

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

REALTEK SEMICONDUCTOR CORP.,

Petitioner

v.

ATI TECHNOLOGIES ULC

Patent Owner

Case No.: IPR2023-00687

Patent 8,854,381

**DECLARATION OF DR. MASSOUND PEDRAM IN SUPPORT OF
PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO.
8,854,381**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	4
II. QUALIFICATIONS AND PERSONAL EXPERIENCE.....	5
III. LEGAL PRINCIPLES.....	9
IV. OVERVIEW OF THE '381 PATENT	15
A. Technology Background	15
B. File History	16
V. PERSON OF ORDINARY SKILL IN THE ART	18
VI. CLAIM CONSTRUCTION	19
A. “Shader Core”.....	19
B. “Task”	19
C. “Execute A First Task ... While Executing A Second Task”	19
D. “Engine”/“Plurality Of Engines”	20
E. “Load”	21
VII. SUMMARY OF THE PRIOR ART	21
A. Wilt.....	21
B. Jiao.....	24
C. Stuttard	26
VIII. GROUND 1: WILT AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20	32
A. Independent Claim 15	32
B. Dependent Claim 16.....	38
C. Dependent Claim 17	41
D. Dependent Claim 18.....	43
E. Dependent Claim 19.....	44
F. Dependent Claim 20.....	44

TABLE OF CONTENTS
(continued)

	Page
IX. GROUND 2: JIAO AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20	45
A. Independent Claim 15	45
B. Dependent Claim 16.....	58
C. Dependent Claim 17.....	58
D. Dependent Claim 18.....	60
E. Dependent Claim 19.....	60
F. Dependent Claim 20.....	61
X. GROUND 3: STUTTARD AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20	62
A. Independent Claim 15	62
B. Dependent Claim 16.....	69
C. Dependent Claim 17.....	70
D. Dependent Claim 18.....	70
E. Dependent Claim 19.....	70
F. Dependent Claim 20.....	71
XI. DECLARATION.....	72

I, Dr. Massoud Pedram, declare as follows:

I. INTRODUCTION

1. I have been engaged by Realtek Semiconductor Corp. (“Petitioner”) to provide my expert opinion on invalidity related issues in connection with Petitioner’s Petition for *Inter Partes* Review in IPR2023-00687 (“Petition”) before the Patent Trial and Appeal Board (“Board”). I was asked to provide my opinions in this expert declaration (“Declaration”). I intend to provide testimony regarding my opinions if called to testify at a deposition.

2. This Declaration addresses my opinions related to the validity of the Asserted Claims of U.S. Patent Nos. 8,854,381 (the “’381 patent”) (Exhibit 1001). I understand the challenged claims to be claims 15-20 of the ’381 patent. Accordingly, my Declaration addresses these claims.

3. I am being compensated for my time working in connection with this Petition. However, my compensation is not affected by or contingent upon the substance of my opinions or on the ultimate outcome of this Petition, and I have no other interest in this Petition.

4. The opinions expressed in this Declaration are my own and are based on my personal knowledge, my education, experience, and training, and on my understanding of the information and documents referenced in this Declaration. If called as an expert in this case, I anticipate that my testimony will concern the

matters addressed in this Declaration. My testimony may be affected by the positions that ATI Technologies ULC (“Patent Owner”) and/or its expert takes on these and other related topics after this Declaration is issued, either in the form of expert declarations or deposition testimony, or any ruling that the Board may make, and therefore I reserve the right to provide testimony regarding those issues.

II. QUALIFICATIONS AND PERSONAL EXPERIENCE

5. I am a Professor of Electrical Engineering and the Charles Lee Powell Chair in Electrical Engineering and Computer Science in the Viterbi School of Engineering at the University of Southern California (“USC”).

6. I am an expert on Very Large Scale Integration (“VLSI”) design including parallel computing, distributed processing, high-speed design, power-efficient design, digital integrated circuits, and semiconductor memory. I also have expertise in neural networks and graphic processing units (“GPUs”), as well as system-on-chip designs targeting applications ranging from high-performance computing to low-power embedded computing. My current research interests are in, among others, computer-aided design methodologies and tools for CMOS circuits and systems, energy-efficient computing, computer architecture and hardware, parallel and distributed computing, neural network acceleration, and integrated circuit and system design including system-on-chip design.

7. I earned my B.S. in Electrical Engineering from the California Institute of Technology in 1986, my M.S. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California at Berkeley in 1989 and 1991, respectively.

8. I was a Graduate Student Researcher in the Department of Electrical Engineering and Computer Sciences at the University of California at Berkeley from 1986 to 1991 during my graduate studies. Concurrently I had a part-time research position at the Xerox Palo Alto Research Center from 1987 to 1989. After receiving my doctorate, I joined the Department of Electrical Engineering at the University of Southern California as an Assistant Professor from 1991 to 1996. I became an Associate Professor and a Professor in 1996 and 2000, respectively. In 2013, I was named the Stephen and Etta Varra Professor in the school, which had been renamed the Viterbi School of Engineering. From 2017 to the present, I am the Charles Lee Powell Chair in Electrical Engineering and Computer Science in the Viterbi School of Engineering at USC.

9. I am a named inventor on twelve U.S. patents and two pending U.S. patent applications. These patents relate to topics ranging from pipelined architecture for logic circuits to novel address bus encoding techniques, and from charge recycling based multi-threshold CMOS design to low-leakage SRAMs. Two recently filed patent applications focus on hybrid arithmetic and logic processing of

neural networks and high-performance and scalable Montgomery modular multiplier circuits.

10. I have (co-)published five books and more than 800 archival and conference papers and book chapters on topic ranging from computer-aided design techniques and tools targeting VLSI circuit and system design to FPGA fabric design and synthesis, from low-power electronics to power distribution, conversion, and regulation.

11. I have been a member of multiple technical and scientific advisory boards for high-tech companies, including EPIC Design, Magma Design Automation, and Atrenta Inc., all of which went on to have successful initial public offerings or were acquired.

12. I have served as the (lead) principal investigator on many U.S. government (including the National Science Foundation, the Defense Advanced Research Projects Agency, the Department of Defense, and the Intelligence Advanced Research Project Activity) and private industry (including the Semiconductor Research Corporation) research projects, for some of which I have been the lead investigator of a team comprised of tens of faculty, students, and industry collaborators.

13. I was a recipient of the 1996 Presidential Early Career Award for Scientists and Engineers, and in 2000 I was selected a Fellow of the IEEE for

contributions to the theory and practice of low-power design and CAD. In 2008, I was named as an ACM Distinguished Scientist. I received the 2015 IEEE Circuits and Systems Society Charles A Desoer Technical Achievement Award for my contributions to modeling and design of low power VLSI circuits and systems, and energy-efficient computing.

14. I was recognized as the Third Most Cited Author at the 50th anniversary of the Design Automation Conference, Austin, Texas, in June 2013. I received a Top Three Author Award at the 20th Anniversary Asia and South Pacific Design Automation Conference, Chiba/Tokyo, Japan, in January 2015, and was named the Second Most Prolific and Second Most Cited Author Awards from the 20th Anniversary International Symposium on Low Power Electronics and Design, Rome, Italy, in July 2015. I have received multiple Best Paper Awards. I have also served on best or outstanding paper award committees.

15. I received the USC Viterbi School of Engineering Senior Research Award in 2017. I was named a fellow of the American Association for the Advancement of Science in early 2023.

16. Since July 2012, I have been the Executive Committee Chair of the IEEE/ACM International Symposium on Low Power Electronics and Design, a technical conference which I co-founded in 1996.

17. I served as the Vice President of Publications and Chair of the Distinguished Lecturer Program of the IEEE Circuits and Systems Society from 2005 to 2006 and 2003 to 2004, respectively. I served as a member of the Board of Governors of the IEEE Circuits and Systems Society from 2000 to 2003. I served as the Technical Program Chair of the International Symposium on Low Power Electronics and Design, Monterey, California, in August 1996. In addition, I have served on multiple advisory boards and chaired multiple conferences and symposiums.

18. I was the Editor-in-Chief of the ACM Transactions on Design Automation of Electronic Systems from June 2008 to May 2014. I was also the Inaugural Editor-in-Chief of the IEEE Journal on Emerging and Selected Topics in Circuits and Systems from January 2010 to December 2013. I have served as the editor or an associate editor for multiple publications.

19. My CV is attached as Exhibit 1007.

III. LEGAL PRINCIPLES

20. I am not a lawyer and offer no opinion of the law. Accordingly, the “Legal Principles” herein have been provided to me from counsel, but I have read and adopted it as my own. For example, I have, however, been informed by counsel of the legal principles that apply with respect to the issues for which I was asked to provide an opinion, including patent invalidity. I have applied these legal principles

in formulating my opinions and arriving at my conclusions. I have also been informed that the claims of an issued patent are presumed valid, but those claims can be found invalid if it can be proven, by clear and convincing evidence, that the claims fail to comply with one or more requirements of patentability.

21. It is my understanding that a patent may claim the benefit of the filing date of an earlier application only if the earlier application provides adequate disclosure of the patent's claims. I further understand that to satisfy the disclosure requirement, the disclosure of the earlier filed application must describe the later claimed invention in sufficient detail that one skilled in the art, as of the earlier filing date sought, could clearly conclude that the inventor invented the claimed subject matter. I understand that while the earlier application need not describe the claimed subject matter in precisely the same terms as found in the claims at issue, the prior application must convey with reasonable clarity to a person of ordinary skill in the art that, as of the filing date sought, the inventor was in possession of the invention. It is also my understanding that claims of a patent that contain new matter that is not adequately disclosed by an earlier filed application do not receive the benefit of the earlier filing date.

22. It is my understanding that a patent claim is invalid if the differences between the patented subject matter and the prior art are such that the subject matter as a whole would have been obvious at the time of the invention to a person of

ordinary skill in the art. I understand that the following factors must be evaluated to determine whether the claimed invention is obvious: (1) the scope and content of the prior art, (2) the difference or differences between each claim of the patent and the prior art, (3) the level of ordinary skill in the art at the time the invention of the patent was made, and (4) any secondary considerations or objective indicia of nonobviousness.

23. I understand that the combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results. In other words, when a claim simply arranges prior-art elements with each performing the same function it had been known to perform and yields no more than one would expect from such an arrangement, then such a combination is obvious. Moreover, when a patent claims a structure already known in the prior art that is altered by the mere substitution of one element for another known in the field, the combination is likely to be obvious unless the combination yields an unpredictable result.

24. I understand that a patentee's admissions about the scope and content of the prior art provide a factual foundation as to what a skilled artisan would have known at the time of invention. For example, by characterizing claimed elements as conventional in the specification, the patentee effectively admits that such things would be known to a person of ordinary skill in the art. I understand that I can rely on

such admissions and/or knowledge in forming an opinion as to whether claims are obvious. Such reliance could include, for example, furnishing a motivation to combine, or supplying a missing claim limitation. Thus, even though a patentee's admissions are not themselves prior art references, they can establish the background knowledge possessed by a person of ordinary skill in the art.

25. I have been informed that when a work is available in one field of endeavor, design incentives and other market forces can prompt variations of it, either in the same field or a different one. If a person of ordinary skill in the art can implement a predictable variation, such a variation is likely unpatentable. For the same reason, if a technique has been used to improve one device, and one of ordinary skill in the art would recognize that it would improve similar devices in the same way, using the technique is obvious unless its actual application is beyond the skill of the person of ordinary skill in the art. One question to consider is whether the improvement is more than the predictable use of prior-art elements according to their established functions.

26. I have been informed that it may often be necessary, in a validity analysis, to look to interrelated teachings of multiple patents or other publications or pieces of prior art; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by one of ordinary

skill in the art, all in order to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue.

27. I have been informed that a validity analysis need not seek out precise teachings directed to the specific subject matter of the challenged claim. Rather, it is appropriate to take account of the inferences and creative steps that a person of ordinary skill in the art would employ. I understand that a person of ordinary skill in the art is also a person of ordinary creativity.

28. I understand that a claim composed of several elements is not proved obvious merely by demonstrating that each element was, independently, known in the prior art. I understand that it can be important to identify a reason that would have prompted a person of ordinary skill in the art in the relevant field to combine the elements in the way the claimed invention does. Therefore, I am informed that, in determining whether a claimed combination is obvious, the correct question is whether the combination was obvious to a person of ordinary skill in the art at the time of the alleged invention. I am told that one of the ways in which subject matter can be proved obvious is by noting that there existed at the time of invention a known problem for which there was an obvious solution encompassed by the patent's claims. I understand that any need or problem known in the field of endeavor at the time of alleged invention and addressed by the patent can provide a reason for combining the elements in the manner claimed.

29. I understand that one should not assume that a person of ordinary skill in the art attempting to solve a problem will be led only to those elements of prior art designed to solve the same problem. Instead, I understand that since familiar items may have obvious uses beyond their primary purposes, in many cases a person of ordinary skill in the art would be able to fit the teachings of multiple prior art references together like pieces of a puzzle.

30. I understand that, when there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill in the art has good reason to pursue the known options within his or her technical grasp. If this leads to the anticipated success, then it is likely the product is not of innovation but of ordinary skill and common sense. I understand that, in such an instance, the fact that the combination was obvious to try may show that the combination is obvious.

31. I understand that, when determining whether a claimed combination is obvious, the correct analysis is not whether a person of ordinary skill in the art, writing on a blank slate, would have chosen the particular combination of elements described in the claim. Instead, I understand the correct analysis considers whether one of ordinary skill, facing the wide range of needs created by developments in the field of endeavor, would have seen a benefit to selecting the combination claimed.

32. I understand that, when determining whether a claimed invention is obvious, any “secondary considerations” of nonobviousness should also be considered. I further understand that there must be a nexus between any alleged secondary considerations and the patent claims. In particular, when the product that practices an invention is not co-extensive with the patented invention—for example, if the patented invention is only one component of a commercially successful product—then the patentee must show a sufficient relationship or nexus between the patented invention in a product and the secondary considerations, *e.g.*, commercial success of the product. In addition, I understand that the burden of introducing evidence of secondary considerations generally is on the Patent Owner.

IV. OVERVIEW OF THE '381 PATENT

A. Technology Background

33. The '381 Patent provides a processing unit that includes a plurality of virtual engines and a shader core. Ex. 1001, Figs. 2, 4. The plurality of virtual engines is configured to (i) receive, from an operating system (OS), a plurality of tasks in parallel with each other and (ii) load a set of state data associated with each of the plurality of tasks. The shader core is configured to execute the plurality of tasks in parallel based on the set of state data associated with each of the plurality of tasks. The processing unit may also include a scheduling module that schedules the plurality of tasks to be issued to the shader core. *Id.*, Abstract.

34. As was conventionally done at the time, the '381 Patent's computing system 100 comprises GPU 110, which can execute different streams of processing tasks from the OS scheduler. *Id.*, 1:48-51, 2:42-50. GPU 110 includes a command processor 230, input logic, a shader core 214, output logic 224, and a memory system 210. *Id.*, 2:57-60. Command processor 230 receives tasks from command buffers filled by the OS scheduler. *Id.*, 2:65-67. Command processor 230 includes a plurality of virtual engines. Background engines 202 process low-priority tasks. Low-latency engines 202C and 202E have priority access to the resources of GPU 110 in order to process high priority tasks. *Id.*, 7:1-19. Shader core 214 includes many processing elements 220 for executing tasks. Processing elements 220 are arranged as SIMD devices, enabling shader core 214 to execute a plurality of data-parallel processing tasks simultaneously. *Id.*, 7:66-8:3. Shader core 214 also includes one or more local data shares (LDS) 228 for storing data used by processing elements 220 to execute processing tasks. LDS 228 also stores state data associated with each task to be executed by shader core 214. *Id.*, 8:24-27.

B. File History

35. Applicant filed the patent application on September 1, 2010. *See* Ex. 1002, 1. On July 18, 2013, the Examiner issued what was entitled a Non-Final Office Action rejecting claims 1-24 as unpatentable over US Patent application 2011/0050712 (*Jackson*) in view of US Patent No. 8,345,053 (*Jiao*) and rejecting

independent claim 25 as being anticipated by Jiao. *Id.*, 121-27. Further, the Examiner rejected independent claims 1, 7, 13, and 19 as being indefinite due to the limitation “receive, from an operating system, a plurality of tasks substantially in parallel with each other...a shader core configured to execute the plurality of tasks substantially in parallel...” (in original claim 1) where the term “substantially” was not definite. *Id.*, 122. In response to this Office Action, Applicant amended the claims to address the indefiniteness issue and argued that the claims were neither anticipated nor obvious. *Id.*, 136-147.

36. On January 14, 2014, the Examiner issued a Final Office action still rejecting the claims as obvious, including two additional dependent claims 26 and 27 Applicant had added. *Id.*, 155-66. Applicant subsequently amended the claims again to “clarify features distinguishable over the applied references.” *Id.*, 181. Specifically, Applicant highlighted that the thread scheduler and hardware units in Jiao were associated with the same processing unit, the shader core, and thus did not teach or suggest “a shader core associated with the first processing unit and configured to receive the plurality of tasks from at least one of the plurality engines . . .,” in which the plurality of engines are “associated with a first processing unit and configured to receive, from a scheduling module associated with a second processing unit, a plurality of tasks and to load state data associated with each of the plurality of tasks,” as recited in claim 1 (emphasis added) and

similarly-recited in claims 7, 13, 19 and 25. *Id.*, 183. However, the same limitation relied upon in claim 1 was not actually recited in all of the other pending claims. Similarly, Applicant argued that U.S. Patent Application No. 2009/0109230 did not teach or suggest the above noted feature, and therefore should be patentable over Jackson in view of Jiao, and further in view of Miller. *Id.*, 184.

37. The Examiner found that the limitations of the newly amended claims required further consideration. *Id.*, 191. After an approved request for continued examination, the Examiner allowed the claims. *Id.*, 201-05. On October 7, 2020, allowed claims 1-27 were issued as U.S. Patent No. 8,854,381. *Id.*, 219; Ex. 1001, 1.

V. PERSON OF ORDINARY SKILL IN THE ART

38. I understand the ALJ has adopted the following definition of a person of ordinary skill in the art (“POSITA”): “a POSITA would have a bachelor degree in electrical engineering, computer engineering, computer science, or a related field, and at least two years of experience in ‘parallel computing or graphics processing including developing, designing, or programming software or hardware for graphics processing or parallel processing units, graphics accelerators[,] or other graphics processing systems,’ with additional education substitution for experience and vice

versa.” Ex. 1008, 9 (alteration in original). That is the definition that I have applied in my analysis.

VI. CLAIM CONSTRUCTION

39. I understand that the Board gives claim terms their ordinary and customary meaning as a POSITA would understand them in light of the patent’s specification and prosecution history. I also understand that several of the claim terms of the ’381 patent have already been construed by the ITC or argued by Petitioners. For the purposes of this Declaration, I have applied the narrower of the available constructions.

A. “shader core”

40. The ALJ in the related ITC matter has construed this term as “an array of processing elements.” Ex. 1008, 33-36. I will adopt this more limiting construction for the purposes of this declaration.

B. “task”

41. The ITC has construed this term as “a unit of processing work or portion of an application to be executed,” and it need not include a priority. Ex. 1008, 36-38. I will adopt this more limiting construction for the purposes of this declaration.

C. “execute a first task ... while executing a second task”

42. In Petitioner’s briefing in the related ITC matter, Petitioner argued that this term should be construed based on its plain and ordinary meaning, such that the

first task is executed at the same time as the second task, *i.e.*, the tasks are executed simultaneously. Ex. 1008, 126-128. I will adopt this construction position for the purposes of this declaration.

D. “engine”/“plurality of engines”

43. The ITC has not construed this term, but expressly instructed that it is not limited to either a hardware-only or a software-only implementation. Ex. 1008, 38-41. In Petitioner’s briefing in the related ITC matter, Petitioner argued that this term should exclude a hardware-only embodiment, and it should be construed to mean a component that processes data. The claim itself already defines the functionality and configuration of the engines and any construction beyond that is unnecessary and risks importing improper limitations from the specification. Patent Owner has proposed that an engine must be, among other things, a “portion of the first processing unit,” or a “hardware component of the first processing unit” (in direct contradiction to the ITC’s instruction). Nothing in the claims or specification requires the engine to be a portion or component of the first processing unit. Any such attempt is a transparent violation of importing limitations into the claims for the sole purpose of avoiding prior art. However, for the purpose of this Petition, since the claim itself already defines the functionality and configuration of the engines, Petitioner is not limiting the engine to either hardware or software will adopt this more limiting construction for the purposes of this Petition. Regardless,

the prior art discloses and/or renders engines obvious even under a hardware only construction. *See, e.g.*, discussion of Wilt herein.

E. “load”

44. The ITC has construed this term to not include extracting and storing, but otherwise did not construe this term, so it is presumed to have its plain and ordinary meaning. Ex. 1008, 41-42. I will adopt this more limiting construction for the purposes of this declaration.

VII. SUMMARY OF THE PRIOR ART

A. Wilt

45. Wilt’s invention sets forth a method for preserving a processing unit context associated with a processing unit thread. One advantage of the disclosed method is that it enables a parallel processing unit (PPU) context push at the beginning of a function and a PPU context pop prior to returning from the function. Thus, the function may execute within a single CPU thread, but advantageously operate on two distinct PPU contexts. Ex. 1004, 1:65-2:11.

46. Wilt discloses a computer system 100, which includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105. A parallel processing subsystem 112 is coupled to memory bridge 105 via a bus or other communication path 113. Ex.

1004, 2:59-3:9. In one embodiment parallel processing subsystem 112 is a graphics subsystem that delivers pixels to a display device 110. *Id.*, 3:24-35.

47. The parallel processing subsystem 112 of embodiment of Fig. 2 of Wilt includes one or more parallel processing units (PPUs) 202, each of which is coupled to a local parallel processing (PP) memory 204. *Id.*, 3:36-48. PPU 202 includes a number of cores 208. Each processing core 208 can simultaneously execute a large number of threads. Cores 208 receive processing tasks to be executed via a work distribution unit 210, which receives commands defining processing tasks from a front-end unit 212. Work distribution unit 210 can implement a variety of algorithms for distributing work. *Id.*, 4:1-23. Cores 208 can be programmed to execute processing tasks relating to a wide variety of applications, including data transforms, filtering of video and/or audio data, modeling operations (*e.g.*, applying laws of physics to determine position, velocity and other attributes of objects), image rendering operations (*e.g.*, vertex shader, geometry shader, and/or pixel shader programs), and so on. *Id.*, 4:33-45. Operation of core 208 is controlled via a core interface 303. In some embodiments, core interface 303 receives data to be processed (*e.g.*, primitive data, vertex data, and/or pixel data) as well as state parameters and commands defining how the data is to be processed from work distribution unit 210. Core interface 303 can load data to be processed into shared memory 306 and parameters into parameter memory 308.

Core interface 303 also initializes each new thread or SIMT thread group in instruction unit 312, then signals instruction unit 312 to begin executing the threads. When the execution of a thread or SIMT thread group is completed, core 208 notifies core interface 303. Core interface 303 can then initiate other processes. *Id.*, 8:14-29.

48. PPU 202 may transfer data from system memory 104 and/or local PP memories 204 into internal (on-chip) memory, process the data, and write result data back to system memory 104 and/or local PP memories 204, where such data can be accessed by other system components. *Id.*, 5:59-6:12. Wilt discloses that some embodiments of the parallel processing subsystem comprise PPUs for graphics processing and PPUs for general-purpose processing. *Id.*, 4:46-60, 8:56-67.

49. Wilt discloses various embodiments and/or configurations of elements and/or features. A person of ordinary skill in the art would have understood that the concepts, components and/or features apply equally across each embodiment. That is, a person of ordinary skill in the art would not have understood them to be mutually exclusive and they could be “mixed and matched” within the spirit of the disclosure. Moreover, a person of ordinary skill in the art would have been motivated to mix and match them as they all are disclosed in the same reference and are directed to the same field of invention and solving the same problems.

B. Jiao

50. Jiao discloses a computing system for graphics processing, which includes a system memory, a computational core, and a bus coupling the two, as described in more detail below.

51. Jiao discloses a plurality of execution units, one of which can be configured to process a rendering thread involving some constants and a priority level; a constant buffer configurable to store the constants; and an execution unit control unit configurable to assign the thread to one of the execution units; a constant buffer control unit providing a translation table to map the constants into the slots of the physical storage space. Ex. 1005, Abstract.

52. Jiao discloses a graphics processor system, comprising a display device and local memory 106 coupled to a graphics processing unit (GPU) 114. Ex. 1005, 4:49-5:67. This GPU includes a graphics processor 202, which comprises the compiler 115, a multiple execution unit (EU) computational core 204, an EU pool control unit 206, and a graphics pipeline with fixed function logic 208. *Id.*, 5:33-57. The computational core 204 comprises an execution unit pool (EUP) 414, which in one embodiment comprises one or more execution units (EUs) 420a through 420h. Each of the EUs 420 includes control logic and local memory, and each EU 420 is capable of processing multiple instructions within a single clock cycle. Thus, the EU pool 414, at its peak, can process multiple threads

simultaneously or substantially simultaneously. *Id.*, 8:15-34. Further, Jiao discloses that the EUP 414 is configured to receive a plurality of tasks from at least a vertex shader, a geometry shader, and/or a pixel shader in graphics pipeline 208. *Id.*, 5:33-57. The EUP 414 is thus configured to execute multiple, simultaneous threads based on respective state data including constants, rendering context information, and graphics data. The EU pool control unit 206, which includes a vertex cache and/or a stream cache, is responsible for assigning vertex shader, geometry shader, and pixel shader tasks to threads of the various execution units 420. *Id.*, 6:65-7:18, 9:39-54.

53. Jiao discloses a bus coupled to the local memory and the GPU (hence the computational core 204) via a memory interface unit and a plurality of 512-bit buses coupled to the computational core 204 and the memory in graphics pipeline 204 and/or the vertex/stream caches in the EU pool control unit 206. Ex. 1005, 4:49-5:6, 7:35-55. Jiao discloses that a plurality of engines including vertex, geometry, and pixel shaders is configured to receive a plurality of tasks from the EU pool control unit 206 and that the shaders load constants, rendering context information, and graphics data before commencing execution of a task. Constants for performing one or more of the shader computations are provided by a constant buffer of a constant buffer system 200. *Id.*, 6:1-48, 8:65-9:14.

54. Jiao discloses an embodiment in which each EU is configured for simultaneous multi-threaded operations running multiple contexts. Ex. 1005, 9:25-38. Jiao also teaches that a low-latency task is associated with a slow access path to constants using the system's memory hierarchy, whereas a regular-latency task is associated with a fast access path to constants using the constant buffers in each EU 420. *Id.*, 10:13-44. Finally, Jiao discloses a graphic processor system, which may be configured as a computer system, comprising components such as arithmetic logic units and other logic and performs functionalities suitable for graphics-processing tasks and general-compute tasks. *Id.*, 11:4-13.

55. Jiao discloses various embodiments and/or configurations of elements and/or features. A person of ordinary skill in the art would have understood that the concepts, components and/or features apply equally across each embodiment. That is, a person of ordinary skill in the art would not have understood them to be mutually exclusive and they could be "mixed and matched" within the spirit of the disclosure. Moreover, a person of ordinary skill in the art would have been motivated to mix and match them as they all are disclosed in the same reference and are directed to the same field of invention and solving the same problems.

C. Stuttard

56. Stuttard discloses a data processing system that comprises a plurality of processing blocks, each block comprising an array of processing elements

capable of executing graphics processing tasks including both pixel and vertex processing. Each processing block is organized as a single-instruction multiple-data block, which means that it can run the same instruction stream (thread) on many data items in parallel. Furthermore, each processing block can run its own thread independently of the others although all processing blocks can also run a single thread. Stuttard's system includes a thread manager that receives threads (including vertex and/or pixel processing threads) from a host CPU via an Embedded Processing Unit and Local Memory, stores them in a cache unit operative to store both pixel and vertex command threads, loads status information for each thread, selects active threads based on the thread priority level and status information before passing them on to the processing blocks for processing. A more detailed description follows.

57. Stuttard discloses a data processing apparatus for producing graphics images for display on a screen, comprising a SIMD (single instruction multiple data) array of processing elements, wherein the processing elements are operably divided into a plurality of processing blocks, the processing blocks being operable to process respective groups of data items. Ex. 1006, 2:17-23, 4:3-9. By organizing its processing elements into a plurality of processing blocks, Stuttard system overcomes the inefficiency associated with conventional SIMD architectures when a system must process a large number of relatively small data

item groups (each data item group is associated with an instruction stream). This is because a SIMD array can only process one (small) data item group at a time, potentially leaving a large proportion of the processing elements in the array unused while data is being processed for that group. *Id.*, 1:24-2:10. With multiple SIMD processing blocks, each processing block is operable to process respective groups of data items. *Id.*, 2:25-33. In other words, Stuttard's system can simultaneously process multiple instruction streams with each stream executing on one processing block.

58. Stuttard discloses that the elements in its data processing apparatus are controlled to execute series of instructions in an instruction stream, and such instruction streams for the processing blocks are known as "threads." *Id.*, 10:7-10. Stuttard explains that by supporting the concurrent execution of multiple threads in a data processing architecture, the processing element array can be kept active, by processing another thread when the current active thread is halted. *Id.*, 10:19-31. Stuttard explicitly discloses threads are assigned priorities relative to one another and that they may perform various 2D or 3D graphics processing activities. *Id.*, 11:16-25. At any moment in time, each thread may be assigned to any one of a number of tasks. For example, thread zero may be assigned for general system control, thread 1 assigned to execute 2D (two dimensional) activities, and threads 2

to 7 assigned to executing 3D activities (such as calculating vertices, primitives or rastering). *Id.*, 11:16-25.

59. Stuttard discloses that the graphics data processing includes performing both vertex and pixel processing instructions. *Id.*, 20:23-21:7, 22:12-32. Therefore, Stuttard's data processing apparatus is a multithreaded graphics system, performing vertex and pixel processing tasks, including geometric transformations, lighting, shading, texture mapping, etc. for graphics objects and shapes to be displayed on a screen. *Id.*, 1:9-22.

60. Stuttard's multithreaded graphics system includes a memory device, an arbiter, and a (command) processing core 10. This processing core, which is connected to the secondary buses 6 for the transfer of data, and to the primary bus 4 for the transfer of instructions, provides the graphics processing capability of the system. The processing core 10 includes a number of control units: thread manager 102, array controller 104, channel controller 108, a binning unit 1069 per block and a microcode store 105. These control units control the operation of processing blocks 106 of the graphics processing system. Ex. 1006, 7:25-8:7; *see also id.*, 8:9-13, Figs. 2, 3. Processing blocks 106 and thread manager 102 provide instructions to processing blocks 106 through array controller 104 and channel controller 108. *Id.*, 8:17-9:9. Processing block 106 includes processor units 1061a for executing the instructions. *Id.*, 9:11-20. Thread manager 102 manages

instruction streams or threads to increase efficiency and reduce response time. *Id.*, 10:7–11:1. Stuttard’s cache unit located inside of the thread manager stores both pixel and vertex threads. *Id.*, 11:16–25.

61. As shown in Fig. 4 of Stuttard, thread manager 102 includes thread scheduler 1025 and thread processors 1026, as well as cache memory unit 1024 for storing instructions and data for each thread—having one buffer per thread. *Ex.* 1006, 11:3–10. The threads are assigned priorities relative to one another. *Id.*, 11:16–17. Stuttard also teaches that the status block 1030 within the thread manager 102 receives status information 1036 from each of the threads. The status information is subsequently transferred to the thread scheduler 1025, which uses the status information to determine which thread should be active at any one time. *Id.*, 12:12–18. Thread processors 1026 control the issuance of instructions from the thread manager to maintain processing of simultaneously active program threads. *Id.*, 11:27–33. Stuttard’s thread scheduler 1025, working in conjunction with the thread processors 1026, in the thread manager 102, arbitrates and schedules a thread based on their relative priorities and (READY vs. YIELDING) status information about each thread. *Id.*, 12:12–18, 23:35–26:32. The thread manager is coupled to the processing blocks through the controllers. *Id.*, 8:17–27, Fig. 3. The instructions for the thread are sent to processor blocks via the controllers. *Id.*, 6:22–24. The processing blocks execute the instructions received from the

controllers. *Id.*, 9:11–20. Stuttard also discloses that the cache unit 1024 within the thread manager 102 could be replaced by a series of FIFO buffers, one per thread. *Id.*, 11:3-10.

62. Fig. 9 of Stuttard depicts a flowchart illustrating data processing using Stuttard's system. In this flowchart which concerns processing and displaying primitive graphical images, the data is prepared as vertices by the host system and then loaded into the processing elements of the Stuttard's system, with each processing element in a processing block representing a vertex of a primitive (*e.g.*, a line segment or a triangle). After some vertex processing operations, each PE transfers its data concerning its primitive to the mathematical expression evaluator (MEE) for processing into pixel data and identifying which pixels occur inside the primitive. In this step, each PE includes data about a respective pixel i.e., data is stored on a pixel per pixel basis. *Id.*, 20:23-22:33. Evidently, since there are eight processing blocks in Stuttard's system (*Id.*, Figs. 2(b), 4), each of which can execute one thread, a processing block may start the vertex processing for a newly arrived primitive while a different block is still finishing up the pixel processing for a previous primitive.

63. Stuttard discloses various embodiments and/or configurations of elements and/or features. A person of ordinary skill in the art would have understood that the concepts, components and/or features apply equally across

each embodiment. That is, a person of ordinary skill in the art would not have understood them to be mutually exclusive and they could be “mixed and matched” within the spirit of the disclosure. Moreover, a person of ordinary skill in the art would have been motivated to mix and match them as they all are disclosed in the same reference and are directed to the same field of invention and solving the same problems.

VIII. GROUND 1: WILT AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20

A. Independent Claim 15

64. It is my understanding that the preamble may not be limiting. To the extent that the preamble is limiting, it is my opinion that Wilt discloses a computing system. Wilt discloses computer system 100, which is a computing system. Ex. 1004, 2:59-3:3 (“FIG. 1 is a block diagram illustrating a computer system 100 configured to implement one or more aspects of the present invention. Computer system 100 includes a central processing unit (CPU) 102 and a system memory 104 communicating via a bus path that includes a memory bridge 105.”), Fig. 1. A parallel processing subsystem 112 is coupled to memory bridge 105 via a bus or other communication path 113. *Id.*, 2:59-3:9. In one embodiment parallel processing subsystem 112 is a graphics subsystem that delivers pixels to a display device 110. *Id.*, 3:24-35.

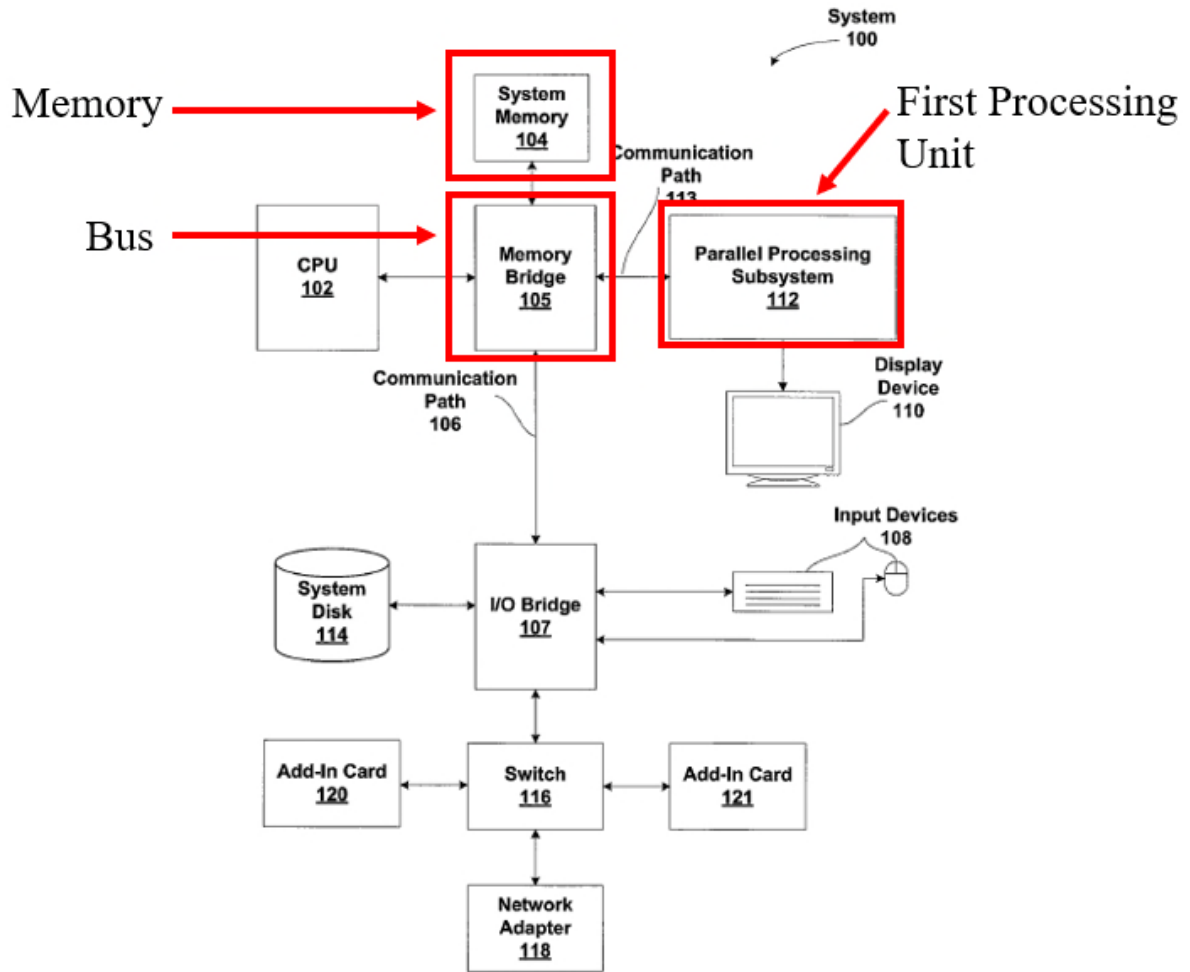


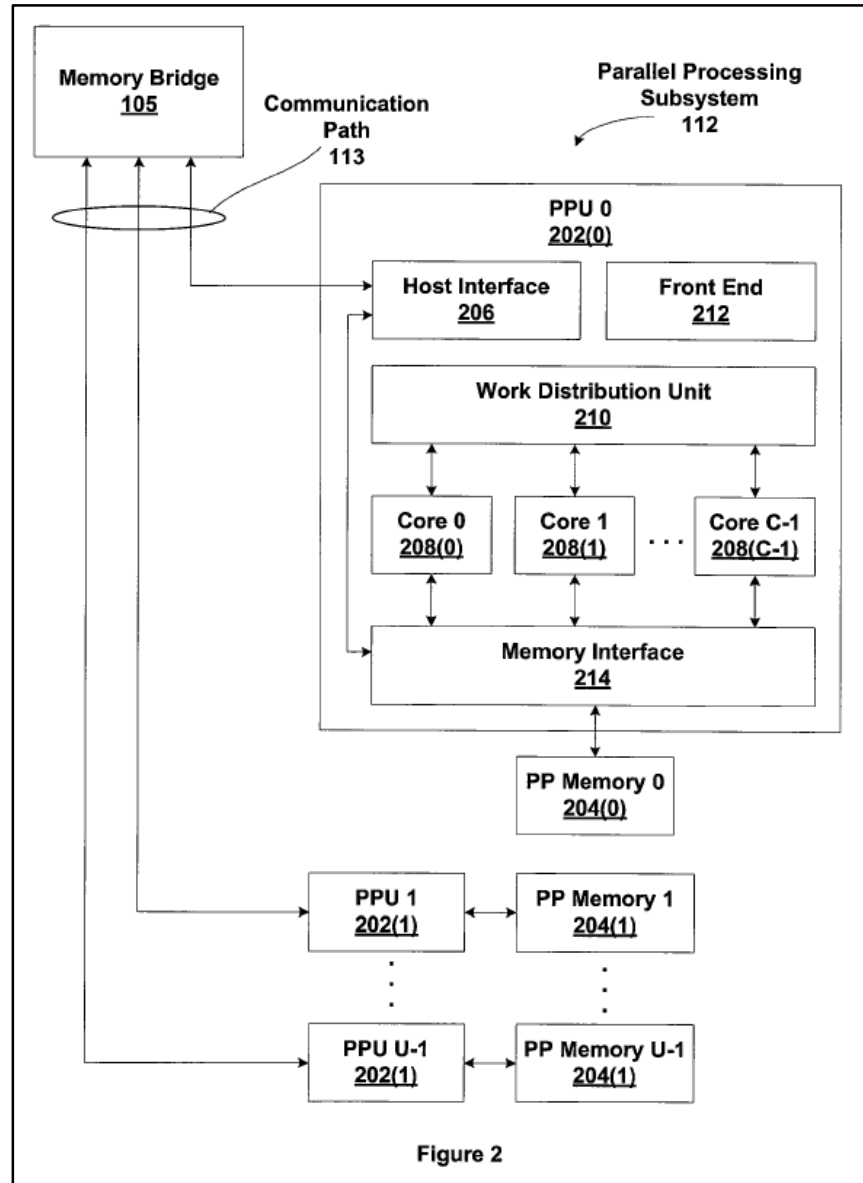
Figure 1

Id., Fig. 1.

65. It is further my opinion that Wilt discloses that its computing system 100 comprises a system memory 104, which satisfies this limitation. *Id.*, 2:59-3:3, Fig. 1.

66. It is also my opinion that Wilt discloses that its computing system 100 comprises a parallel processing subsystem 112. *Id.*, 3:3-9, Fig. 1. Wilt further discloses that parallel processing subsystem 112 comprises a plurality of parallel

processing units (PPUs) 202, each of which is coupled to a local parallel processing (PP) memory 204. *See, e.g., id.*, 3:36-48, Fig. 2 (reproduced below).



67. Each PPU 202 includes a number of cores 208. *Id.*, 4:1-3. Each processing core 208 can simultaneously execute a large number of threads. Cores 208 receive processing tasks to be executed via a work distribution unit 210, which receives commands defining processing tasks from a front-end unit 212. *Id.*, 4:3-7.

Wilt discloses that a “work distribution unit 210 can implement a variety of algorithms for distributing work.” *Id.*, 4:7-23. “Cores 208 can be programmed to execute processing tasks.” *Id.*, 4:33-40. Wilt explains that such tasks can relate to a “wide variety of applications, including data transforms, filtering of video and/or audio data, modeling operations (*e.g.*, applying laws of physics to determine position, velocity and other attributes of objects), image rendering operations (*e.g.*, vertex shader, geometry shader, and/or pixel shader programs), and so on.” *Id.*, 4:33-45.

68. “Operation of core 208 is controlled via a core interface 303.” *Id.*, 8:14-15. Wilt further explains that “core interface 303 receives data to be processed (*e.g.*, primitive data, vertex data, and/or pixel data) as well as state parameters and commands defining how the data is to be processed from work distribution unit 210.” *Id.*, 8:15-19. Core interface 303 carries out a number of processing functions, including loading data into shared memory 306, loading parameters into parameter memory 308, and initializing each new thread or SIMT thread group in instruction unit 312, and signaling instruction unit 312 to begin executing the threads. *Id.*, 8:19-24. When the execution of a thread or SIMT thread group is completed, core 208 also notifies core interface 303 and can initiate other processes. *Id.*, 8:14-29.

69. Accordingly, it is my opinion that a person of ordinary skill in the art would understand Wilt's parallel processing subsystem 112 to satisfy the claimed first processing unit of claim 15.

70. It is my opinion that Wilt discloses a bus coupled to the system memory and the parallel processing subsystem. For example, Wilt discloses a bus or other communication path 113 coupled to the parallel processing subsystem 112 (including PPU's) and the system memory 104 via memory bridge 105. *Id.*, 2:59-3:9, 3:49-67, Figs. 1, 2.

71. In my opinion, a person of ordinary skill in the art would have understood the work distribution unit 210, the front-end unit 212, and the core interface unit 303, all of which are within each PPU, to be "plurality of engines" as recited in claim 15. *Id.*, 3:36-67, 4:1-23, 4:46-5:3, 8:14-29, Figs. 1-3. These units receive and process tasks from software programs running on the host CPU, *e.g.*, a "driver program." *Id.*, 9:1-17, 9:63-10:31.

72. It is further my opinion that a person of ordinary skill in the art would have understood in Wilt's computer system the claimed "scheduling module" as recited in claim 15 is the application software running on the CPU. Indeed, Wilt gives an example of a "driver program executing on CPU 102" as the module that send tasks to Wilt's parallel processing subsystem. *Id.*, 10:21-25. Evidently, CPU 112 within Wilt's computer system is the claimed "second processing unit."

73. Alternatively, the work distribution unit 210, the front-end unit 212, and the core interface unit 303 of one PPU constitute one claimed “engine,” and the engines of the plurality PPUs 202 constitute the claimed “plurality of engines.”

74. “State data” refers to “texture handles, shader constants, transform matrices, or the like” associated with tasks. *See, e.g.*, Ex. 1001, 1:53-57, 3:3-7. Wilt uses the phrase “state parameters” to refer to “parameters such as the number of threads in the [collaborative Thread Array] CTA, the location in global memory 320 of an input data set to be processed using the CTA, the location in global memory 320 of the CTA program to be executed, and the location in global memory 320 where output data is to be written.” Ex. 1004, 10:25-32. Wilt discloses that core interface 303 loads the state parameters associated with each command thread into core 208 by loading parameters into parameter memory 308, which is part of core 208 of PPU 202. *Id.*, 8:14-29, 10:32-26.

75. In my opinion, Wilt discloses that parallel processing subsystem 112 comprises a plurality of parallel processing units (PPUs) 202, each of which is coupled to a local parallel processing (PP) memory 204. *See, e.g.*, Ex. 1004, 3:36-48, Fig. 2. Further, Wilt discloses a plurality of single-instruction multiple-thread (SIMT) parallel (i.e., multithreaded) processing cores 208 inside each PPU, each core is capable of executing hundreds of threads.

76. Each parallel processing unit (PPU) 202 is the claimed “shader core configured to receive the plurality of tasks” from its work distribution unit and the front end unit and to execute concurrent tasks (using its processing cores 208) based on respective state parameters associated with the tasks. *Id.*, 3:36-48, 4:1-5:3, 5:41-6:52, 6:66-7:57, Figs. 1-3.

77. Alternatively, the plurality of parallel processing units (PPUs) 202 is the claimed “shader core configured to receive the plurality of tasks” from the pluralities of work distribution units and the front end units and to execute concurrent tasks (using their pluralities of processing cores 208) based on respective state parameters associated with the tasks.

B. Dependent Claim 16

78. Each of the cores 208 inside a PPU 202 (which is the claimed “shader core”) of Wilt’s parallel processing subsystem 112 comprises a plurality of parallel processing engines 302. Wilt’s parallel processing engines 302 within a core can execute a thread and that the plurality of parallel processing engines 302 can run a certain number of concurrently executing threads (referred to as a “warp” or “thread group”). Moreover, a plurality of related thread groups may be simultaneously active on a core 208, with each thread group running on different parallel processing engines 302 within the core. *Id.*, 6:13-27, 6:28-35, 8:56-67, 9:1-17. As I explained above, core interface unit 303 loads state parameters associated

with each command thread into core 208 by loading parameters into parameter memory 308, which is part of core 208 of PPU 202. *Id.*, 8:14-29, 9:63-10:36. Therefore, the processing cores 208 constitute the claimed “processing elements” of the ’381 Patent. Each SIMT parallel processing core 208 “executes instances of a single parallel program on different data across a plurality of parallel processing engines 302 included in the core 208.” Thus, each core 208 may be configured to execute a thread on each of its parallel processing engines 302, and therefore, execute a “warp” or “thread group” on its plurality of parallel processing engines 302. *Id.*, 6:13-27, and 6:36-65. Further, each core 208 includes a shared memory 306 that is shared among all of the processing engines 302 and an additional on-chip parameter memory and/or cache 308 in each of its constituent parallel processing engines 302. “This Parameter memory/cache 308 can be used, e.g., to hold state parameters and/or other data (e.g., various constants) that may be needed by multiple threads.” *Id.*, 6:66-7:24. In addition, “each processing engine 302 is multithreaded and can execute up to some number G (e.g., 24) of threads concurrently, e.g., by maintaining current state information associated with each thread in a different portion of its assigned lane in local register file 304.” Note that “since each processing engine 302 can support up to G threads concurrently, it follows that up to G SIMT thread groups can be executing in core 208 at any given time.” *Id.*, 7:25-32, 7:26-8:4. Moreover, “in a given cycle, all processing engines

302 in core 208 are nominally executing the same instruction for different threads in the same SIMT thread group.” *Id.*, 7:65-8:1. Therefore, although processing engines 302 in core 208 are nominally executing the same instruction stream for different threads in the same SIMT thread group, different cores 208 can execute different instruction streams in different SIMT thread groups. In other words, cores 208 can simultaneously execute different tasks (each task comprising its own SIMT thread group) based on their associated state data (which is stored in the parameter memory 308 of each processing core 208).

79. As I explained above, SIMT parallel processing cores 208 can execute multiple tasks based on their associated state data simultaneously. Therefore, the collection of processing cores 208 inside each PPU 208 can be grouped into a first plurality of processing cores configured to execute a first task based on a first set of associated state data and a second plurality of processing cores configured to execute the second task based on a second set of associated state data.

80. Alternatively, each of the PPUs 202 among the plurality of PPUs (which is the claimed “shader core”) of Wilt’s parallel processing subsystem 112 constitute the claimed “processing elements” of the ’381 Patent. The PPUs work independently of each other and may have different functionality and logic implementations. The PPUs 202 can execute multiple tasks including, for example, graphics processing and general-purpose computation tasks. *Id.* at 3:36-47, 4:46-60

Moreover, the PPUs 202 can simultaneously execute multiple graphics processing tasks, one task per PPU based on its associated state data (which is stored in the parameter memories 308 of cores 208 inside the PPU). *Id.* at 3:49-67 4:1-10, and 7:9-24. Therefore, the collection PPUs 202 in Wilt's parallel processing subsystem 112 can be grouped into a first plurality of PPUs configured to execute a first task based on a first set of associated state data and a second plurality of PPUs configured to execute the second task based on a second set of associated state data.

81. In my opinion, Wilt teaches a shader core comprising first and second pluralities of processing elements configured to execute the first and second tasks based on the first and second set of state data, respectively.

C. Dependent Claim 17

82. Wilt explains that some or all of PPUs 202 in the parallel processing subsystem 112 are graphics processors with rendering pipelines that can be configured to perform various tasks, including both graphics processing and general-purpose computational tasks. *Id.*, 4:46-60. It is my opinion that a POSITA would understand that to support concurrent execution of multiple tasks in a pipelined architecture comprising a plurality of engines, queues, which are configured to receive tasks supplied by these engines, are needed to enable the flow and scheduling of tasks from the plurality of engines to the shader core

(Wilt's core 208) and that higher efficiency and reduced processing time can be achieved by using multiple queues dedicated to the work distribution unit 210, front end 212, and core interface 303 in a PPU 202. Thus, it would have been obvious to a person of ordinary skill in the art to use a first queue configured to receive the first task and a second queue configured to receive the second task.

83. Further, Wilt explains that “in one embodiment, work distribution unit 210 receives a “ready” signal from each core 208 indicating whether that core has sufficient resources to accept a new processing task. When a new processing task arrives, work distribution unit 210 assigns the task to a core 208 that is asserting the ready signal; if no core 208 is asserting the ready signal, work distribution unit 210 holds the new processing task until a ready signal is asserted by a core 208.” *Id.*, 4:1-23. It is my opinion that a POSITA would know the work distribution unit 210 must make use of buffers (queues) to affect the work distribution and the “check-and-hold” functions.

84. As I explained above Wilt's work distribution unit 210, front end unit 212, and core interface unit 303 are the claimed “engines” for each PPU 202 (the claimed “first processing unit”).

85. Alternatively, where the work distribution unit, front-end, and core interface of one PPU constitute one engine, and the engines of a plurality PPUs

constitute the plurality of engines, each engine comprises a queue configured to receive tasks.

D. Dependent Claim 18

86. Wilt teaches that intermediate results of thread executions can be stored can be stored to global memory 320 (outside core 208) or retained in shared memory 306 (within core 208) for further processing and that shared memory 306 can be accessed with lower latency and/or greater bandwidth than global memory, storing intermediate results in shared memory 306 advantageously improves processor throughput. *Id.*, 9:63-10:36. Wilt also explains that, in some embodiments, any processing engine 302 can read to or write from any location in shared memory 306 with equally low latency and that the shared memory 306 may be implemented as a shared register file or shared cache memory. *Id.*, 6:66-7:8. In my opinion, a person of ordinary skill in the art would have known that these different implementations will result in different access latencies for tasks. A person of ordinary skill in the art would have further understood that, using the appropriate type of memory resource, Wilt's computing system can support simultaneous execution of multiple tasks, some of which are low-latency tasks while others are regular-latency tasks.

87. Finally, it is my opinion that a person of ordinary skill in the art would know that different application requirements translated to different latency

requirements. For example, real-time display rendering required low latency to achieve a smooth playback whereas background tasks can be treated as regular-latency tasks in a graphics pipeline. Thus, a person of ordinary skill in the art would have understood this limitation to be obvious.

E. Dependent Claim 19

88. Wilt discloses that some embodiments of the parallel processing subsystem comprise PPUs for graphics processing and PPUs for general-purpose processing. *Id.*, 3:24-31, 4:33-45, 4:46-60, 8:30-44, 8:45-52, 8:56-67. Therefore, it is my opinion that Wilt teaches that in its computing system the tasks can be either graphics-processing or general-compute type.

F. Dependent Claim 20

89. Wilt discloses context swapping using a PPU context stack. *Id.*, Abstract, 12:51-63. In my opinion, a person of ordinary skill in the art would have understood this to be necessitated by tasks of different priority, such that a task with a higher priority, aided by pushing context to and popping context from a stack, can pause the execution of a task with a lower priority. Thus, a person of ordinary skill in the art would have understood this limitation to be obvious.

IX. GROUND 2: JIAO AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20

A. Independent Claim 15

90. I have been informed that the preamble may not be limiting.

Regardless, Jiao's disclosure is directed to graphics processing/processor systems, comprising: a plurality of execution units, one of which can be configured to process a rendering thread involving some constants and a priority level; a constant buffer configurable to store the constants; and an execution unit control unit configurable to assign the thread to one of the execution units; a constant buffer control unit providing a translation table to map the constants into the slots of the physical storage space. Ex. 1005, Abstract. In my opinion, a person of ordinary skill would understand Jiao's graphics processing systems (e.g., graphics processor system 100) to be a computing system.

91. Indeed, Jiao expressly discloses that the graphics processor system is implemented as a "computer system." Ex. 1005, 4:54-55. Figure 1 (reproduced below) shows a "computer system" embodiment of Jiao's graphics processor system that includes, among other things, a CPU 126, a constant buffering system (CB System 200), and a GPU 114, that itself also includes a graphics processor 202 and computational core 204, along with various memories 106, 124, and display components 102 and 104. *Id.*, Figs. 1, 2.

92. Jiao's disclosed systems perform "computations," process multiple instructions within a single clock cycle, and execute and/or process multiple, simultaneous threads. *See*, Ex. 1005, 3:31-4:23, 5:33-57, 6:65-7:18, 8:15-34, 9:39-54. In short, a person of ordinary skill would understand that Jiao discloses a graphic processor system, configured as a computer system, comprising components such as arithmetic logic units and other logic that performs functionalities suitable for graphics-processing tasks and general-compute tasks. *Id.*, 11:4-13.

93. In my opinion, Jiao discloses a memory as part of its graphics processor system. For example, Jiao discloses system memory 124, local memory 106 (e.g., which may comprise a display buffer, frame buffer, texture buffer, command buffer, etc.), memory 306 and various caches (e.g., depth or Z-cache, color or D-cache, texture or T-cache, etc.) in the logical graphics pipeline of Figure 3, a vertex/stream cache in EU pool control and vertex/stream cache 206, and L2 cache 422 in computational core 204. *See*, Ex. 1005, 4:49-5:6, 33-57, 6:1-48, 6:65-7:18-27, 35-55, 8:15-41, Figures 1-4.

94. It is further my opinion that Jiao discloses that its graphics processor system includes a graphic processing unit (GPU 114). Jiao's GPU is comprised of a computational core 204 (also known as a software programmable core processing unit), which satisfies the requirement of claim 15 for a first processing unit. The

computational core is comprised of a pool of multiple execution units to meet the computing requirement imposed by shader tasks associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208. Ex. 1005, 5:47-51; *see also* 4:49-5:57, 6:1-48, 6:49-7:55, Figs. 1-4.

95. Jiao discloses a bus coupled to the memory and the GPU. For example, Jiao discloses a bus coupled to local memory 106 and the GPU via a memory interface unit (MIU 110), and a plurality of 512-bit buses coupled to memory 306 and graphics processor 202 (hence, computational core 204), and 512-bit buses coupled to vertex/stream cache and computational core 204. Ex. 1005, 4:49-5:6, 7:19-55; *see also* Figure 1. Alternatively, Jiao also discloses a bus coupled to system memory 124 and the GPU (hence, computational core 204) via PCI-E BIU 118. *Id.*, 4:49-5:6, Fig. 1.

96. In my opinion, a person of ordinary skill in the art would have understood Jiao to disclose buses to couple the various processing components and memory. *See, e.g.*, Ex. 1005, 8:60-64 (“Other components not shown but understood as present by those having ordinary skill in the art include crossbars or buses for the transfer of input and output data, registers, among other components, for processing by the various EUs 420.”). Alternatively, a person of ordinary skill in the art would have found it obvious that Jiao discloses a bus coupled to system

memory 124 and the GPU (hence, computational core 204) via PCI-E BIU 118, a bus coupled to local memory 106 and the GPU via a memory interface unit (MIU 110), and a plurality of 512-bit buses coupled to memory 306 and graphics processor 202 (hence, computational core 204), and 512-bit buses coupled to vertex/stream cache and computational core 204. Ex. 1005, 4:49-5:6, 7:19-55; *see also* Fig. 1.

97. In my opinion, a POSITA would have understood the vertex shader 304, geometry shader 306, and pixel shader 310 in graphics pipeline 208 of GPU 114 to be “engines” as recited in claim 15. These shaders receive and process graphics data (e.g., graphics primitives) to produce pixel data. Ex. 1005, 6:1-48, 49-64. As identified above, Jiao’s computational core comprises vertex, geometry, and pixel shaders. *Id.*, 5:47-51. (“The computational core 204 comprises a pool of multiple execution units to meet the computing requirements imposed by shader tasks associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208.”). *Id.*, 6:59-64. (“Various types of shader programs can be executed or mapped to the computational core 204, including the vertex shader 304, geometry shader 306, and pixel shader 310. That computational core 204, for multi-issue processors, is capable of processing multiple instructions within a single clock cycle.”).

98. Jiao discloses that each execution unit 420 of the EU pool 414 may store a plurality of tasks or threads (e.g., in some embodiments up to 32) at the same time. Ex. 1005, 9:11-15. He also teaches that there exist “thread slots” within each of the execution units 420 of the EU pool 414 and that “the EU pool control unit 206 assigns a thread to one of the empty thread slots in the respective execution unit 420.” *Id.*, 9:15-21. In the alternative, the claimed “engines” are the vertex shader 304, geometry shader 306, and pixel shader 310 in graphics pipeline 208 of GPU 114 plus the “thread slots” within the execution units 420 of the EU pool 414.

99. Jiao discloses that the plurality of vertex, geometry, and pixel shaders (“engines”) are configured to receive a plurality of tasks from the EU pool control unit 206. For example, the plurality of vertex, geometry, and pixel shaders are configured to receive a plurality of tasks from the driver software 128 residing in the system memory 124 via the EU pool control unit 206. Ex. 1005, 5:17-23, 5:47-51, 6:65-7:27 and Fig. 1 and 4. Further, the shaders load constants, rendering context information, and graphics data before commencing execution of any tasks. *See, e.g., id.*, 6:13-44; 6:65-7:27, 8:35-41, 9:15-24, 9:39-54, Figs. 3, 4.

100. Jiao states the following: “The system memory 124 also comprises driver software 128, which communicates instruction sets or commands through the use of the CPU 126 to registers in the GPU 114, and an applications

programming interface (API) space 130, from which constants are either memory mapped or constant buffer mapped by a compiler 115 in the GPU 114.” Ex. 1005, 5:17-23. In the context of Jiao, it is my opinion that the commands satisfy the claimed tasks.

101. In Jiao, the vertex shader 304, geometry shader 306, and pixel shader 310, which constitute the claimed “engines” receive graphics (high level) tasks from the driver software 128 residing in the system memory 124 via the EU pool control unit 206 and produce (low level) shader tasks that are then mapped to threads of the various execution units 420 by the EU pool control unit 206. *Id.*, 9:43-49. This view is also corroborated here: “The computational core 204 comprises a pool of multiple execution units to meet the computing requirements imposed by shader tasks associated with various shader programs, including a vertex shader, geometry shader, and/or pixel shader, processing data for the graphics pipeline 208.” *Id.*, 5:47-51. In my opinion, a POSITA would have understood that Jiao’s disclosure discloses a high level task that has given rise to the low level thread, i.e., Jiao discloses task as claimed and construed by the ALJ. Ex. 1008, 36-38.

102. Petitioner has adopted the ITC’s construction for the term task to mean “a unit of processing work or portion of an application to be executed.” Threads are not tasks. The ‘381 Patent describes the relationship between task and

thread as follows: “To perform a single graphics-processing task, a GPU may execute a plurality of different threads (sequence of instructions). Each thread may comprise a shader program, such as a geometry shader, a pixel shader, a vertex shader, or the like. Each thread (e.g., shader program) is typically associated with a set of state data (such as texture handles, shader constants, transform matrices, or the like) that is locally stored in data-storage units of the GPU. The locally stored State data is called a context.” Ex. 1001, 1:49-57. An example of a graphics task may be drawing a triangle of given shape and size with white boundaries filled with a solid blue color in the right upper corner of a 1920 x1080 LCD Display. This task will be decomposed into a plurality of fixed-function graphics operations (*e.g.*, vertex and covered pixel generation) and shader programs (*e.g.*, vertex processing, lighting, line segment processing, pixel processing, color and texture mapping). The shader programs can run on the same programmable shader core. Consider a full 3-D view to comprise of thousands of such primitive shapes overlapping with one another (potentially covering each other) being seen from a particular viewing angle. This is how multiple shader programs will have to be executed on thousands of vertices and an even much larger number of pixels, and this is also why the highly parallel SIMD nature of modern GPUs is so effective, by running a relatively small number of shader programs on a large number of

vertices and pixels simultaneously and independently. Thus, the '381 Patent is consistent with the ALJ's construction, and a "task" is different than a thread.

103. Based on the ITC's construction, Jiao expressly distinguishes tasks and threads and discloses the claimed tasks. *See, e.g.*, Ex. 1005, 9:43-49 ("In this regard, the EU pool control unit 206 is responsible for assigning vertex shader geometry shader, and pixel shader tasks to threads of the various execution units 420, and also performs an associated 'bookkeeping' of the tasks and threads."). At times, Jiao could be read to conflate the terms, i.e., uses the term "tasks" and "threads" interchangeably. *See, e.g., id.*, 8:65-67, 9:10-13. However, it is my opinion that a person of ordinary skill in the art would have understood that Jiao's disclosure inherently discloses a higher level task that has given rise to the low level thread, i.e., Jiao discloses task as claimed and construed by the ALJ. To the extent not considered an express disclosure, for the same reasons I explained above, Jiao's disclosure would have also rendered this claim limitation obvious.

104. The '381 Patent uses the term "command" as a graphics task output in a standardized format by an end-user application through an application programming interface (API.) Ex. 1001, 1:33-41. The '381 Patent goes on to state that "the API communicates with a driver. The driver translates standard code received from the API into a native format of instructions understood by the GPU. The driver is typically written by the manufacturer of the GPU. The GPU then

executes the instructions from the driver.” *Id.*, 1:41-46. Further, the ’381 Patent states: “To pass tasks to the GPU, an operating-system (OS) scheduler stores the tasks in a command buffer. A conventional GPU processes one command buffer at a time. The OS scheduler serially places tasks in the command buffer, and the GPU typically processes the tasks in the order in which they are placed in the command buffer,” and “Command processor 230 receives tasks (e.g., graphics processing and general-compute tasks) from one or more command buffers filled by the OS scheduler.” *See e.g., id.*, 2:8-14, 6:65-67. Therefore, the term “commands” as used in the ’381 Patent satisfy the ALJ’s construction of the term “tasks”. Jiao use the term “command” to mean the same thing: “The system memory 124 also comprises driver software 128, which communicates instruction sets or commands through the use of the CPU 126 to registers in the GPU 114, and an applications programming interface (API) space 130, from which constants are either memory mapped or constant buffer mapped by a compiler 115 in the GPU 114.” Ex. 1005, 5:17-23. “Responsive to a command from the driver software 128, such as to draw a triangle, vertex information is passed to vertex shader 304 in the computational core 204 to implement vertex transformations. In particular, objects are transformed from object space to work space and/or screen space as triangles.” *Id.*, 7:57-62.

105. Jiao's EU pool control unit 206 is the "scheduling module" as recited in claim 15. As I explained above, Jiao discloses that the EU pool control unit 206 sends to the vertex, geometry, and pixel shaders ("engines") various tasks. *See, e.g., id.*, 6:65-7:27, 9:15-24, 9:39-54, Figs. 3, 4. For example, the EU pool control unit 206 "functions as a global scheduler for the tasks and assigns appropriate threads in the execution units 420 using a data-driven approach (e.g., vertex, pixel, geometry packets in the input)" and "is responsible for assigning vertex shader, geometry shader, and pixel shader tasks to threads of the various execution units 420." *Id.*, 9:15-24, 9:39-54. Further, Jiao's EU pool control unit 206 is associated with the command stream processor 406, which is the claimed "second processing unit." *See, e.g., id.*, 7:19-27 ("The command stream processor 406 provides triangle vertex indices to the EU pool control unit 206. In the embodiment of FIG. 4, the indices are 256-bits. The EU pool control unit 206 assembles vertex shader inputs from the stream cache and sends data to the computational core 204 (input E). The EU pool control unit 206 also assembles geometry shader inputs and provides those inputs to the computational core 204 (input F). In general, the EU pool control unit 206 controls the respective inflow and outflow to the computational core 204."). The command stream processor 406 is indeed a processing unit associated with EU pool control unit 206 because it performs "new

constant updates” for use by the EU pool control logic, which assembles inputs for shader tasks. *Id.*, 3:48-60, 4:10-23, 10:55-11:46, 14:13-23.

106. Alternatively, the driver software 128 constitutes the claimed “scheduling module.” Jiao discloses that the driver software “communicates instruction sets or commands through the use of the CPU 126 to registers in the GPU 114.” Ex. 1005, 5:17-23. Furthermore, responsive to a command from the driver software 128, such as to draw a triangle, vertex information is passed to the vertex shader 304 in the computational core 204 to implement vertex transformations, and after some processing in the fixed-function graphics engine 208, are passed to the pixel shader 310 in the computational core 204 to implement pixel shader functionality. *Id.*, 7:56-8:14. Further, Jiao’s driver software 128 is associated with the CPU 126, which is the claimed “second processing unit.” *See, e.g., id.*, 5:8-13.

107. Claim 15 requires only that the scheduling module be associated with a second processing unit. There is no “association” requirement for the “first processing unit” in claim 15. There is nothing in the asserted claims requiring any “association” with the first processing unit or precluding overlapping associations of any other elements. Thus, no other association, either with any other element or even the second processing unit, is relevant to the claim.

108. “State data” refers to “texture handles, shader constants, transform matrices, or the like” associated with tasks. *See, e.g.*, Ex. 1001, 1:53-57, 3:3-7. Jiao uses the phrase “rendering context” to refer to “specific visual properties related to the appearance of surfaces (e.g., materials, reflectance, shape, textures, etc.) and/or operations related to the same (e.g., required transformations) which may collectively be referred to as a rendering context for this object or group of objects. When rendering an object or group of objects for a given rendering context, a shader utilizes a constant data pool (e.g., context-specific constants stored in a constant buffer).” Ex. 1005, 1:50-58.

109. Jiao discloses that its shaders load constants, rendering context information, and graphics data before commencing execution of a task. Further, constants for performing one or more of the shader computations are provided by a constant buffer of a constant buffer system 200. Ex. 1005, 6:1-7:55, 8:65-9:14. In some embodiments, a shader loads constants through an EU pool control logic. *Id.*, 9:15-24, 12:33-52, 12:63-13:6, Figs. 3, 4.

110. Jiao’s computational core 204 comprises an execution unit pool (EUP) 414, which comprises one or more execution units (EUs) 420a through 420h (collectively referred to herein as EUs 420). Ex. 1005, 8:15-18. Each of the EUs 420 includes control logic and local memory, as described further below, and each EU 420 is capable of processing multiple instructions within a single clock cycle.

Id., 8:18-21. The EUP is configured to receive the plurality of tasks from at least a vertex shader, a geometry shader, and/or a pixel shader (i.e., the “engines”) in graphics pipeline processing. *See, e.g., id.*, 9:35-38 (“As various shader tasks are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are delivered to the respective execution units 420 to be carried out.”). A person of ordinary skill in the art would have understood Jiao’s EU pool (EUP) 414 to be a “shader core” as recited in claim 15.

111. Jiao’s EUP is configured to execute multiple, simultaneous tasks based on respective state data including constants, rendering context information, and graphics data. *See, e.g., Ex. 1005*, 3:48-60, 6:49-7:18. Further, the EUP can execute a first task while executing a second task. *See, e.g., id.*, 8:21-23 (“Thus, the EU pool 414, at its peak, can process multiple threads simultaneously or substantially simultaneously.”). Each task requires its corresponding context-specific constants to be able to be mapped or executed on some execution unit of the computational core. *See, e.g., id.*, 1:43-58, 6:49-7:18, 8:15-9:14, 9:25-54, 9:65-10:8.

112. In my opinion, a POSITA would know that Jiao’s high-level tasks (commands) are processed by the vertex shader 304, geometry shader 306, and pixel shader 310 and mapped to low level threads running on the execution units of the computational core by the EU pool control unit.

B. Dependent Claim 16

113. Jiao discloses that the EU pool (EUP) 414 (which is the claimed “shader core”) comprises a plurality of EUs 420. “Each of the EUs 420 includes control logic and local memory, as described further below, and each EU 420 is capable of processing multiple instructions within a single clock cycle. Thus, the EU pool 414, at its peak, can process multiple threads simultaneously or substantially simultaneously.” Ex. 1005, 8:15-34. Jiao’s EUP is configured to execute multiple threads simultaneously or substantially simultaneously based on respective state data including constants, rendering context information, and graphics data. *See, e.g., id.*, 6:1-64. Further, “[e]ach of these execution units 420 is identical in design and configurable for programmed operation. In one embodiment, each execution unit 420 is capable of multi-threaded operations simultaneously. As various shader tasks are generated by the vertex shader 304, geometry shader 306, and pixel shader 310, they are delivered to the respective execution units 420 to be carried out.” *See, e.g., id.*, 9:25-38. A person of ordinary skill in the art would have understood Jiao’s plurality of EUs 420 to constitute the claimed “first and second pluralities of processing elements” as recited in claim 16.

C. Dependent Claim 17

114. Jiao discloses that each execution unit 420 of the EU pool 414 may store a plurality of tasks or threads (e.g., in some embodiments up to 32) at the

same time. Ex. 1005, 9:11-15. He also teaches that there exist “thread slots” within each of the execution units 420 of the EU pool 414 and that “the EU pool control unit 206 assigns a thread to one of the empty thread slots in the respective execution unit 420.” *Id.*, 9:15-21. Jiao further explains that as individual tasks are generated, the EU pool control unit 206 handles assignment of these tasks to available threads within the various execution units 420 and that as tasks are completed, the EU pool control unit 206 further manages the release of the relevant threads. In this regard, the EU pool control unit 206 is responsible for assigning vertex shader, geometry shader, and pixel shader tasks to threads of the various execution units 420, and also, performs an associated “bookkeeping of the tasks and threads. *Id.*, 9:39-47. Therefore, Jiao teaches that there are buffers (e.g., queues) with multiple thread slots (up to 32 in some embodiments) in each of its execution units.

115. In the alternative where the claimed “engines” include the vertex shader 304, geometry shader 306, and pixel shader 310 in the graphics pipeline 208 of GPU 114 plus the “thread slots” within the execution units 420 of the EU pool 414, Jiao teaches that engines comprise queues, one per a group of threads. As I also explained above, Jiao’s EUP is configured to execute multiple, simultaneous tasks based on respective state data including constants, rendering context information, and graphics data. *See, e.g.*, Ex. 1005, 3:48-60, 6:49-7:18. Further,

the EUP can execute a first task while executing a second task. *See, e.g., id.*, 8:21-23 (“Thus, the EU pool 414, at its peak, can process multiple threads simultaneously or substantially simultaneously.”). Therefore, Jiao teaches that its “engines” comprise a first queue configured to receive the first task and a second queue configured to receive the second task.

D. Dependent Claim 18

116. Jiao teaches a fast access path 506 and a slow access path 508 for the provisioning of constants from API space 130 to the CB system 200, and the management by the CB system 200 of the constants for multiple rendering contexts. Ex. 1005, 10:13-54. A person of ordinary skill in the art would understand that a slow access path may be used for processing a regular-latency whereas a fast access path may be used for processing a low-latency task.

E. Dependent Claim 19

117. As I explained previously, Jiao discloses a graphic processor system, which may be configured as a computer system. *See* discussion with respect to claim 15 above. Further, the system comprises components such as arithmetic logic units and other logic and performs functionalities suitable for graphics-processing tasks and general-compute tasks. Ex. 1005, 6:1-48, 11:4-13. Further, Jiao teaches that the graphics processor 202 utilizes programmable vertex 304, geometry 306, and pixel shaders 310 and that each of these execution units 420 is

identical in design and configurable for programmed multi-threaded operations.

Id., 9:25-38. A person of ordinary skill in the art would have understood and found it obvious that such programmable execution units can be used to do both graphics-processing and general-compute tasks.

F. Dependent Claim 20

118. Jiao discloses that the rendering contexts associated with tasks processing have priority levels. *See, e.g.*, Ex. 1005, Abstract. “Embodiments of the present disclosure are directed to graphics processing systems, comprising: a plurality of execution units, wherein one of the execution units is configurable to process a thread corresponding to a rendering context, wherein the rendering context comprises a plurality of constants with a priority level; a constant buffer configurable to store the constants of the rendering context into a plurality of slot in a physical storage space; and an execution unit control unit configurable to assign the thread to one of the execution units; a constant buffer control unit providing a translation table for the rendering context to map the corresponding constants into the slots of the physical storage space.” *Id.*, 2:24-36. For example, Jiao explains that the constant blocks may be loaded to the CB system 200 using a fast access path 506 or a slow access path 508 based on the high or low priority level 504 associated with the respective rendering context. *Id.*, 10:13-54.

X. GROUND 3: STUTTARD AND THE KNOWLEDGE OF A PERSON OF ORDINARY SKILL IN THE ART RENDER OBVIOUS CLAIMS 15-20

A. Independent Claim 15

119. I have been informed that the preamble may not be limiting. To the extent that the preamble is limiting, it is my opinion that Stuttard discloses a computing system. Precisely, Stuttard describes a graphics data processing system 3, which interfaces with a host system via an interface 2. The system includes an embedded processing unit (EPU) 8 for controlling the overall function of the graphics processor and for interfacing with the host system. The also system includes a processing core 10 which processes the graphical data for output to the display screen via a video output interface 14. Local memory 12 is provided for the graphics system 3. Such a data processing can be connected for operation to a host system or could provide a standalone processing system, without the need for a specific host system. Ex. 1006, 4:21-5:6, Fig. 1. “The host system can be any type of computer system, for example, a PC 99 specification personal computer or a workstation.” *Id.*, 5:14-16. Stuttard’s system is clearly a computing system as known by a person skilled in the art.

120. It is also my opinion that Stuttard discloses a local memory coupled to the processing core. Indeed, local memory 12 is provided for the graphics system 3. Ex. 1006, 4:21-5:6, Fig. 1. This memory, which is connected to a primary bus

via memory interface units 121 is used to store various information for use by the graphics system 3. *Id.*, 7:11-16. Further, Stuttard describes a cache memory unit 1024 used to store instructions fetched for each thread by an instruction fetch unit 1023 of a thread manager 102. *Id.*, 11:3-10, Fig. 4.

121. Stuttard discloses a processing core 10 of the graphics data processing system 3. Ex. 1006, 4:21-5:6, Fig. 1. This processing core, which is connected to the secondary buses 6 for the transfer of data, and to the primary bus 4 for the transfer of instructions, provides the graphics processing capability of the system. The processing core 10 includes a number of control units: thread manager 102, array controller 104, channel controller 108, a binning unit 1069 per block and a microcode store 105. *Id.*, 7:25-8:7, 8:9-13, 8:17-10:3, Figs. 2, 3. The processing core 10 is thus the claimed “first processing unit.”

122. Stuttard discloses that a primary bus 4 and one or more secondary buses 6, which are coupled to the local memory 12 and the processing core 10. The primary bus is used for connection of latency intolerant devices whereas and the secondary buses are used for connection of latency tolerant devices. Ex. 1006, 6:6-19, Fig. 2. In Stuttard’s system, access to the primary bus 4 is controlled by a primary arbiter 41 whereas access to the secondary buses 6 by a pair of secondary arbiters 61. *Id.*, 6:21-27. The primary bus is connected to local memory 12, which is used to “store various information for use by the graphics system.” *Id.*, 7:11-16.

That is, the primary or secondary buses are also coupled to the processing core 10. This coupling can also be observed in Fig. 2 embodiment of Stuttard's computer system.

123. Stuttard discloses that the processing core 10 includes a number of controllers: thread manager 102, array controller 104, channel controller 108, a binning unit 1069 per block, and a microcode store 105. These control units control the operation of processing blocks 106 of the graphics processing system. Ex. 1006, 7:25-8:7. The thread manager 102 receives control signals from an embedded processor unit (EPU 8) to fetch instructions and subsequently provide them to the array controller 104 and the channel controller 108. The array and channel controllers 104 and 108 are connected to transfer control signals to the processing blocks 106 depending on the received instructions. *Id.*, 8:17-27. The array of processing elements is controlled to carry out a series of instructions in an instruction stream "thread." *Id.*, 10:7-17. "Each thread works cooperatively with other threads to perform a task or tasks." *Id.*

124. The thread manager 102 comprises an instruction fetch unit 1023, a cache memory unit 1024 for storing instructions fetched for each thread, a thread scheduler 1025, thread processors 1026, and a status block 1030. Ex. 1006, 11:3-10. Stuttard explains that ("Instructions for a thread are fetched from local memory or the EPU 8 by the fetch unit 1023 and supplied to the cache memory 1024 via

connecting logic.” *Id.*, 11:12-14. The thread manager 102 includes one thread processor 1026 for each thread. The thread processors 1026 control the issuance of core instructions from the thread manager 102. *Id.*, 11:27-12:7. “The purpose of the thread processor 1026 is to allow high level flow control to be performed for a thread, (such as looping and conditional branches), and to assemble instructions to be issued to the array controller 104 and channel controller 108.” *Id.*, 26:3-7. The thread scheduler 1025 works to ensure that each processor block 106 is kept busy as much as possible. *Id.*, 23:18-28. The array controller 104 comprises an instruction launcher 1041, which directs instruction information to either a PE instruction sequencer 1044 or a load/store controller 1045. *Id.*, 12:30-13:7.

125. The binning unit 1069, the channel controller 108, the array controller 104, and components of the thread manager 102 (including the cache memory unit 1024, the thread processors 1026, and the status block 1030) constitute the claimed “plurality of engines.” As I explained above, this plurality of engines is configured to receive from the instruction fetch unit 1023 and/or the thread scheduler 1025 (either of which can be the claimed “scheduling module”) associated with the embedded processor unit EPU 8 (which is the claimed “second processing unit”), a plurality of tasks. Alternatively, the plurality of engines is configured to receive from the EPU 8 (the claimed “scheduling module”) associated with a host system (the claimed “second processing unit”), a plurality of tasks. “The thread scheduler

can be disabled through the EPU interface. When the scheduler is disabled the EPU is able to control activation of the threads. For example, the EPU could start and stop the active thread, set the active thread pointer to a particular thread, and single step through the active thread.” See Ex. 1006, 25:19-24.

126. Stuttard teaches that the status block 1030 within the thread manager 102 receives status information 1036 from each of the threads. The status information is subsequently transferred to the thread scheduler 1025, which uses the status information to determine which thread should be active at any one time. Ex. 1006, 12:12-18. The thread manager 102 sends instructions to an instruction launcher 1041 within the array controller 104. This instruction launcher then looks up “specific instruction information” contained in an instruction table 1042, on which basis directs instruction information to either a PE instruction sequencer 1044 (which receives instruction information relating to data processing and handles instructions that involve data processing in the processor unit 1061a) or to a load/store controller 1045 (which receives information relating to data transfer operations and handles instructions that involve data transfer between the registers of the processor unit 1061a and the PE memory unit 1061c). *Id.*, 9:11-20, 12:30-13:7, 13:23-30, Fig. 5. “The PE instruction sequencer 1044 uses received instruction information to index a PE microcode store 105, for transferring PE microcode instructions to the PEs in the processing array.” *Id.*, 31:9-12. The PE

instruction sequencer 1044 and/or the load/store controller 1045 within the array controller 104 load the “state data” associated with [each task], e.g., data associated with each task that is present in the PE register file 1061b and/or the PE memory unit 1061c.

127. Alternatively, the channel controller 108 loads state registers 1103 associated with each task:

“Instruction decoder 1101 receives instructions from the channel controller 108 and triggers the state machine 1102 into operation. The state machine 1102 is the logic that sequences the other parts of the binning unit to perform a particular function such as reading or writing primitive pointers to or from external memory. The state machine 1102 may be implemented as several communicating state machines. Control signals to all other parts of the binning unit are not shown. The binning unit according to a set of internal registers 1103 that define the current binning context, that is the location of bin lists in external memory, the region to be rasterized next, the operation mode and so on. This set of ‘state’ registers 1103 is multiple ported to the channel controller 108, the block I/O bus and the EPU 8 (i.e., the registers have a number of ports that can be used simultaneously.”

Ex. 1006, 43:22-44:9, Fig. 7.

128. Stuttard therefore discloses a plurality of engines configured to receive, from a scheduling module associated with a second processing unit, a plurality of tasks and to load state data associated with each of the plurality of tasks.

129. Stuttard clearly teaches that its 8 parallel threads can be assigned to different tasks at the same time: “At any particular moment in time, each thread may be assigned to any one of a number of tasks. For example, thread zero may be assigned for general system control, thread 1 assigned to execute 2D (two dimensional) activities, and threads 2 to 7 assigned to executing 3D activities (such as calculating vertices, primitives or rastering).” Ex. 1006, 11:19-25. Stuttard also teaches that multiple threads can be simultaneously active and running on different processing blocks at the same time: “The thread processors 1026 control the issuance of core instructions from the thread manager so as to maintain processing of simultaneously active program threads, so that each the processing blocks 106 can be active for as much time as possible.” *Id.*, 11:29-33. Finally, as also stated earlier, the thread manager 102 loads status information associated with threads and sends them to processing blocks 106 via the array controller 104. *Id.*, 9:11-20, 11:27-12:7, 12:30-13:7, 13:23-30. Thus, Stuttard discloses a plurality of processing blocks 106, which constitutes a shader core. “A data processing apparatus comprises a SIMD (single instruction multiple data) array (10) of processing elements. The processing elements are operably divided into a plurality of processing blocks, the processing blocks being operable to process respective groups of data items.” *Id.*, Abstract; *see also id.*, 9:11-20, 17:28-18:8, Figs. 3, 8.

The processing blocks 106 receive the plurality of tasks from at least the thread manager 102 via the array controller 104. *Id.*, 10:7-17, 11:27-12:7, 12:30-13:7.

130. Stuttard therefore discloses a shader core configured to receive the plurality of tasks from at least one of the plurality of engines and to execute a first task from the plurality of tasks while executing a second task from the plurality of tasks based on respective state data associated with each of the first and second tasks.

B. Dependent Claim 16

131. Each processing block 106 (that comprises an array 1061 of processor elements in Stuttard, *see, e.g., id.*, 8:17-10:3) constitutes a “processing element.”

A plurality of processing blocks 106 is a plurality of processing elements.

Alternatively, each of the array 1061 of processor elements in each processing block 106 constitutes the claimed “processing element.”

132. Stuttard discloses that a first plurality of processing blocks or processor elements is configured to execute the first task based on a first set of state data associated with the first task; and a second plurality of process blocks or processing elements configured to execute the second task based on a second set of state data associated with the second task. *See* 15.e above. *See also* Ex. 1006, 17:28-18:8.

C. Dependent Claim 17

133. Stuttard discloses a plurality of FIFO buffers within the thread manager 102, one per thread. Ex. 1006, 11:3-10, 11:27-12:7, 23:3-11, Fig. 4.

D. Dependent Claim 18

134. Stuttard discloses pairing the graphics data processing system with latency intolerant devices. *Id.*, 6:6-1. Stuttard also discloses masking the latency of memory I/O through the use of different tasks. *Id.*, 10:7-31. Stuttard further discloses threads with different priority levels (e.g., high or low). *Id.*, 11:16-25, 23:18-28, 25:5-17, 38:1-16, 38:18-27. These disclosures at least render obvious that tasks would comprise low-latency and regular-latency tasks because latency was a concern of Stuttard, and such latency classification could have been achieved through priority designation.

E. Dependent Claim 19

135. Stuttard discloses general system control, 2D graphics, and 3D graphics tasks. Ex. 1006, 10:19-31, 11:16-25. Further, it is my opinion that a POSITA would know that in a computing system of scale and richness of computational resources as Stuttard's computing system, both graphics-processing and general-compute tasks can be executed. To the extent this is not considered an express disclosure, it is my opinion that a person of ordinary skill would have considered this limitation obvious in view of the knowledge of a person of ordinary skill in the art. For example, a person of ordinary skill in the art would have

considered general system control to be a type of general compute tasks and/or, at a minimum, find it obvious.

F. Dependent Claim 20

136. Stuttard discloses threads with different priority levels (e.g., high or low) and using priority of threads to determine order of execution. For example, “The threads are assigned priorities relative to one another. Of course, although the example described here has eight threads, any number of threads can be controlled in this manner. At any particular moment in time, each thread may be assigned to any one of a number of tasks. For example, thread zero may be assigned for general system control, thread 1 assigned to execute 2D (two dimensional) activities, and threads 2 to 7 assigned to executing 3D activities (such as calculating vertices, primitives or rastering).” Ex. 1006, 11:16-25. As another example, Stuttard discloses that because “[t]he switching from one thread to another involves a state saving and restoring overhead . . . the priority of threads is used to reduce the number of thread switches, thereby reducing the associated overheads.” *Id.*, 23:18-28. Further, Stuttard’s thread scheduler recalculates which thread should be active based in part on thread priorities. *Id.*, 23:35-24:6, 25:5-9, 25:11-17. Finally, Stuttard teaches that instructions from the thread manager 102 are pushed into three separate instruction FIFOs for low priority, high priority, and biner instructions. *Id.*, 37:28-38:8.

137. Based on the disclosure of Stuttard, it is my opinion that a POSITA would understand that a task would comprise one or more threads and that a high-priority task would give rise to high-priority threads. Conversely, once all high-priority threads associated with a task are completed based on their assigned priority levels, the task has received a high priority processing by the computing system.

XI. DECLARATION

138. In signing this declaration, I understand that this declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the USPTO.

139. I am aware that willful false statements and the like made by me in connection with this declaration are punishable by fine or imprisonment, or both (18 U.S.C. § 1001) and may jeopardize the validity of the Petition. All statements herein made by me of my own knowledge are true and all statements made on information and belief are believed to be true. I declare under penalty of perjury that the foregoing is true and correct.

Executed this 7th day of March, 2023 at Los Angeles, California.

Respectfully Submitted,

A handwritten signature in black ink, appearing to read "M. Pedram", is written over a horizontal line.

Massoud Pedram, Ph.D.