parsing syntax elements with descriptor ae(v) in subclauses 7.3.4 and 7.3.5 when entropy_coding_mode_flag is equal to 1.

Source: H.264 Standard at § 9.3.

123. Further, the Accused Products comprise a context-based coder for assigning the first numbers to one of a plurality of contexts representative of the first numbers and operative to assign the first number of a first number pair to a context at least partly in dependence on a first number of a second number pair. This is shown for example in the reference encoder source code. (*See, e.g.*, ITU-T H.264 – Reference software for ITU-T H.264 advanced video coding (03/2005), at JM10.1a\lencod\src\lencod.c; JM10.1a\lencod\src\image.c; JM10.1a\lencod\src\cabac.c; and JM10.1a\lencod\src\macroblock.c.). On information and belief, the Accused Products operate consistently with the reference encoder. The reference encoder is not part of the H.264 standard

124. Moreover, though the H.264 standard does not describe encoders, it does show that an H.264 bitstream comprises the first numbers which have been assigned to one of a plurality of contexts representative of the first numbers at least partly in dependence on a first number of a second number pair.

9.3.3.1.3 Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1

Inputs to this process are ctxIdxOffset and binIdx.

Output of this process is ctxIdxInc.

The assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1 as well as for coded_block_flag depends on categories of different blocks denoted by the variable ctxBlockCat. The specification of these block categories is given in Table 9-33.

…

Let numDecodAbsLevelEq1 denotes the accumulated number of decoded transform coefficient levels with absolute value equal to 1, and let numDecodAbsLevelGt1 denotes the accumulated number of decoded transform coefficient levels with absolute value greater than 1. Both numbers are related to the same transform coefficient block, where the current decoding process takes place. Then, for decoding of coeff_abs_level_minus1, ctxIdxInc for coeff_abs_level_minus1 is specified depending on binIdx as follows.

– If binIdx is equal to 0, ctxIdxInc is derived by

ctxIdxInc = ((numDecodAbsLevelGt1 != 0) ? 0: Min(4, 1 + numDecodAbsLevelEq1)) (9-17)

– Otherwise (binIdx is greater than 0), ctxIdxInc is derived by

ctxIdxInc = 5 + Min(4 – (ctxBlockCat == 3), numDecodAbsLevelGt1) (9-18)

Source: H.264 Standard at § 9.3.3.1.3.

125. Thus, as described above the Accused Products, including the Lenovo X1 Carbon laptop, infringe one or more claims of the ’701 Patent, including claim 18.

126. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’701 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

H. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the ’891 Patent.

127. The Accused Products infringe one or more claims of the ’891 patent, including for example, claim 33. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15” Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:

#Gameready

The Yoga 730 offers a NVIDIA® GTX™ 1050 discrete graphics with up to 4GB GPU RAM powered by NVIDIA Pascal™, the world's most advanced gaming GPU architecture, built for the latest multi-dimensional gaming environments. Savor a smooth, power-efficient performance that will take your gaming and cinematic experiences to the next level. Paired with the Yoga 730's optional 4K display, it's a whole new way to game.

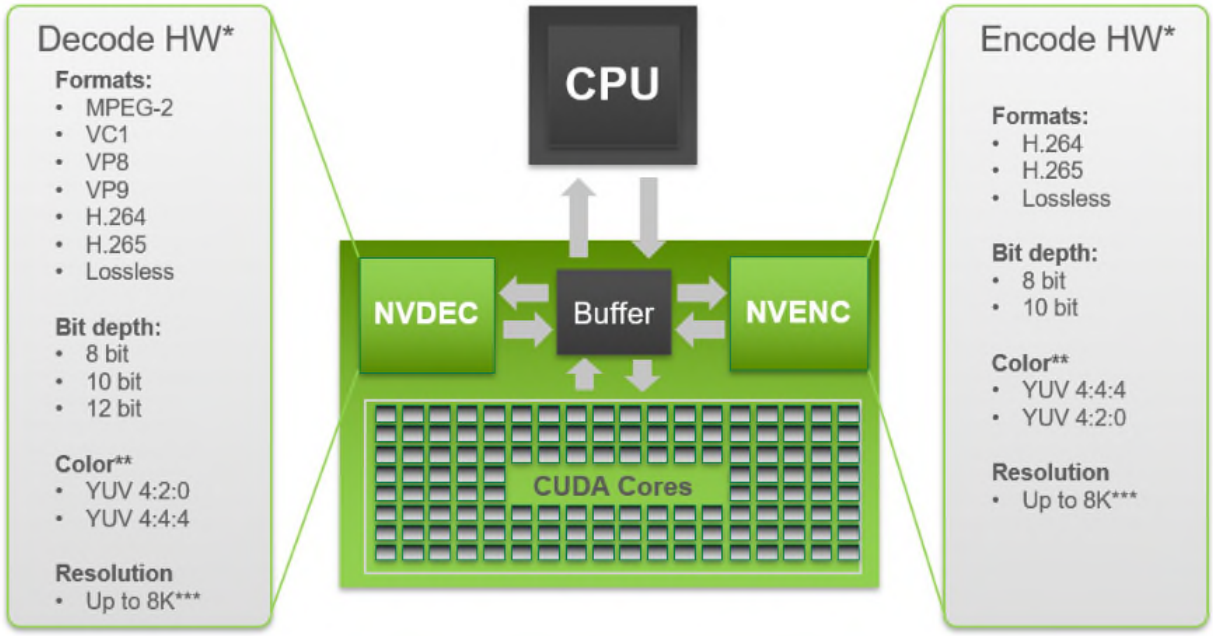
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

128. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

129. Thus, for example and as shown below, the Accused Products infringe claim 33 of the ’891 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products comprise mobile terminals that comprise an adaptive block boundary filter configured to perform an adaptive block boundary filtering operation on a block boundary formed between a first decoded image block on a first side of the block boundary and a second decoded image block on a second side of the block boundary. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3 and 8.7.

> **3 Definitions**
>
> For the purposes of this Recommendation | International Standard, the following definitions apply.
>
> **. . .**
>
> **3.15 block**: An MxN (M-column by N-row) array of samples **. . .**
>
> **. . .**
>
> **3.27 coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame* **. . .**
>
> **. . .**
>
> **3.37 decoded picture**: A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field* **. . .**
>
> **. . .**
>
> **3.41 decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.
>
> **. . .**
>
> **3.53 frame**: A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples. A *frame* consists of two *fields*, a *top field* and a *bottom field*.
>
> **3.54 frame macroblock**: A *macroblock* representing samples from the two *fields* of a *coded frame* **. . .**
>
> **. . .**
>
> **3.75 macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples **. . .**
>
> **. . .**
>
> **Source:** H.264 Standard at § 3.

8.7 Deblocking filter process

A conditional filtering process is applied to all NxN (where N = 4 or N = 8 for luma, and N = 4 for chroma) block edges of a picture, except edges at the boundary of the picture and any edges for which the deblocking filter process is disabled by disable_deblocking_filter_idc, as specified below. This filtering process is performed on a macroblock basis after the completion of the picture construction process prior to deblocking filter process (as specified in subclauses 8.5 and 8.6) for the entire decoded picture, with all macroblocks in a picture processed in order of increasing macroblock addresses.

. . .

The deblocking filter process is invoked for the luma and chroma components separately. For each macroblock and each component, vertical edges are filtered first, starting with the edge on the left-hand side of the macroblock proceeding through the edges towards the right-hand side of the macroblock in their geometrical order, and then horizontal edges are filtered, starting with the edge on the top of the macroblock proceeding through the edges towards the bottom of the macroblock in their geometrical order. Figure 8-10 shows edges of a macroblock which can be interpreted as luma or chroma edges.

. . .

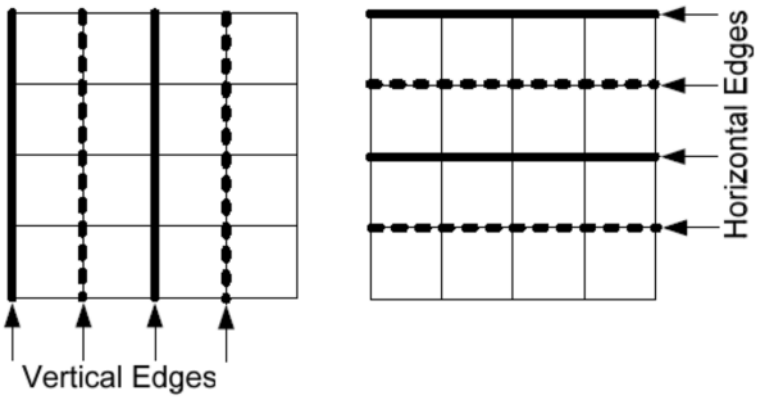


Figure 8-10 – Boundaries in a macroblock to be filtered

8.7.1 Filtering process for block edges

Inputs to this process are chromaEdgeFlag, the chroma component index iCbCr (when chromaEdgeFlag is equal to 1), verticalEdgeFlag, fieldModeFilteringFlag, and a set of nE sample locations (xEk, yEk), with k = 0..nE - 1, expressed relative to the upper left corner of the macroblock CurrMbAddr. The set of sample locations (xEk, yEk) represent the sample locations immediately to the right of a vertical edge (when verticalEdgeFlag is equal to 1) or immediately below a horizontal edge (when verticalEdgeFlag is equal to 0).

. . .

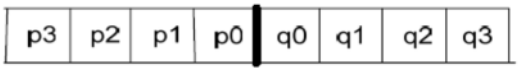


Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary

. . .

Source: H.264 Standard at § 8.7.1.

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values pi and qi with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag.

Outputs of this process are the filtered result sample values p'i and q'i with i in the range of 0..2.

. . .

Source: H.264 Standard at § 8.7.2.

130. Further, the adaptive boundary block filter in the Accused Products is arranged to perform an adaptive boundary block filtering operation where the first decoded image block was encoded using a first type of prediction encoding method and the second decoded image block was encoded using a second type of prediction encoding method. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 7.3.5 and 7.4.5.

7.3.5 Macroblock layer syntax

macroblock_layer( ) {	C	Descriptor
mb_type	2	ue(v) | ae(v)
if(mb_type == I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	2	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i]	2	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i]	2	u(v)

. . .

Source: H.264 Standard at § 7.3.5.

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of mb_type depend on the slice type.

Tables and semantics are specified for the various macroblock types for I, SI, P, SP, and B slices. Each table presents the value of mb_type, the name of mb_type, the number of macroblock partitions used (given by the NumMbPart(mb_type) function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the MbPartPredMode(mb_type, 0) function) and the prediction mode of the second partition (given by the MbPartPredMode(mb_type, 1) function).

. . .

Table 7-10 shows the allowed collective macroblock types for each slice_type.

. . .

Table 7-10 – Allowed collective macroblock types for slice_type

slice_type	allowed collective macroblock types
I (slice)	I (see Table 7-11) (macroblock types)
P (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)
B (slice)	B (see Table 7-14) and I (see Table 7-11) (macroblock types)
SI (slice)	SI (see Table 7-12) and I (see Table 7-11) (macroblock types)
SP (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)

. . .

Macroblock types that may be collectively referred to as I macroblock types are specified in Table 7-11.

. . .

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-33	Equation 7-33
0	I_NxN	1	Intra_8x8	na	Equation 7-33	Equation 7-33
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0
7	I_16x16_2_1_0	na	Intra_16x16	2	1	0
8	I_16x16_3_1_0	na	Intra_16x16	3	1	0
9	I_16x16_0_2_0	na	Intra_16x16	0	2	0
10	I_16x16_1_2_0	na	Intra_16x16	1	2	0
11	I_16x16_2_2_0	na	Intra_16x16	2	2	0
12	I_16x16_3_2_0	na	Intra_16x16	3	2	0
13	I_16x16_0_0_1	na	Intra_16x16	0	0	15
14	I_16x16_1_0_1	na	Intra_16x16	1	0	15
15	I_16x16_2_0_1	na	Intra_16x16	2	0	15
16	I_16x16_3_0_1	na	Intra_16x16	3	0	15
17	I_16x16_0_1_1	na	Intra_16x16	0	1	15
18	I_16x16_1_1_1	na	Intra_16x16	1	1	15
19	I_16x16_2_1_1	na	Intra_16x16	2	1	15
20	I_16x16_3_1_1	na	Intra_16x16	3	1	15
21	I_16x16_0_2_1	na	Intra_16x16	0	2	15
22	I_16x16_1_2_1	na	Intra_16x16	1	2	15
23	I_16x16_2_2_1	na	Intra_16x16	2	2	15
24	I_16x16_3_2_1	na	Intra_16x16	3	2	15
25	I_PCM	na	na	na	na	na

. . .

The following semantics are assigned to the macroblock types in Table 7-11.

I_NxN: A mnemonic name for mb_type equal to 0 with MbPartPredMode(mb_type, 0) equal to Intra_4x4 or Intra_8x8.

I_16x16_0_0_0, I_16x16_1_0_0, I_16x16_2_0_0, I_16x16_3_0_0, I_16x16_0_1_0, I_16x16_1_1_0, I_16x16_2_1_0, I_16x16_3_1_0, I_16x16_0_2_0, I_16x16_1_2_0, I_16x16_2_2_0, I_16x16_3_2_0, I_16x16_0_0_1, I_16x16_1_0_1, I_16x16_2_0_1, I_16x16_3_0_1, I_16x16_0_1_1, I_16x16_1_1_1, I_16x16_2_1_1, I_16x16_3_1_1, I_16x16_0_2_1, I_16x16_1_2_1, I_16x16_2_2_1, I_16x16_3_2_1: the macroblock is coded as an Intra_16x16 prediction mode macroblock.

. . .

Intra_4x4 specifies the macroblock prediction mode and specifies that the Intra_4x4 prediction process is invoked as specified in subclause 8.3.1. Intra_4x4 is an Intra macroblock prediction mode.

Intra_8x8 specifies the macroblock prediction mode and specifies that the Intra_8x8 prediction process is invoked as specified in subclause 8.3.2. Intra_8x8 is an Intra macroblock prediction mode.

Intra_16x16 specifies the macroblock prediction mode and specifies that the Intra_16x16 prediction process is invoked as specified in subclause 8.3.3. Intra_16x16 is an Intra macroblock prediction mode.

For a macroblock coded with mb_type equal to I_PCM, the Intra macroblock prediction mode shall be inferred.

. . .

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for P and SP slices are specified in Tables 7-13 and 7-11. mb_type values 0 to 4 are specified in Table 7-13 and mb_type values 5 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type.

. . .

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

The following semantics are assigned to the macroblock types in Table 7-13.

– P_L0_16x16: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples.

– P_L0_L0_MxN, with MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively.

– P_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).

– P_8x8ref0: has the same semantics as P_8x8 but no syntax element for the reference index (ref_idx_l0) is present in the bitstream and ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0 for all sub-macroblocks of the macroblock (with indices mbPartIdx equal to 0..3).

– P_Skip: no further data is present for the macroblock in the bitstream.

. . .

Source: H.264 Standard at § 7.4.5.

131. Further, the adaptive boundary block filter in the Accused Products is arranged to perform an adaptive boundary block filtering operation where the first decoded image block was encoded using a first type of prediction encoding method and the second decoded image block was encoded using a second type of prediction encoding method, where the first type of prediction encoding method and the second type of prediction encoding method are selected

from a group of prediction encoding methods comprising at least: intra coding, copy coding, motion-compensated prediction coding, and not-coded coding. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to tables 7-11 and 7-13 and §§ 3, 7.4.5, and 8.4.1.

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-33	Equation 7-33
0	I_NxN	1	Intra_8x8	na	Equation 7-33	Equation 7-33
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

Source: H.264 Standard at § 7.4.5.

3 Definitions

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

. . .

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

. . .

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

. . .

Source: H.264 Standard at § 3.

8.4.1 Derivation process for motion vector components and reference indices

. . .

– If mb_type is equal to P_Skip, the derivation process for luma motion vectors for skipped macroblocks in P and SP slices in subclause 8.4.1.1 is invoked with the output being the luma motion vectors mvL0 and reference indices refIdxL0, and predFlagL0 is set equal to 1

. . .

Source: H.264 Standard at § 8.4.1.

8.4.1.1 Derivation process for luma motion vectors for skipped macroblocks in P and SP slices

This process is invoked when mb_type is equal to P_Skip.

Outputs of this process are the motion vector mvL0 and the reference index refIdxL0.

The reference index refIdxL0 for a skipped macroblock is derived as follows.

refIdxL0 = 0. (8-170)

For the derivation of the motion vector mvL0 of a P_Skip macroblock type, the following applies.

– The process specified in subclause 8.4.1.3.2 is invoked with mbPartIdx set equal to 0, subMbPartIdx set equal to 0, currSubMbType set equal to "na", and listSuffixFlag set equal to 0 as input and the output is assigned to mbAddrA, mbAddrB, mvL0A, mvL0B, refIdxL0A, and refIdxL0B.

– The variable mvL0 is specified as follows.

– If any of the following conditions are true, both components of the motion vector mvL0 are set equal to 0.

– mbAddrA is not available

– mbAddrB is not available

– refIdxL0A is equal to 0 and both components of mvL0A are equal to 0

– refIdxL0B is equal to 0 and both components of mvL0B are equal to 0

– Otherwise, the derivation process for luma motion vector prediction as specified in subclause 8.4.1.3 is invoked with mbPartIdx = 0, subMbPartIdx = 0, refIdxL0, and currSubMbType = "na" as inputs and the output is assigned to mvL0.
NOTE – The output is directly assigned to mvL0, since the predictor is equal to the actual motion vector.

Source: H.264 Standard at § 8.4.1.1.

8.4.1.3 Derivation process for luma motion vector prediction

Inputs to this process are

- the macroblock partition index mbPartIdx,
- the sub-macroblock partition index subMbPartIdx,
- the reference index of the current partition refIdxLX (with X being 0 or 1),
- the variable currSubMbType.

Output of this process is the prediction mvpLX of the motion vector mvLX (with X being 0 or 1).

The derivation process for the neighbouring blocks for motion data in subclause 8.4.1.3.2 is invoked with mbPartIdx, subMbPartIdx, currSubMbType, and listSuffixFlag = X (with X being 0 or 1 for refIdxLX being refIdxL0 or refIdxL1, respectively) as the input and with mbAddrN\mbPartIdxN\subMbPartIdxN, reference indices refIdxLXN and the motion vectors mvLXN with N being replaced by A, B, or C as the output.

The derivation process for median luma motion vector prediction in subclause 8.4.1.3.1 is invoked with mbAddrN\mbPartIdxN\subMbPartIdxN, mvLXN, refIdxLXN with N being replaced by A, B, or C and refIdxLX as the input and mvpLX as the output, unless one of the following is true.

- MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 0, and refIdxLXB is equal to refIdxLX,

  mvpLX = mvLXB (8-200)

- MbPartWidth(mb_type) is equal to 16, MbPartHeight(mb_type) is equal to 8, mbPartIdx is equal to 1, and refIdxLXA is equal to refIdxLX,

  mvpLX = mvLXA (8-201)

- MbPartWidth(mb_type) is equal to 8, MbPartHeight(mb_type) is equal to 16, mbPartIdx is equal to 0, and refIdxLXA is equal to refIdxLX,

  mvpLX = mvLXA (8-202)

Source: H.264 Standard at § 8.4.1.3.

132. Further, the adaptive boundary block filter in the Accused Products is configured to examine the type of the first prediction encoding method and the type of the second prediction encoding method. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 8.7.2.1.

8.7.2.1 Derivation process for the luma content dependent boundary filtering strength

Inputs to this process are the input sample values p0 and q0 of a single set of samples across an edge that is to be filtered and verticalEdgeFlag.

Output of this process is the variable bS.

Let the variable mixedModeEdgeFlag be derived as follows.

– If MbaffFrameFlag is equal to 1 and the samples p0 and q0 are in different macroblock pairs, one of which is a field macroblock pair and the other is a frame macroblock pair, mixedModeEdgeFlag is set equal to 1

– Otherwise, mixedModeEdgeFlag is set equal to 0.

The variable bS is derived as follows.

– If the block edge is also a macroblock edge and any of the following conditions are true, a value of bS equal to 4 is the output:

– the samples p0 and q0 are both in frame macroblocks and either or both of the samples p0 or q0 is in a macroblock coded using an Intra macroblock prediction mode

– the samples p0 and q0 are both in frame macroblocks and either or both of the samples p0 or q0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p0 or q0 is in a macroblock coded using an Intra macroblock prediction mode

– MbaffFrameFlag is equal to 1 or field_pic_flag is equal to 1, and verticalEdgeFlag is equal to 1, and either or both of the samples p0 or q0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– Otherwise, if any of the following conditions are true, a value of bS equal to 3 is the output:

– mixedModeEdgeFlag is equal to 0 and either or both of the samples p0 or q0 is in a macroblock coded using an Intra macroblock prediction mode

– mixedModeEdgeFlag is equal to 0 and either or both of the samples p0 or q0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p0 or q0 is in a macroblock coded using an Intra macroblock prediction mode

– mixedModeEdgeFlag is equal to 1, verticalEdgeFlag is equal to 0, and either or both of the samples p0 or q0 is in a macroblock that is in a slice with slice_type equal to SP or SI

– Otherwise, if the following condition is true, a value of bS equal to 2 is the output:

– the luma block containing sample p0 or the luma block containing sample q0 contains non-zero transform coefficient levels

– Otherwise, if any of the following conditions are true, a value of bS equal to 1 is the output:

– mixedModeEdgeFlag is equal to 1

– mixedModeEdgeFlag is equal to 0 and for the prediction of the macroblock/sub-macroblock partition containing the sample p0 different reference pictures or a different number of motion vectors are used than for the prediction of the macroblock/sub-macroblock partition containing the sample q0.
NOTE 1 – The determination of whether the reference pictures used for the two macroblock/sub-macroblock partitions are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether or not the index position within a reference picture list is different or not.

– mixedModeEdgeFlag is equal to 0 and one motion vector is used to predict the macroblock/sub-macroblock partition containing the sample p0 and one motion vector is

used to predict the macroblock/sub-macroblock partition containing the sample q0 and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 4 in units of quarter luma frame samples.

– mixedModeEdgeFlag is equal to 0 and two motion vectors and two different reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample p0 and two motion vectors for the same two reference pictures are used to predict the macroblock/sub-macroblock partition containing the sample q0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two macroblock/sub-macroblock partitions for the same reference picture is greater than or equal to 4 in units of quarter luma frame samples.

– mixedModeEdgeFlag is equal to 0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample p0 and two motion vectors for the same reference picture are used to predict the macroblock/sub-macroblock partition containing the sample q0 and both of the following conditions are true:

– The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two macroblock/sub-macroblock partitions is greater than or equal to 4 in units of quarter luma frame samples.

– The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample p0 and the list 1 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q0 is greater than or equal to 4 in units of quarter luma frame samples or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the macroblock/submacroblock partition containing the sample p0 and list 0 motion vector used in the prediction of the macroblock/sub-macroblock partition containing the sample q0 is greater than or equal to 4 in units of quarter luma frame samples.
NOTE 2 – A vertical difference of 4 in units of quarter luma frame samples is a difference of 2 in units of quarter luma field samples

– Otherwise, a value of bS equal to 0 is the output.

Source: H.264 Standard at § 8.7.2.1.

133. Further, the adaptive boundary block filter in the Accused Products is configured to determine a first number of pixels to be examined on the first side of the block boundary and a second number of pixels to be examined on the second side of the block boundary, as a parameter of the adaptive block boundary filtering operation, based on the types of the first and second prediction encoding method. As shown below, this structure and functionality is

described in the H.264 Standard, including but not limited to § 8.7.2.

8.7.2 Filtering process for a set of samples across a horizontal or vertical block edge

Inputs to this process are the input sample values pi and qi with i in the range of 0..3 of a single set of samples across an edge that is to be filtered, chromaEdgeFlag, verticalEdgeFlag, and fieldModeFilteringFlag.

Outputs of this process are the filtered result sample values p'i and q'i with i in the range of 0..2.

The content dependent boundary filtering strength variable bS is derived as follows.

– If chromaEdgeFlag is equal to 0, the derivation process for the content dependent boundary filtering strength specified in subclause 8.7.2.1 is invoked with p0, q0, and verticalEdgeFlag as input, and the output is assigned to bS.

– Otherwise (chromaEdgeFlag is equal to 1), the bS used for filtering a set of samples of a horizontal or vertical chroma edge is set equal to the value of bS for filtering the set of samples of a horizontal or vertical luma edge, respectively, that contains the luma sample at location (SubWidthC * x, SubHeightC * y) inside the luma array of the same field, where (x, y) is the location of the chroma sample q0 inside the chroma array for that field.

The process specified in subclause 8.7.2.2 is invoked with p0, q0, p1, q1, chromaEdgeFlag, and bS as input, and the output is assigned to filterSamplesFlag, indexA, α, and β.

Depending on the variable filterSamplesFlag, the following applies.

– If filterSamplesFlag is equal to 1, the following applies.

– If bS is less than 4, the process specified in subclause 8.7.2.3 is invoked with pi and qi (i = 0..2), chromaEdgeFlag, bS, β, and indexA given as input, and the output is assigned to p'i and q'i (i = 0..2).

– Otherwise (bS is equal to 4), the process specified in subclause 8.7.2.4 is invoked with pi and qi (i = 0..3), chromaEdgeFlag, α, and β given as input, and the output is assigned to p'i and q'i (i = 0..2).

– Otherwise (filterSamplesFlag is equal to 0), the filtered result samples p'i and q'i (i = 0..2) are replaced by the corresponding input samples pi and qi:

for i = 0..2, p'i = pi (8-460)

for i = 0..2, q'i = qi (8-461)

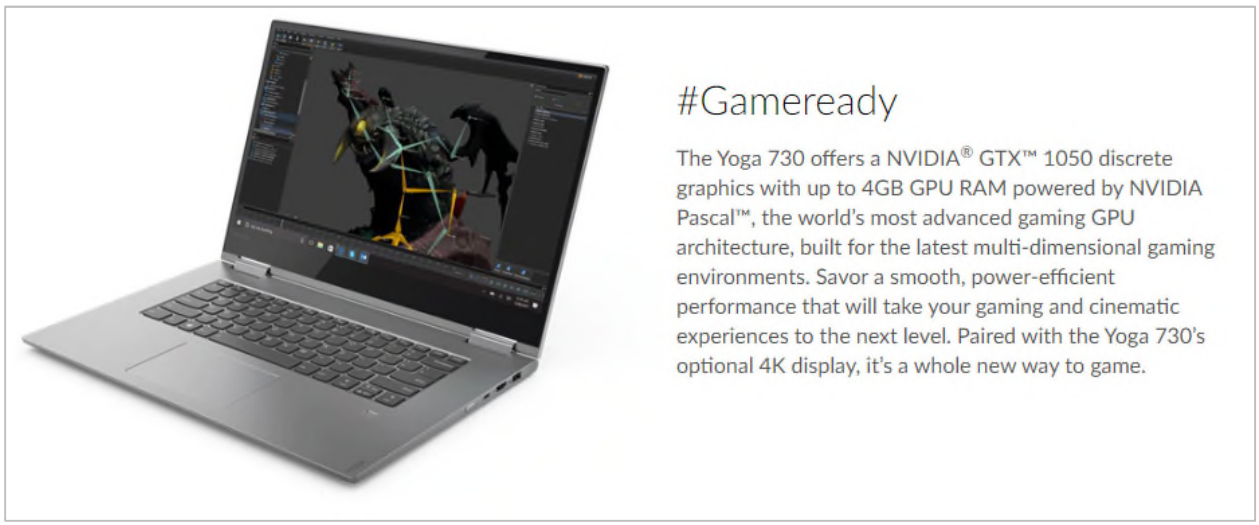
Source: H.264 Standard at § 8.7.2.

134. Thus, as described above the Accused Products, including the Lenovo yoga 730 15” Platinum laptop, infringe one or more claims of the ’891 Patent, including claim 33.

135. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the ’891 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

I. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the ’818 Patent.

136. The Accused Products infringe one or more claims of the ’818 patent, including for example, claim 6. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15” Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:



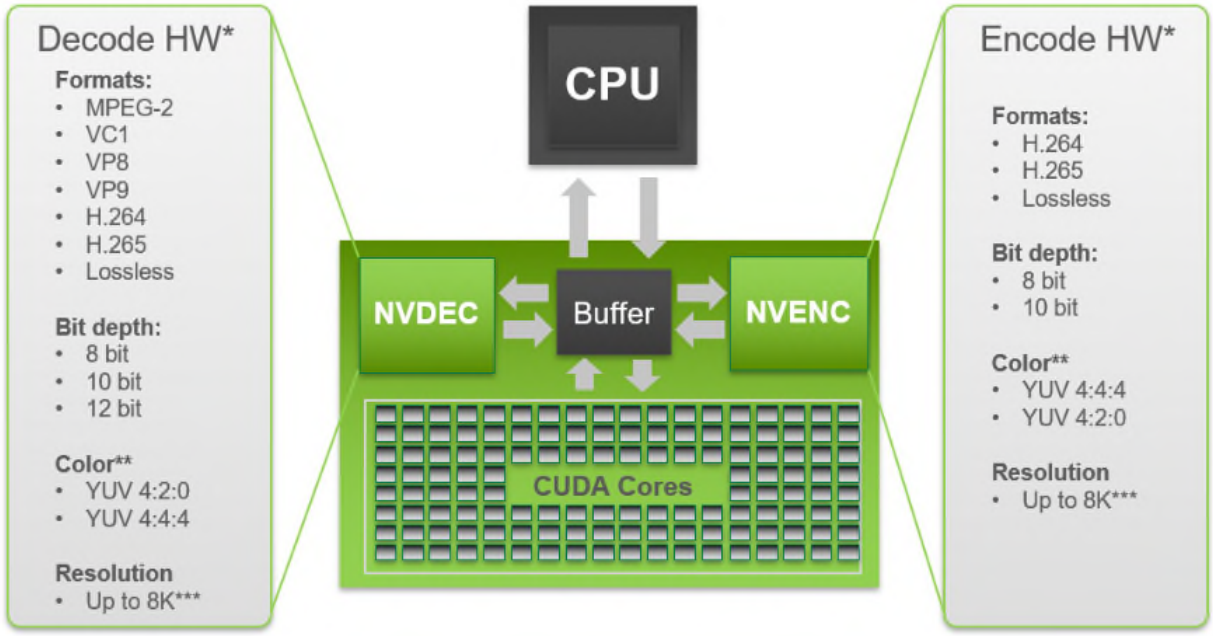
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

137. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

138. Thus, for example and as shown below, the Accused Products infringe claim 6 of the ’818 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products perform a method of decoding sequences of pictures from a bitstream, wherein parameters are defined in a parameter set and each picture comprises information of one or more slices. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 6.3 and 7.4.1.

> **3 Definitions**
> For the purposes of this Recommendation | International Standard, the following definitions apply.
> . . .
> **3.1 access unit**: A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures* or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.
> . . .
> **3.12 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*.
> . . .
> **3.27 coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.
> . . .
> **3.30 coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.
> . . .
> **3.62 instantaneous decoding refresh (IDR) picture**: A *coded picture* in which all *slices* are *I* or *SI slices* **. . .**
> . . .
> **3.75 macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.
> . . .
> **3.103 picture parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the pic_parameter_set_id *syntax element* found in each *slice header*.
> . . .
> **3.131 sequence parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a seq_parameter_set_id *syntax element* found in the *picture parameter set* referred to by the pic_parameter_set_id *syntax element* found in each *slice header*.
> . . .
> **3.136 slice**: An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group* . . .
> . . .

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

. . .

Source: H.264 Standard at § 3.

6.3 Spatial subdivision of pictures and slices

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

. . .

Source: H.264 Standard at § 6.3.

7.4.1 NAL unit semantics

. . .

Table 7-1 – NAL unit type codes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C
0	Unspecified	
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp( )	2, 3, 4
2	Coded slice data partition A slice_data_partition_a_layer_rbsp( )	2
3	Coded slice data partition B slice_data_partition_b_layer_rbsp( )	3
4	Coded slice data partition C slice_data_partition_c_layer_rbsp( )	4
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp( )	2, 3
6	Supplemental enhancement information (SEI) sei_rbsp( )	5
7	Sequence parameter set seq_parameter_set_rbsp( )	0
8	Picture parameter set pic_parameter_set_rbsp( )	1
9	Access unit delimiter access_unit_delimiter_rbsp( )	6
10	End of sequence end_of_seq_rbsp( )	7
11	End of stream end_of_stream_rbsp( )	8
12	Filler data filler_data_rbsp( )	9
13..23	Reserved	
24..31	Unspecified	

Source: H.264 Standard at § 7.4.1.

139. Further, the Accused Products recognize, in a decoder, a sequence parameter set and form at least one sequence parameter pertaining to a sequence using the parameter set. As

shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.131, 7.3.2.1, 7.4.2.1, and 7.4.1.2.1.

> **3 Definitions**
> For the purposes of this Recommendation | International Standard, the following definitions apply.
> . . .
> **3.131 sequence parameter set**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a seq_parameter_set_id *syntax element* found in the *picture parameter set* referred to by the pic_parameter_set_id *syntax element* found in each *slice header*.

Source: H.264 Standard at § 3.

7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 100 || profile_idc == 110 || profile_idc == 122 || profile_idc == 144) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc == 3)		
residual_colour_transform_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < 8; i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i – 6], 64, UseDefaultScalingMatrix8x8Flag[i – 6])	0	
}		
}		
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)

frame_mbs_only_flag	0	u(1)
if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters( )	0	
rbsp_trailing_bits( )	0	
}		

Source: H.264 Standard at § 7.3.2.1.

140. Further, the Accused Products recognize, in a decoder, a picture parameter set and form at least one first picture parameter value pertaining to a picture using the parameter set. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.103, 7.3.2.2, 7.4.2.2, and 7.4.1.2.1.

3 Definitions

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

. . .

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

Source: H.264 Standard at § 3.

7.3.2.2 Picture Parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type == 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup]	1	ue(v)

else if(slice_group_map_type == 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup]	1	ue(v)
bottom_right[iGroup]	1	ue(v)
}		
else if(slice_group_map_type == 3 || slice_group_map_type == 4 || slice_group_map_type == 5) {		
slice_group_change_direction_flag	1	u(1)
slice_group_change_rate_minus1	1	ue(v)
} else if(slice_group_map_type == 6) {		
pic_size_in_map_units_minus1	1	ue(v)
for(i = 0; i <= pic_size_in_map_units_minus1; i++)		
slice_group_id[i]	1	u(v)
}		
}		
num_ref_idx_l0_active_minus1	1	ue(v)
num_ref_idx_l1_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)
if(more_rbsp_data( )) {		
transform_8x8_mode_flag	1	u(1)
pic_scaling_matrix_present_flag	1	u(1)
if(pic_scaling_matrix_present_flag)		
for(i = 0; i < 6 + 2* transform_8x8_mode_flag; i++) {		
pic_scaling_list_present_flag[i]	1	u(1)
if(pic_scaling_list_present_flag[i])		
if(i < 6)		

scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	1	
else		
scaling_list(ScalingList8x8[i – 6], 64, UseDefaultScalingMatrix8x8Flag[i – 6])	1	
}		
second_chroma_qp_index_offset	1	se(v)
}		
rbsp_trailing_bits( )	1	
}		

Source: H.264 Standard at § 7.3.2.2.

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

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

. . .

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

. . .

Source: H.264 Standard at § 7.4.1.2.1.

141. Further, the Accused Products form, in a decoder, at least one second picture parameter value using information of a slice header, the at least one second picture parameter value remaining unchanged at least in all slice headers of one picture. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.140, 7.3.3 and 7.4.3.

7.3.3 Slice header syntax

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

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

Source: H.264 Standard at § 7.3.3.

3 Definitions

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

. . .

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

. . .

Source: H.264 Standard at § 3.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements pic_parameter_set_id, frame_num, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], sp_for_switch_flag, and slice_group_change_cycle shall be the same in all slice headers of a coded picture.

Source: H.264 Standard at § 7.4.3.

142. Further, the Accused Products use, in the decoder, the at least one second picture parameter value in decoding. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.140, 7.3.3 and 7.4.3.

7.3.3 Slice header syntax

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type == 5)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type == 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type == 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0 **]**	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1 **]**	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type == B)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type == P || slice_type == SP || slice_type == B) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type == B)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering( )	2	
if((weighted_pred_flag && (slice_type == P || slice_type == SP)) || (weighted_bipred_idc == 1 && slice_type == B))		
pred_weight_table( )	2	
if(nal_ref_idc != 0)		
dec_ref_pic_marking( )	2	
if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)		
cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type == SP || slice_type == SI) {		
if(slice_type == SP)		
sp_for_switch_flag	2	u(1)
slice_qs_delta	2	se(v)
}		
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
}		

Source: H.264 Standard at § 7.3.3.

3 Definitions

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

. . .

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

. . .

Source: H.264 Standard at § 3.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements pic_parameter_set_id, frame_num, field_pic_flag, bottom_field_flag, idr_pic_id, pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt[0], delta_pic_order_cnt[1], sp_for_switch_flag, and slice_group_change_cycle shall be the same in all slice headers of a coded picture.

Source: H.264 Standard at § 7.4.3.

143. Thus, as described above the Accused Products, including the Lenovo Yoga 730 15" Platinum laptop, infringe one or more claims of the '818 Patent, including claim 6.

144. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the '818 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference

J. Lenovo Makes, Imports, Uses, Sells, and/or Offers for Sale Products and Services that Infringe the '125 Patent.

145. The Accused Products infringe one or more claims of the '125 patent, including for example, claim 1. Each of the Accused Products is compliant with the H.264 Standard. For example, as shown below, the Lenovo Yoga 730 15" Platinum laptop (Part No. 81CU000TUS) supports the H.264 video format. Lenovo advertises that this product uses the NVIDIA GeForce GTX 1050 GPU:

#Gameready

The Yoga 730 offers a NVIDIA® GTX™ 1050 discrete graphics with up to 4GB GPU RAM powered by NVIDIA Pascal™, the world's most advanced gaming GPU architecture, built for the latest multi-dimensional gaming environments. Savor a smooth, power-efficient performance that will take your gaming and cinematic experiences to the next level. Paired with the Yoga 730's optional 4K display, it's a whole new way to game.

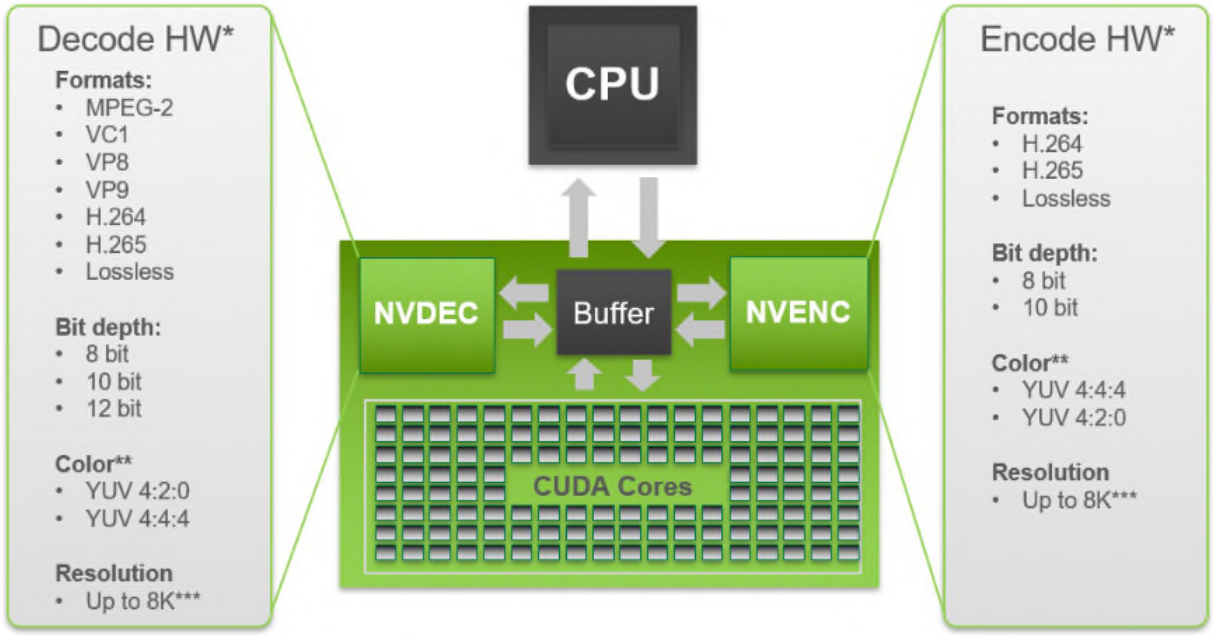
Graphics	• NVIDIA® GeForce® GTX 1050 2 GB • NVIDIA® GeForce® GTX 1050 4 GB

Source: https://www.lenovo.com/us/en/laptops/yoga/700-series/Yoga-730-15/p/88YG7000965.

146. The NVIDIA GeForce GTX 1050 GPU supports the H.264 Standard:

NVIDIA GPUs contain one or more **hardware-based decoder and encoder(s) (separate from the CUDA cores)** which provides fully-accelerated hardware-based video decoding and encoding for several popular codecs. With decoding/encoding offloaded, the graphics engine and the CPU are free for other operations.

GPU hardware accelerator engines for video decoding (referred to as **NVDEC**) and video encoding (referred to as **NVENC**) support faster than real-time video processing which makes them suitable to be used for transcoding applications, in addition to video playback.



GPU	*H.265 (HEVC) 4:4:4		H.265 (HEVC) 4:2:0		H.264 (AVCHD) 4:2:0	
	MAX Color	MAX Res.	MAX Color	MAX Res.	MAX Color	MAX Res.
Kepler	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (1st Gen)*	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (2nd Gen)	N/A	N/A	N/A	N/A	8-bit	4096 x 4096
Maxwell (GM206)	N/A	N/A	10-bit	4096 x 2304	8-bit	4096 x 4096
Pascal	N/A	N/A	12-bit	8192 x 8192**	8-bit	4096 x 4096

Source: https://developer.nvidia.com/nvidia-video-codec-sdk.

147. Thus, for example and as shown below, the Accused Products infringe claim 1 of the ’125 Patent by virtue of their compatibility with and practice of the H.264 Standard. For example, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to § 3.

> **3 Definitions**
> For the purposes of this Recommendation | International Standard, the following definitions apply.
> **. . .**
> **3.14 bitstream**: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences* **. . .**
> **. . .**
> **3.27 coded picture**: A *coded representation* of a *picture* **. . .**
> **. . .**
> **3.30 coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.
> **. . .**
> **3.37 decoded picture**: A *decoded picture* is derived by decoding a *coded picture* **. . .**
> **. . .**
> **3.41 decoding process**: The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.
> **. . .**
> **3.47 encoding process**: A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.

Source: H.264 Standard at § 3.

148. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence, the digital video sequence comprising a number of frames, each frame of said sequence comprising an array of pixels divided into a plurality of blocks, each block comprising a certain number of said pixels. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3 and 6.3.

> **3 Definitions**

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

. . .

3.14 bitstream: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences* **. . .**

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

. . .

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

. . .

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

. . .

3.53 frame: A *frame* contains an array of *luma* samples and two corresponding arrays of *chroma* samples **. . .**

3.54 frame macroblock: A *macroblock* representing samples from the two *fields* of a *coded frame* **. . .**

. . .

3.75 macroblock: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples **. . .**

Source: H.264 Standard at § 3.

6.3 Spatial subdivision of pictures and slices

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

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

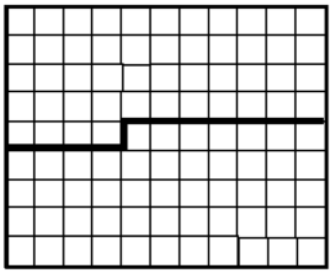


Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices

Source: H.264 Standard at § 6.3.

149. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video

sequence, frames of the digital video sequence encoded by applying motion compensated prediction to blocks of pixels for producing corresponding blocks of prediction error values. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3, 0.6, 7.3.5.1, 7.3.5.3, 7.4.5.1, and 8.4.

3 Definitions

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

. . .

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

. . .

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

. . .

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

. . .

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

. . .

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

. . .

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

Source: H.264 Standard at § 3.

0.6 Overview of the design characteristics

. . .

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality **. . .** A number of techniques may be used to achieve highly efficient compression **. . .** Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures **. . .** The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantized, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors **. . .** are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

. . .

Source: H.264 Standard at § 0.6.

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	C	Descriptor

. . .

} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 || mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1[mbPartIdx **]**	2	te(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode (mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1[mbPartIdx **][** 0 **][** compIdx **]**	2	se(v) | ae(v)
}		
}		

7.4.5.1 Macroblock prediction semantics

. . .

ref_idx_l0[mbPartIdx **]** when present, specifies the index in reference picture list 0 of the reference picture to be used for prediction.

. . .

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

. . .

mvd_l0[mbPartIdx **][** 0 **][** compIdx **]** specifies the difference between a vector component to be used and its prediction . . .

. . .

mvd_l1[mbPartIdx **][** 0 **][** compIdx **]** has the same semantics as mvd_l0, with l0 and L0 replaced by l1 and L1, respectively.

7.3.5.3 Residual data syntax

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

Source: H.264 Standard at § 7.

8.4 Inter prediction process

This process is invoked when decoding P and B macroblock types.

Outputs of this process are Inter prediction samples for the current macroblock that are a 16x16 array $pred_L$ of luma samples and when chroma_format_idc is not equal to 0 (monochrome) two 8x8 arrays $pred_{Cr}$ and $pred_{Cb}$ of chroma samples, one for each of the chroma components Cb and Cr.

Source: H.264 Standard at § 8.4.

150. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence, frames of the digital video sequence encoded by applying a transform coding technique to said blocks of prediction error values to produce sets of transform coefficient values representative of said blocks of prediction error values. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3, 0.6, and 8.5.

3 Definitions

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

. . .

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

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

. . .

Source: H.264 Standard at § 3.

0.6 Overview of the design characteristics

. . .

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality **. . .** A number of techniques may be used to achieve highly efficient compression **. . .** Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures **. . .** The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantized, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors **. . .** are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

. . .

Source: H.264 Standard at § 0.6.

8.5 Transform coefficient decoding process and picture construction process prior to deblocking filter process

. . .

This subclause specifies transform coefficient decoding and picture construction prior to the deblocking filter process.

. . .

8.5.1 Specification of transform decoding process for 4x4 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 0.

When the current macroblock prediction mode is not equal to Intra_16x16, the variable LumaLevel contains the levels for the luma transform coefficients. For a 4x4 luma block indexed by luma4x4BlkIdx = 0..15, the following ordered steps are specified.

1. The inverse transform coefficient scanning process as described in subclause 8.5.5 is invoked with LumaLevel[luma4x4BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 4x4 blocks as specified in subclause 8.5.10 is invoked with c as the input and r as the output.

. . .

8.5.3 Specification of transform decoding process for 8x8 luma residual blocks

This specification applies when transform_size_8x8_flag is equal to 1.

The variable LumaLevel8x8[luma8x8BlkIdx] with luma8x8BlkIdx = 0..3 contains the levels for the luma transform coefficients for the luma 8x8 block with index luma8x8BlkIdx.

For an 8x8 luma block indexed by luma8x8BlkIdx = 0..3, the following ordered steps are specified.

1. The inverse scanning process for 8x8 luma transform coefficients as described in subclause 8.5.6 is invoked with LumaLevel8x8[luma8x8BlkIdx] as the input and the two-dimensional array c as the output.

2. The scaling and transformation process for residual 8x8 blocks as specified in subclause 8.5.11 is invoked with c as the input and r as the output.

. . .

8.5.10 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

. . .

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

. . .

8.5.11 Scaling and transformation process for residual 8x8 luma blocks

Input to this process is an 8x8 array c with elements c_{ij} which is an array relating to an 8x8 residual block of the luma component.

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

. . .

The transform process shall convert the block of scaled transform coefficients to a block of output samples in a manner mathematically equivalent to the following.

. . .

Source: H.264 Standard at § 8.5.

151. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence, frames of the digital video sequence encoded by applying a level of quantization to said sets of transform coefficient values to yield sets of quantized transform coefficient values representative of said blocks of prediction error values. As shown below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3 and 8.5.

3 Definitions

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

. . .

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

. . .

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

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

Source: H.264 Standard at § 3.

8.5.10 Scaling and transformation process for residual 4x4 blocks

Input to this process is a 4x4 array c with elements c_{ij} which is either an array relating to a residual block of the luma component or an array relating to a residual block of a chroma component.

Outputs of this process are residual sample values as 4x4 array r with elements r_{ij}.

. . .

The variable qP is derived as follows.

– If the input array c relates to a luma residual block and sMbFlag is equal to 0

$qP = QP'_Y$ (8-334)

– Otherwise, if the input array c relates to a luma residual block and sMbFlag is equal to 1

$qP = QS_Y$ (8-335)

– Otherwise, if the input array c relates to a chroma residual block and sMbFlag is equal to 0

$qP = QP'_C$ (8-336)

– Otherwise (the input array c relates to a chroma residual block and sMbFlag is equal to 1),

$qP = QS_C$ (8-337)

Scaling of 4x4 block transform coefficient levels c_{ij} proceeds as follows.

– If all of the following conditions are true

– i is equal to 0

– j is equal to 0

– c relates to a luma residual block coded using Intra_16x16 prediction mode or c relates to a chroma residual block

the variable d_{00} is derived by

$d_{00} = c_{00}$ (8-338)

– Otherwise, the following applies.

– If qP is greater than or equal to 24, the scaled result is derived as follows

$d_{ij} = (c_{ij} * \text{LevelScale}(qP \% 6, i, j)) << (qP / 6 – 4)$, with i,j = 0..3 except as noted above (8-339)

– Otherwise (qP is less than 24), the scaled result is derived as follows

$d (c * \text{LevelScale}(qP \% 6, i, j)\ 2^{3-qP/6}\ ) >> (4\ qP / 6)$, with i, j 0..3 except as noted above (8-340)

. . .

8.5.11 Scaling and transformation process for residual 8x8 luma blocks

Input to this process is an 8x8 array c with elements c_{ij} which is an array relating to an 8x8 residual block of the luma component.

Outputs of this process are residual sample values as 8x8 array r with elements r_{ij}.

. . .

The scaling process for 8x8 block transform coefficient levels c_{ij} proceeds as follows.

– If QP'_Y is greater than or equal to 36, the scaled result is derived as

$d_{ij} = (c_{ij} * \text{LevelScale8x8}(QP'_Y \% 6, i, j)) << (QP'_Y / 6 – 6)$,

with i,j = 0..7 (8-359)

– Otherwise (QP'_Y is less than 36), the scaled result is derived as

$d_{ij} = (c_{ij} * \text{LevelScale8x8}(QP'_Y \% 6, i, j)) + 2^{5-QP'Y/6}\ ) >> (6 – QP'_Y /6)$, with i,j = 0..7 (8-360)

. . .

Source: H.264 Standard at § 8.5.

152. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence, frames of the digital video sequence encoded by applying a level of quantization to said sets of transform coefficient values to yield sets of quantized transform coefficient values representative of said blocks of prediction error values, wherein an indication of said level of

quantization is provided in the encoded bit-stream including the encoded digital video sequence. As shown in the limitation above and the evidence below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3, 8.5, 7.3.2.2, 7.4.1.2.1, 7.4.2.2, 7.3.3, 7.4.3, 7.3.5, and 7.4.5.

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)

. . .

weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)

. . .

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

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data . . .

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 . . .

. . .

7.4.2.2 Picture parameter set RBSP semantics

. . .

pic_init_qp_minus26 specifies the initial value minus 26 of $SliceQP_Y$ for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value of pic_init_qp_minus26 shall be in the range of $-(26 + QpBdOffset_Y)$ to +25, inclusive.

7.3.3 Slice header syntax

. . .

slice_header( ) {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
frame_num	2	u(v)

. . .

cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type == SP || slice_type == SI) {		
if(slice_type == SP)		
sp_for_switch_flag	2	u(1)

. . .

7.4.3 Slice header semantics

. . .

slice_qp_delta specifies the initial value of QP_Y to be used for all the macroblocks in the slice until modified by the value of mb_qp_delta in the macroblock layer. The initial QP_Y quantisation parameter for the slice is computed as:

$$SliceQP_Y = 26 + pic_init_qp_minus26 + slice_qp_delta \quad (7\text{-}27)$$

. . .

7.3.5 Macroblock layer syntax

. . .

macroblock_layer( ) {	C	Descriptor
mb_type	2	ue(v) | ae(v)
if(mb_type == I_PCM) {		
while(!byte_aligned( ))		
pcm_alignment_zero_bit	2	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i]	2	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i]	2	u(v)

. . .

if(CodedBlockPatternLuma > 0 || CodedBlockPatternChroma > 0 || MbPartPredMode(mb_type, 0) == Intra_16x16) {		
mb_qp_delta	2	se(v) | ae(v)
residual( )	3 | 4	
}		
}		
}		

7.4.5 Macroblock layer semantics

. . .

mb_qp_delta can change the value of QP_Y in the macroblock layer. The decoded value of mb_qp_delta shall be in the range of $-(26 + QpBdOffset_Y / 2)$ to $+(25 + QpBdOffset_Y / 2)$, inclusive. mb_qp_delta shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = ((QP_{Y,PREV} + mb_qp_delta + 52 + 2 * QpBdOffset_Y) \% (52 + QpBdOffset_Y)) - QpBdOffset_Y \quad (7\text{-}34)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y, of the previous macroblock in decoding order in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ derived in Equation 7-27 at the start of each slice.

The value of QP'_Y is derived as

$QP'_Y = QP_Y + QpBdOffset_Y$ (7-35)

Source: H.264 Standard at § 7.

153. Further, the Accused Products perform a method of decoding an encoded digital video sequence for use in a video decoding application to produce a decoded digital video sequence, said decoding method comprising selecting a default level of inverse quantization for use in decoding of the encoded digital video sequence to inverse quantize the sets of quantized transform coefficient values, said default level selected based on the indication of said level of quantization provided in the encoded bit-stream. As shown in the limitation above and the evidence below, this structure and functionality is described in the H.264 Standard, including but not limited to §§ 3, 7.3.2.2, and 7.4.2.2.

3 Definitions

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

. . .

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

. . .

Source: H.264 Standard at § 3.

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)

. . .

weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)

. . .

7.4.2.2 Picture parameter set RBSP semantics

. . .

pic_init_qp_minus26 specifies the initial value minus 26 of SliceQP$_Y$ for each slice. The initial value is modified at the slice layer when a non-zero value of slice_qp_delta is decoded, and is modified further when a non-zero value of mb_qp_delta is decoded at the macroblock layer. The value of pic_init_qp_minus26 shall be in the range of -(26 + QpBdOffset$_Y$) to +25, inclusive.

. . .

Source: H.264 Standard at § 7.

154. Thus, as described above the Accused Products, including the Lenovo Yoga 730 15" Platinum laptop, infringe one or more claims of the '125 Patent, including claim 6.

155. Lenovo encourages its users to use the Accused Products in a manner that infringes claims of the '125 Patent. In addition, Lenovo advertises the ability to stream video on its products and promotes that use. See ¶¶ 71-72 above, incorporated by reference.

COUNT I: PATENT INFRINGEMENT OF THE '808 PATENT

156. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

157. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the '808 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the '808 Patent.

158. The Accused Products directly infringe one or more claims of the '808 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the '808 Patent.

159. Lenovo has had knowledge and notice of the '808 Patent at least as of March 18, 2019, by virtue of Nokia providing the '808 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia's patent portfolio, which includes the '808 Patent.

160. Lenovo indirectly infringes claims of the ’808 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’808 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo’s inducement, Lenovo’s customers and end users use the Accused Products in the way Lenovo intends and directly infringe the ’808 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’808 Patent and with the intent, or willful blindness, that the induced acts directly infringe the ’808 Patent.

161. Lenovo also indirectly infringes claims of the ’808 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo’s affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo’s customers and end-users use of the Accused Products, such that the ’808 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the ’808 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the ’808 Patent. Lenovo has performed and continues to

perform these affirmative acts with knowledge of the ’808 Patent and with intent, or willful blindness, that they cause the direct infringement of the ’808 Patent.

162. Lenovo’s infringement of the ’808 Patent has damaged and will continue to damage Nokia.

COUNT II: PATENT INFRINGEMENT OF THE ’469 PATENT

163. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

164. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’469 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’469 Patent.

165. The Accused Products directly infringe one or more claims of the ’469 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States, these devices and thus directly infringes claims of the ’469 Patent.

166. Lenovo has had knowledge and notice of the ’469 Patent at least as of March 18, 2019, by virtue of Nokia providing the ’469 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’469 Patent.

167. Lenovo indirectly infringes claims of the ’469 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’469 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation,

and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo's inducement, Lenovo's customers and end users use the Accused Products in the way Lenovo intends and directly infringe the '469 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '469 Patent and with the intent, or willful blindness, that the induced acts directly infringe the '469 Patent.

168. Lenovo also indirectly infringes claims of the '469 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo's affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo's customers and end-users use of the Accused Products, such that the '469 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '469 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '469 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '469 Patent and with intent, or willful blindness, that they cause the direct infringement of the '469 Patent.

169. Lenovo's infringement of the '469 Patent has damaged and will continue to damage Nokia.

COUNT III: PATENT INFRINGEMENT OF THE '599 PATENT

170. Nokia incorporates by reference the preceding paragraphs as though fully set forth

herein.

171. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’599 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’599 Patent.

172. The Accused Products directly infringe one or more claims of the ’599 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the ’599 Patent.

173. Lenovo has had knowledge and notice of the ’599 Patent at least as of March 18, 2019, by virtue of Nokia providing the ’599 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’599 Patent.

174. Lenovo indirectly infringes claims of the ’599 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’599 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo’s inducement, Lenovo’s customers and end users use the Accused Products in the way Lenovo intends and directly infringe the ’599 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’599 Patent and

with the intent, or willful blindness, that the induced acts directly infringe the ’599 Patent.

175. Lenovo also indirectly infringes claims of the ’599 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo’s affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo’s customers and end-users use of the Accused Products, such that the ’599 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the ’599 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the ’599 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’599 Patent and with intent, or willful blindness, that they cause the direct infringement of the ’599 Patent.

176. Lenovo’s infringement of the ’599 Patent has damaged and will continue to damage Nokia.

COUNT IV: PATENT INFRINGEMENT OF THE ’273 PATENT

177. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

178. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’273 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’273 Patent.

179. The Accused Products directly infringe one or more claims of the ’273 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the

United States these devices and thus directly infringes claims of the ’273 Patent.

180. Lenovo has had knowledge and notice of the ’273 Patent at least as of March 18, 2019, by virtue of Nokia providing the ’273 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’273 Patent.

181. Lenovo indirectly infringes claims of the ’273 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’273 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo’s inducement, Lenovo’s customers and end users use the Accused Products in the way Lenovo intends and directly infringe the ’273 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’273 Patent and with the intent, or willful blindness, that the induced acts directly infringe the ’273 Patent.

182. Lenovo also indirectly infringes claims of the ’273 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo’s affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale,

contribute to Lenovo's customers and end-users use of the Accused Products, such that the '273 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '273 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '273 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '273 Patent and with intent, or willful blindness, that they cause the direct infringement of the '273 Patent.

183. Lenovo's infringement of the '273 Patent has damaged and will continue to damage Nokia.

COUNT V: PATENT INFRINGEMENT OF THE '764 PATENT

184. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

185. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the '764 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the '764 Patent.

186. The Accused Products directly infringe one or more claims of the '764 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the '764 Patent.

187. Lenovo has had knowledge and notice of the '764 Patent at least as of March 18, 2019, by virtue of Nokia providing the '764 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia's patent portfolio, which includes the '764 Patent.

188. Lenovo indirectly infringes claims of the '764 Patent, as provided in 35 U.S.C.

§ 271(b), by inducing infringement by others, such as Lenovo's customers and end-users, in this District and elsewhere in the United States. For example, Lenovo's customers and end-users directly infringe through their use of the inventions claimed in the '764 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo's inducement, Lenovo's customers and end users use the Accused Products in the way Lenovo intends and directly infringe the '764 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '764 Patent and with the intent, or willful blindness, that the induced acts directly infringe the '764 Patent.

189. Lenovo also indirectly infringes claims of the '764 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo's affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo's customers and end-users use of the Accused Products, such that the '764 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '764 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '764 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '764 Patent and with intent, or willful

blindness, that they cause the direct infringement of the ’764 Patent.

190. Lenovo’s infringement of the ’764 Patent has damaged and will continue to damage Nokia.

COUNT VI: PATENT INFRINGEMENT OF THE ’005 PATENT

191. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

192. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’005 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’005 Patent.

193. The Accused Products directly infringe one or more claims of the ’005 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the ’005 Patent.

194. Lenovo has had knowledge and notice of the ’005 Patent at least as of March 18, 2019, by virtue of Nokia providing the ’005 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’005 Patent.

195. Lenovo indirectly infringes claims of the ’005 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’005 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an

infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo's inducement, Lenovo's customers and end users use the Accused Products in the way Lenovo intends and directly infringe the '005 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '005 Patent and with the intent, or willful blindness, that the induced acts directly infringe the '005 Patent.

196. Lenovo also indirectly infringes claims of the '005 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo's affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo's customers and end-users use of the Accused Products, such that the '005 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '005 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '005 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '005 Patent and with intent, or willful blindness, that they cause the direct infringement of the '005 Patent.

197. Lenovo's infringement of the '005 Patent has damaged and will continue to damage Nokia.

COUNT VII: PATENT INFRINGEMENT OF THE '701 PATENT

198. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

199. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the '701 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the '701 Patent.

200. The Accused Products directly infringe one or more claims of the '701 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the '701 Patent.

201. Lenovo has had knowledge and notice of the '701 Patent at least as of the date of this complaint. Lenovo indirectly infringes claims of the '701 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo's customers and end-users, in this District and elsewhere in the United States. For example, Lenovo's customers and end-users directly infringe through their use of the inventions claimed in the '701 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo's inducement, Lenovo's customers and end users use the Accused Products in the way Lenovo intends and directly infringe the '701 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '701 Patent and with the intent, or willful blindness, that the induced acts directly infringe the '701 Patent.

202. Lenovo also indirectly infringes claims of the '701 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo's affirmative acts of

selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo's customers and end-users use of the Accused Products, such that the '701 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '701 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '701 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '701 Patent and with intent, or willful blindness, that they cause the direct infringement of the '701 Patent.

203. Lenovo's infringement of the '701 Patent has damaged and will continue to damage Nokia.

COUNT VIII: PATENT INFRINGEMENT OF THE '891 PATENT

204. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

205. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the '891 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the '891 Patent.

206. The Accused Products directly infringe one or more claims of the '891 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the '891 Patent.

207. Lenovo has had knowledge and notice of the '891 Patent at least as of March 18, 2019, by virtue of Nokia providing the '891 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia's patent portfolio, which includes the '891

Patent.

208. Lenovo indirectly infringes claims of the ’891 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’891 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo’s inducement, Lenovo’s customers and end users use the Accused Products in the way Lenovo intends and directly infringe the ’891 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’891 Patent and with the intent, or willful blindness, that the induced acts directly infringe the ’891 Patent.

209. Lenovo also indirectly infringes claims of the ’891 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo’s affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo’s customers and end-users use of the Accused Products, such that the ’891 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the ’891 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially

adapted for use in infringement of the ’891 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’891 Patent and with intent, or willful blindness, that they cause the direct infringement of the ’891 Patent.

210. Lenovo’s infringement of the ’891 Patent has damaged and will continue to damage Nokia.

COUNT IX: PATENT INFRINGEMENT OF THE ’818 PATENT

211. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

212. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’818 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’818 Patent.

213. The Accused Products directly infringe one or more claims of the ’818 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the ’818 Patent.

214. Lenovo has had knowledge and notice of the ’818 Patent at least as of March 18, 2019, by virtue of Nokia providing the ’818 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’818 Patent.

215. Lenovo indirectly infringes claims of the ’818 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’818 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or

otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo's inducement, Lenovo's customers and end users use the Accused Products in the way Lenovo intends and directly infringe the '818 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '818 Patent and with the intent, or willful blindness, that the induced acts directly infringe the '818 Patent.

216. Lenovo also indirectly infringes claims of the '818 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo's affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo's customers and end-users use of the Accused Products, such that the '818 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the '818 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the '818 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the '818 Patent and with intent, or willful blindness, that they cause the direct infringement of the '818 Patent.

217. Lenovo's infringement of the '818 Patent has damaged and will continue to damage Nokia.

COUNT X: PATENT INFRINGEMENT OF THE ’125 PATENT

218. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

219. Lenovo infringes, contributes to the infringement of, and/or induces infringement of the ’125 Patent by making, using, selling, offering for sale, and/or importing into the United States the Accused Products that are covered by one or more claims of the ’125 Patent.

220. The Accused Products directly infringe one or more claims of the ’125 Patent. Lenovo makes, uses, sells, offers for sale, and/or imports, in this District and elsewhere in the United States these devices and thus directly infringes claims of the ’125 Patent.

221. Lenovo has had knowledge and notice of the ’125 Patent at least as of March 18, 2019, by virtue of Nokia presenting the ’125 Patent to Lenovo. Lenovo has been involved in licensing discussions with Nokia regarding Nokia’s patent portfolio, which includes the ’125 Patent.

222. Lenovo indirectly infringes claims of the ’125 Patent, as provided in 35 U.S.C. § 271(b), by inducing infringement by others, such as Lenovo’s customers and end-users, in this District and elsewhere in the United States. For example, Lenovo’s customers and end-users directly infringe through their use of the inventions claimed in the ’125 Patent. Lenovo induces this direct infringement through its affirmative acts of manufacturing, selling, distributing, and/or otherwise making available the Accused Products, and providing instructions, documentation, and other information to customers and end users suggesting they use the Accused Products in an infringing manner, including in-store technical support, online technical support, marketing, product manuals, advertisements, online documentation, developer information, and API documentation. As a result of Lenovo’s inducement, Lenovo’s customers and end users use the Accused Products in the way Lenovo intends and directly infringe the ’125 Patent. Lenovo has

performed and continues to perform these affirmative acts with knowledge of the ’125 Patent and with the intent, or willful blindness, that the induced acts directly infringe the ’125 Patent.

223. Lenovo also indirectly infringes claims of the ’125 Patent, as provided by 35 U.S.C. § 271(c), by contributing to direct infringement committed by others, such as customers and end-users, in this District and elsewhere in the United States. Lenovo’s affirmative acts of selling and offering to sell, in this District and elsewhere in the United States, the Accused Products and causing the Accused Products to be manufactured, used, sold, and offered for sale, contribute to Lenovo’s customers and end-users use of the Accused Products, such that the ’125 Patent is directly infringed. The accused components within the Accused Products are material to the invention of the ’125 Patent, are not staple articles or commodities of commerce, have no substantial non-infringing uses, and are known by Lenovo to be especially made or especially adapted for use in infringement of the ’125 Patent. Lenovo has performed and continues to perform these affirmative acts with knowledge of the ’125 Patent and with intent, or willful blindness, that they cause the direct infringement of the ’125 Patent.

224. Lenovo’s infringement of the ’125 Patent has damaged and will continue to damage Nokia.

COUNT XI: DECLARATORY JUDGMENT THAT NOKIA HAS NEGOTIATED IN GOOD FAITH TOWARDS A LICENSE WITH LENOVO AND COMPLIED WITH THE ITU COMMON PATENT POLICY AND NOKIA’S RELEVANT LICENSING DECLARATIONS

225. Nokia incorporates by reference the preceding paragraphs as though fully set forth herein.

226. Nokia owns patents containing claims essential to the ITU H.264 Standard.

227. Lenovo makes, uses, imports, and/or sells devices that comply with the H.264 Standard.

228. Lenovo requires a license to practice one or more essential claims of Nokia's H.264 patents.

229. Nokia has voluntarily declared to ITU that it is "prepared to grant" licenses to its essential H.264 claims "on a worldwide, non-discriminatory basis and on reasonable terms and conditions. . ." According to Nokia's relevant Patent Statement and Licensing Declarations, as well as ITU's Common Patent Policy, "[s]uch negotiations are left to the parties concerned and are performed outside" of the ITU. Sec. 2.2, ITU Common Patent Policy.

230. Nokia has at all times been prepared to grant a license to Lenovo under its essential H.264 claims. To that end, Nokia negotiated in good faith with Lenovo since at least March 2019. Nokia provided Lenovo with a list of patents having essential claims to the H.264 standard and detailed claim charts showing Lenovo how its products infringed certain of Nokia's essential patent claims. In addition, Nokia repeatedly contacted Lenovo to move negotiations along and sent high-level executives to negotiation meetings to answer any of Lenovo's questions.

231. A dispute exists between Nokia and Lenovo concerning whether Nokia has negotiated in good faith towards a license with Lenovo and complied with the ITU Common Patent Policy and Nokia's relevant Patent Statement and Licensing Declarations. Accordingly, there is a case or controversy of sufficient immediacy, reality, and ripeness to warrant the issuance of a declaratory judgment.

232. Nokia seeks a declaration that Nokia has negotiated in good faith towards a license with Lenovo and has complied with the ITU Common Patent Policy and Nokia's relevant Patent Statement and Licensing Declarations.

233. In addition to a declaration, Nokia also requests an award of damages for the

expenses it has incurred as a result of Lenovo's failure to negotiate in good faith with Nokia.

DAMAGES

234. As a result of Lenovo's acts of infringement and negotiation conduct, Nokia has suffered and continues to suffer actual and consequential damages. However, Nokia does not yet know the full extent of the infringement and the amount of damages cannot be ascertained except through discovery and special accounting. To the fullest extent permitted by law, Nokia seeks recovery of damages at least for reasonable royalties, unjust enrichment, and benefits received by Lenovo as a result of using misappropriated technology, as well as damages incurred as a result of Lenovo's negotiation conduct. Nokia further seeks any other damages to which Nokia is entitled under law or in equity.

DEMAND FOR JURY TRIAL

235. Nokia hereby demands a jury trial for all issues so triable.

PRAYER FOR RELIEF

WHEREFORE, Nokia respectfully requests that this Court enter judgment in its favor as follows:

A. that Lenovo infringes the asserted claims of the Patents-in-Suit;

B. that Lenovo's infringement of the asserted claims of the Patents-in-Suit was willful, and that Lenovo's continued infringement of the asserted claims is willful;

C. awarding Nokia actual damages in an amount sufficient to compensate Nokia for Lenovo's infringement of the asserted claims of the Patents-in-Suit until such time as Lenovo ceases its infringing conduct;

D. awarding enhanced damages pursuant to 35 U.S.C. § 284;

E. awarding Nokia pre-judgment and post-judgment interest to the full extent allowed

under the law, as well as its costs;

F. as to asserted claims that are not essential to the H.264 Standard, entering an injunction precluding Lenovo and any entities in active concert with it from future acts of infringement;

G. as to asserted claims that are essential to the H.264 Standard, to the extent that Lenovo is adjudicated to have failed to negotiate in good faith with Nokia, and/or is adjudicated to have lost the right to claim benefits under Nokia's relevant Patent Statement and Licensing Declarations, entering an injunction precluding Lenovo and any entities in active concert with it from future acts of infringement;

H. that this is an exceptional case and awarding Nokia its reasonable attorneys' fees pursuant to 35 U.S.C. § 285;

I. ordering an accounting of damages for acts of infringement;

J. declaring that Nokia has negotiated in good faith towards a license with Lenovo and complied with the ITU Common Patent Policy and Nokia's relevant Patent Statement and Licensing Declarations;

K. declaring that Lenovo failed to negotiate in good faith towards a license with Nokia, and thus has lost or forfeited its right to claim third-party beneficiary status under Nokia's relevant Patent Statement and Licensing Declarations to the extent applicable to the claims of the asserted patents;

L. awarding Nokia actual damages in an amount sufficient to compensate Nokia for Lenovo's negotiation conduct;

M. awarding Nokia its costs of suit; and

N. awarding such other equitable relief which may be requested and to which Nokia is entitled.

Dated: September 25, 2019

/s/ Thomas G. Walker

Thomas G. Walker (N.C. State Bar 17635)
ALSTON & BIRD LLP
Bank of America Plaza
Suite 4000
101 South Tyron Street
Charlotte, NC 28280-4000
Telephone: (704) 444-1248
Facsimile: (704) 444-1111
Email: thomas.walker@alston.com

Matthew P. McGuire (N.C. State Bar 20048)
Sarah R. Cansler (N.C. State Bar 52058)
ALSTON & BIRD LLP
555 Fayetteville Street, Suite 600
Raleigh, NC 27601
Telephone: (919) 862-2200
Facsimile: (919) 862-2260
Email: matt.mcguire@alston.com
sarah.cansler@alston.com

LOCAL CIVIL RULE 83.1(D)
COUNSEL FOR PLAINTIFF

Theodore Stevenson, III, lead counsel
TX State Bar No. 19196650
Warren Lipschitz
TX State Bar No. 24078867
MCKOOL SMITH, PC
300 Crescent Court, Suite 1500
Dallas, TX 75201
Telephone: (214) 978-4000
Telecopier: (214) 978-4044
Email: tstevenson@mckoolsmith.com
wlipschitz@mckoolsmith.com

Local Civil Rule 83.1(e) Special Appearance Forthcoming for Plaintiff Nokia Technologies Oy