

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
U.S. Patent No. 8,854,381

**PATENT OWNER'S RESPONSE
PURSUANT TO 37 C.F.R. § 42.107(A)**

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. TECHNICAL OVERVIEW	3
A. Basic Graphics Processing	3
B. Single-Threading Versus Multi-Threading	3
C. Shader Programs.....	4
III. '381 PATENT.....	5
A. Parallel Engines	7
B. Engines Versus Shader Programs	8
C. Simultaneous Multi-Tasking	10
D. Challenged Claims	11
IV. PRIOR ART OVERVIEW	12
A. Wilt (U.S. Patent No. 8,151,095)	12
B. Jiao (U.S. Patent No. 8,319,774).....	15
C. Stuttard (International Publication No. WO 00/62182).....	18
V. CLAIM CONSTRUCTION	21
A. “task”	21
B. “engine”	23
C. “configured to”	25
D. “each of the plurality of tasks”	27
VI. PETITIONER HAS FAILED TO SHOW THAT ANY CHALLENGED CLAIMS ARE OBVIOUS UNDER ANY GROUND	27
A. Claim 19	27
1. Ground 1 Fails.....	29
a. Petitioner Did Not Allege That Ground 1’s “Single PPU” Theory Invalidates Claim 19	29
b. Ground 1’s “Multiple PPU” Theory Does Not Invalidate Claim 19	30
2. Ground 2 Fails.....	32
3. Ground 3 Fails.....	34
B. Claim 20	35

1.	Ground 1 Fails.....	36
2.	Ground 2 Fails.....	38
3.	Ground 3 Fails.....	38
C.	All Challenged Claims	40
1.	Ground 1 Fails For All Challenged Claims	40
a.	Wilt Does Not Disclose Simultaneously Executing Different Tasks.....	40
b.	The Multi-PPU Theory Does Not Disclose or Render Obvious the Claims	45
c.	Wilt’s PPU(s) Do Not Receive Tasks From The Alleged Engines Because The Alleged Engines Are Inside The PPU(s).....	47
d.	Rewriting Petitioner’s Theory to Contradict Petitioner’s Explicit Theory Is Improper.....	50
2.	Ground 2 Fails For All Challenged Claims	51
a.	Jiao’s Shaders Do Not Disclose or Render Obvious the “Plurality of Engines”	52
b.	Jiao’s Alleged Scheduling Module Is Associated with the First Processing Unit.....	58
c.	Jiao’s Alleged Shader Core Does Not Receive “The Plurality of Tasks”	59
d.	Jiao Does Not Disclose Receiving and Simultaneously Executing Tasks.....	61
3.	Ground 3 Fails For All Challenged Claims	63
a.	Stuttard Does Not Disclose Receiving and Simultaneously Executing Tasks.....	63
b.	The Alleged Engines Are Within the Alleged Shader Core.....	72
c.	The Alleged Scheduling Module Is Not “Associated With” The Second Processing Unit.....	72
d.	The Alleged Engines Are Not Engines.....	74
VII.	CONCLUSION.....	75

TABLE OF AUTHORITIES

	Page(s)
 Federal Cases	
<i>Aspex Eyewear, Inc. v. Marchon Eyewear, Inc.</i> , 672 F.3d 1335 (Fed. Cir. 2012)	24
<i>Ball Aerosol & Specialty Container, Inc. v. Ltd. Brands, Inc.</i> , 555 F.3d 984 (Fed. Cir. 2009)	27
<i>Chemours Co. FC, LLC v. Daikin Indus.</i> , 4 F.4th 1370 (Fed. Cir. 2021)	27
<i>In re Magnum Oil Tools Int’l, Ltd.</i> , 829 F.3d 1364 (Fed. Cir. 2016)	50
<i>Intellectual Ventures I LLC v. Altera Corp.</i> , 2013 WL 3913646 (D. Del. July 26, 2013)	25
<i>Koninklijke Philips N.V. v. Google LLC</i> , 948 F.3d 1330 (Fed. Cir. 2020)	50
<i>Mitek Sys., Inc. v. TIS America Inc.</i> , 2014 WL 3891237 (D. Del. Aug. 6, 2014)	24
<i>Oren Techs., LLC v. Proppant Express Invs. LLC</i> , 2021 U.S. App. LEXIS 21859 (Fed. Cir. July 23, 2021)	50
<i>SIPCO, LLC v. Abb, Inc.</i> , 2012 WL 3112302 (E.D.Tex. July 30, 2012)	25
<i>Speedtrack, Inc. v. Amazon</i> , 998 F.3d 1373 (Fed. Cir. 2021)	48
<i>TQ Delta, LLC v. Cisco Sys.</i> , 942 F.3d 1352 (Fed. Cir. 2019)	27

TABLE OF EXHIBITS

Exhibit	Description
2006	Declaration of John Hart Regarding U.S. Patent No. 8,854,381
2007	Deposition of Massoud Pedram (January 5 and 8, 2024)

I. INTRODUCTION

Petitioner Realtek has failed to prove that any of the challenged claims 15-20 of Patent Owner ATI's U.S. Patent No. 8,854,381 are unpatentable.

All Claims

All asserted grounds fail to invalidate any of the challenged claims for numerous independent reasons. For example:

Ground 1 fails because Wilt does not teach the following limitations required by each challenged claim:

- a shader core that executes a first task while executing a second task;
- plurality of engines separate from the shader core; and
- engines that receive tasks.

Ground 2 fails because Jiao does not teach the following limitations required by each challenged claim:

- engines configured to receive a plurality of tasks;
- a shader core that executes a first task while executing a second task; and
- a shader core that receives the claimed plurality of tasks.

Ground 3 fails because Stuttard does not teach the following limitations required by each challenged claim:

- a shader core that executes a first task while executing a second task;
- elements configured to receive and execute tasks; and

- a scheduling module associated with a second module unit.

Claim 19

In addition to the foregoing deficiencies, Petitioner failed to prove claim 19 unpatentable for other independent reasons.

Claim 19 depends from claim 15 and requires that (1) the “first task” executed by the claimed shader core is “a graphics-processing task”; and (2) the “second task” that the shader core executes *while* also executing the first task is “a general-compute task.” But all the Petition’s Grounds fail even to allege that the purported shader cores can execute these different types of tasks at the same time as claim 19 requires, much less prove it.

Claim 20

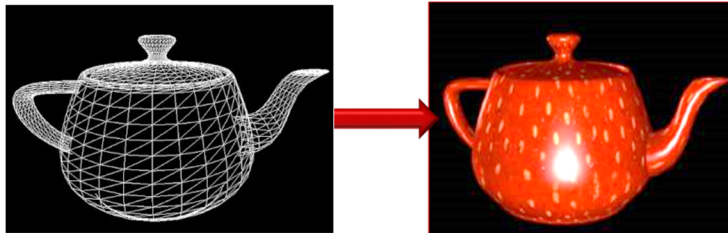
In addition to the foregoing deficiencies regarding, Petitioner failed to prove claim 20 unpatentable for other independent reasons.

Claim 20 depends from claim 15 and further requires that the shader core “is configured to receive the plurality of tasks based on a priority associated with each of the plurality of tasks.” But all Petition’s Grounds fail to identify in any of the references a single priority associated with a task, much less a task priority that affects how the shader core receives the tasks.

II. TECHNICAL OVERVIEW

A. Basic Graphics Processing

Graphics processing involves manipulating and rendering 3D graphical objects onto a 2D display screen. Ex. 2006 (Hart POR Decl.), ¶38. 3D objects (typically represented by “primitives” geometrical shapes such as triangles) are converted into a 2D image (typically comprising pixels) as shown below. *Id.* The 2D image represents a depiction of the 3D objects, rendered from the perspective of a given viewer, and shaded to convey the object’s surface material illuminated from a given light source. *Id.*



The conversion of a 3D shape consisting of a mesh of triangles into a photorealistic 2D image consisting of an array of pixels happens in a number of processing stages. Ex. 2006, ¶39.

B. Single-Threading Versus Multi-Threading

An ordinary “single-threaded” graphics processing system executes a single thread, proceeding instruction by instruction, in their original order, on one piece of data a time. Ex. 2006, ¶40. A multi-threaded processing system can improve upon single-threading by enabling the switch of the active thread being processed to

another thread. *Id.* A multi-threaded processing system does not enable the processing of multiple *tasks* (as claimed by the '381 Patent). *Id.*

Such multi-threaded systems have substantial disadvantages, though. Ex. 2006, ¶41. First, switching between threads requires context switching, which is time-consuming. *Id.*; Ex. 1001, 2:38-43, 46-56. Second, it requires the system to have additional local memory to hold data (the “context”) for an instruction that is in the middle of execution at the time of switch, so that progress is not lost and the system can later switch back and keep executing without starting from the beginning. Ex. 1001, 2:43-46.

C. Shader Programs

Shader programs, such as vertex shaders, geometry shaders, and pixel shaders, are pre-programmed software modules that receive data and operate on them. Ex. 2006, ¶34. The concept of a “shader” program dates back to Hanrahan and Lawson’s “A Language for Shading and Lighting Calculations,” which showed how Pixar’s non-real-time rendering system (which would take minutes or hours to render each frame) could include short custom programs called “shaders” that would be run pixel-by-pixel to compute the appearance of a surface. *Id.*; *see also generally* Ex. 2002 (Hanrahan). These are “preprogrammed shaders” written in code, compiled once, loaded into the processor, and then run. Ex. 2002, Sections 2, 4, Table 1; Ex. 2006, ¶¶42-45. While running, they process (or “evaluate”) data that they receive.

Ex. 2006, ¶43. Notably, shader programs are not intelligent processing units (such as the claimed shader core) that can receive, interpret, and execute “tasks.” *Id.*, ¶46. They do not receive the claimed “tasks” at all. *Id.*

III. ’381 PATENT

The ’381 Patent provides and claims an inventive approach to multi-tasking on a graphics processing unit (GPU), replacing a serial task entry point into the GPU with a parallel task entry point into the GPU, enabling simultaneous multi-tasking. Ex. 2006, ¶47.

Where other systems serially entered tasks into a shader core at a single entry point so that the shader core executed tasks one at a time, the ’381 Patent enabled multi-tasking, the execution of two or more different streams of processing tasks at the same time. Ex. 1001, 4:30-5:45; Ex. 2006, ¶48. This was achieved the novel introduction of a set of parallel engines (as shown, for example, as blocks 202A–E in Fig. 2 below) that provide multiple entry paths for tasks to reach the shader core. *Id.*; Ex. 1001, 4:30-5:45. Each parallel engine is configured to (i) receive, from a scheduling module (*e.g.*, an operating system (OS) scheduler), a plurality of tasks substantially in parallel with each other and (ii) load a set of state data associated with each of the plurality of tasks. *Id.*, Abstract, 3:3-7; Ex. 2006, ¶48.

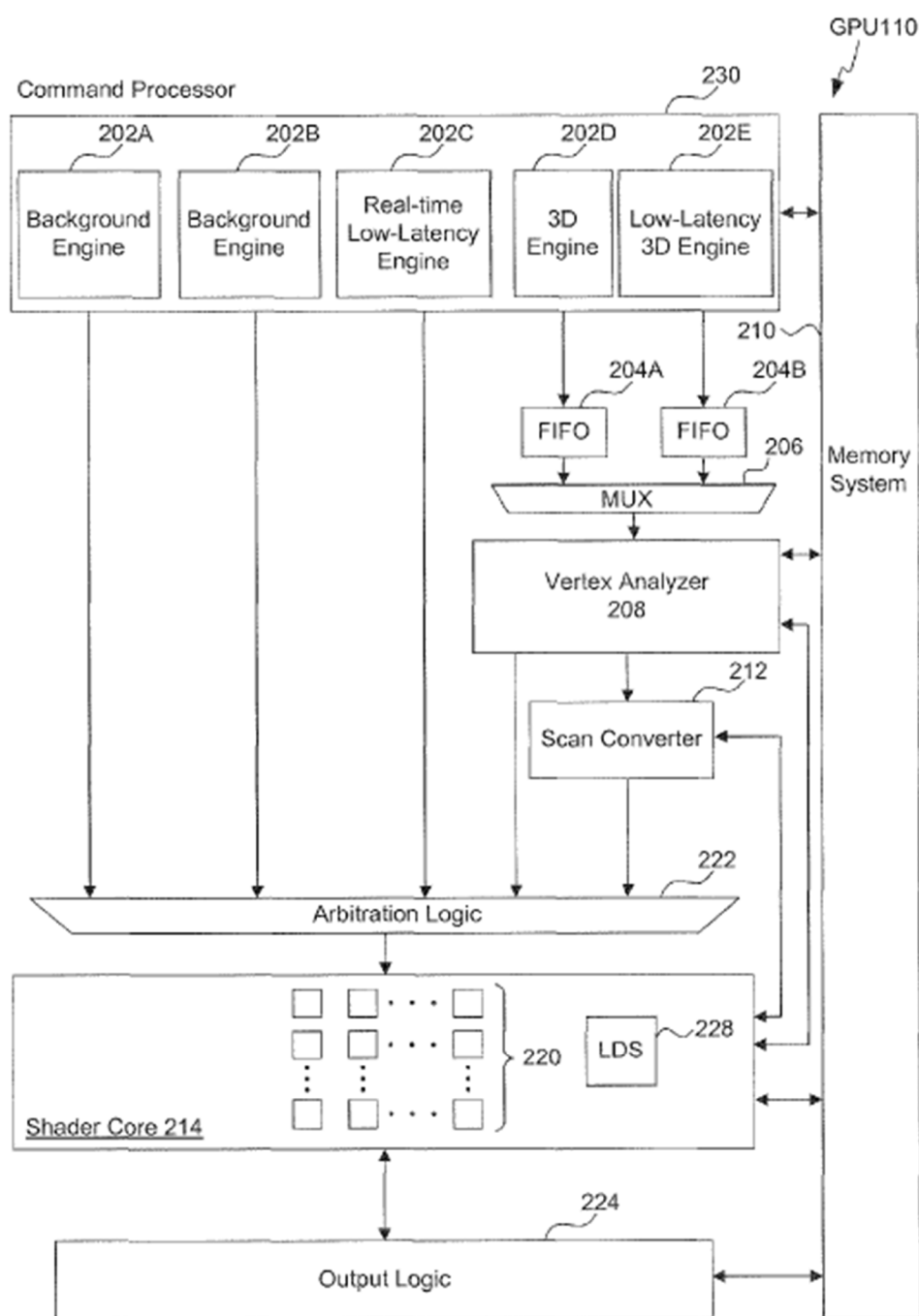


FIG. 2

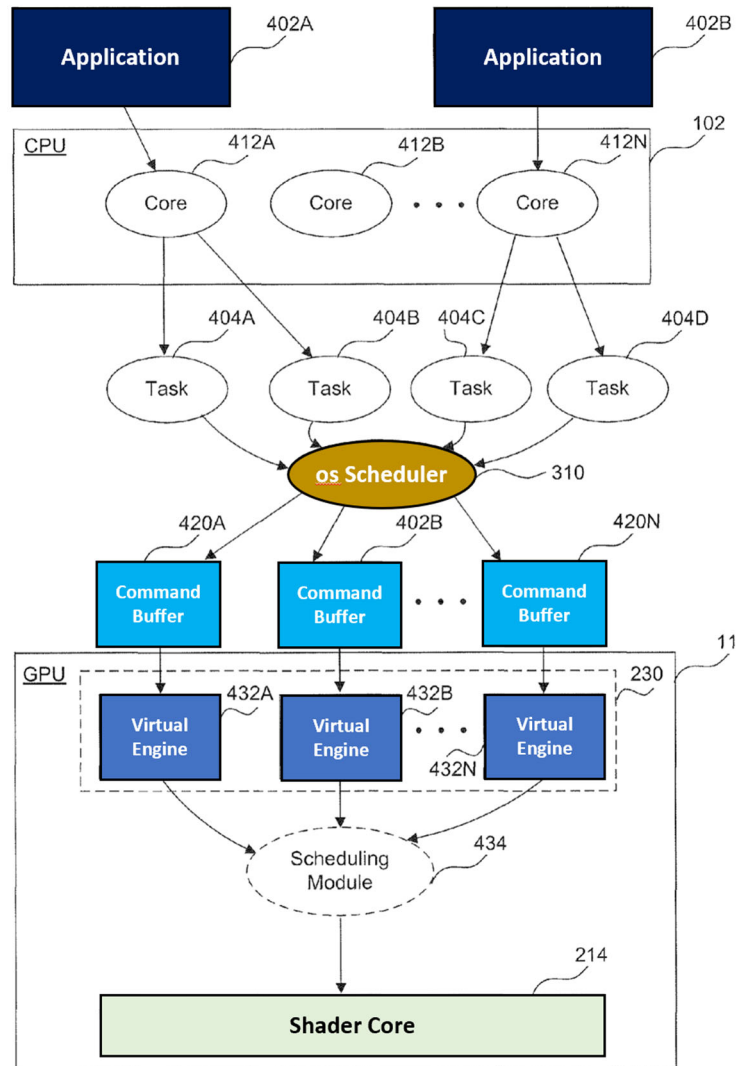
Ex. 1001, Fig. 2.

This approach also improved scheduling of tasks to the GPU, allowing more tasks to be provided to the shader core. Ex. 1001, 4:30-5:45; Ex. 2006, ¶49. Further,

latency is reduced because each of these engines can load state data associated with their respective tasks serves, unlike conventional systems that utilize a single functional unit that switches between different state data. Ex. 1001, 4:30-5:45; Ex. 2006, ¶49.

A. Parallel Engines

The '381 Patent distinguishes its parallel configuration of the engines from the conventional serial reception of tasks by the GPU as in the prior art. Ex. 2006, ¶50. For example, the '381 Patent explains that the engines function independently to receive tasks scheduled for execution by shared resources of the GPU. *Id.*; Ex. 1001, 4:48-51, 6:1-6. This is depicted in Figure 4:



Ex. 1001, Fig. 4 (annotated).

B. Engines Versus Shader Programs

The '381 Patent differentiates its parallel engines from other elements, including shader programs. Ex. 2006, ¶51. For example, as shown in Fig. 2 and as disclosed at 7:5-19, the graphics-processing tasks are received by, *e.g.*, the 3D engine 202D and the low-latency 3D engine 202E, which connect to certain graphics resources of the GPU, such as the vertex analyzer 208 and scan converter 212. *Id.*;

Ex. 1001, 7:5-19, Fig. 2. The '381 Patent describes that these graphics components may invoke shader programs to be run on the shader core: “Vertex analyzer 208 identifies shader programs associated with a graphics-processing and/or general-compute task and schedules when each shader program can be launched in shader core 214 based on input and output data that will be available.” Ex. 1001, 7:42-46. “Processing elements 220 may implement a plurality of different shader programs (*e.g.*, geometry shader, vertex shader, pixel shader, tessellation shader, or the like).” *Id.*, 8:34-41. By this point in the processing chain, the engines 202 have already received the tasks, and as explained in the '381 patent and in Section II.C above, the shader programs merely receive data and do not receive tasks: they are preprogrammed pieces of software that are invoked as part of carrying out a task, *i.e.*, they are constituent components of tasks. Ex. 2006, ¶51. They bear no similarity to the “engines” recited in the '381 Patent, nor do they perform any of the actions the engines are required to do in the challenged claims. *Id.*

The '381 Patent confirms in other places that “shader programs” are actually constituent components of tasks; they are not *engines*, which according to the claim language, receive tasks. Ex. 2006, ¶52. The '381 Patent teaches that in its terminology, “shader programs” are parts of “tasks”—they are not engines that *receive* tasks. More specifically, each thread (of a task) may execute a shader program. The '381 Patent states that “[t]o 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.**)”. Ex. 1001, 1:47-52. The ’381 Patent further states, “[t]o efficiently execute the ***various threads (e.g., shader programs)*** of a single graphics-processing task, the GPU includes an array of processing elements, called a shader core.” *Id.*, 1:58-61.

C. Simultaneous Multi-Tasking

As distinguished from single-threading and multi-threading described above, the ’381 Patent facilitated the ability to run tasks at the same time by partitioning the shader core (element 214 in Fig. 2 above) to run separate tasks in parallel. Ex. 2006, ¶53. “To enable shader core 214 to (substantially) simultaneously process tasks from the plurality of virtual engines of command processor 230, processing elements 220 of shader core 214 are partitioned in space and/or time.” Ex. 1001, 8:3-7. Further, the “spatial partitioning” of the shader core was implemented, in part, by organizing the shader core into multiple data-parallel subsets of processing elements. *Id.*, 8:8-10; Ex. 2006, ¶53.

By sharing available shader resources, the ’381 Patent explains how the plurality of engines serve to improve utilization of the GPU relative to conventional approaches which included a single path to the GPU’s execution resources (e.g., the shader core). Ex. 2006, ¶54. For example, the ’381 Patent discloses that “[a]s

illustrated in FIG. 3, command processor 230 includes a plurality of virtual engines that share resources of GPU 110.” *Id.*, 6:67-7:2. Moreover, the ’381 Patent discloses that “[s]haring the resources of the GPU to provide a plurality of virtual engines in accordance with embodiments of the present invention improves the use of the GPU resources, especially on large chips.” *Id.*, 5:29-32.

D. Challenged Claims

Claim 15 is the sole challenged independent claim, and recites the benefits described above:

15. A computing system comprising:

a memory;

a first processing unit; and

a bus coupled to the memory and the first processing unit, wherein the first processing unit comprises:

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*; and

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.

IV. PRIOR ART OVERVIEW

A. Wilt (U.S. Patent No. 8,151,095)

Wilt (Ex. 1004) is directed to improvements in utilizing context data by processing units in a parallel processing subsystem. Ex. 2006, ¶55. Specifically, Wilt explains that conventional systems may include multiple parallel processing units (PPUs) “that may be configured to beneficially provide a high volume of computational throughput that is impractical to achieve with a single processing unit.” Ex. 1004 (Wilt), 1:11-14.

Figure 1 shows an overview of the system:

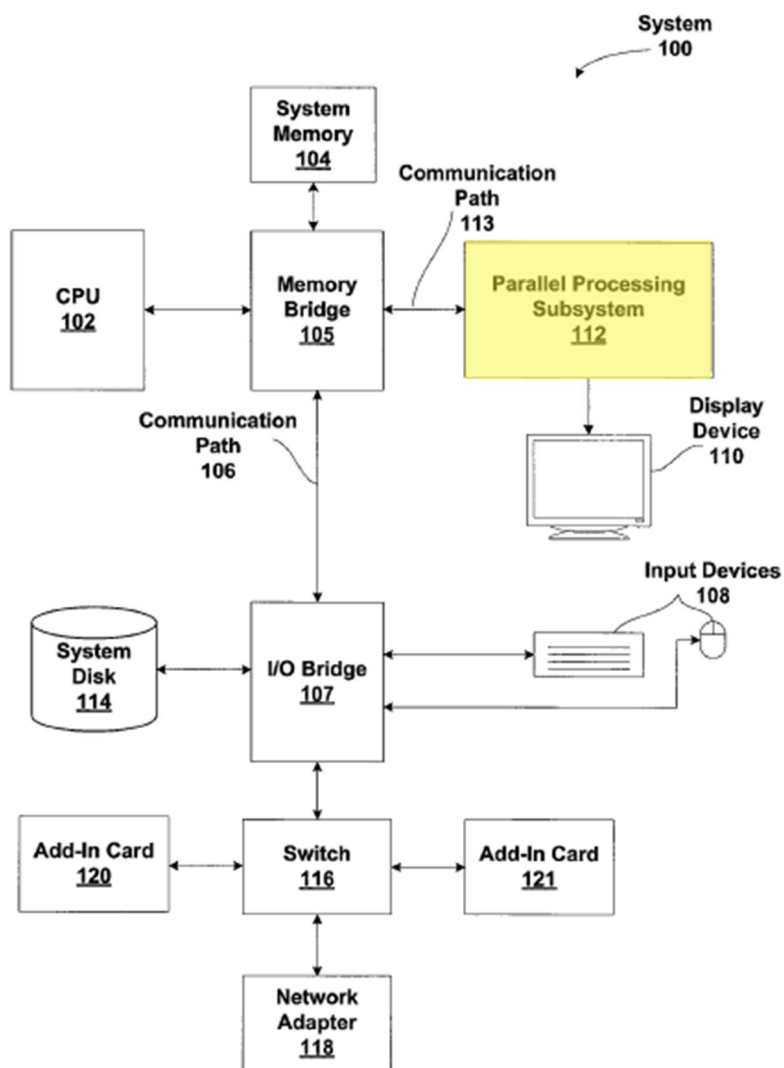


Figure 1

Ex. 1004, Fig. 1 (annotated). Figure 2 provides more detail on the parallel processing subsystem:

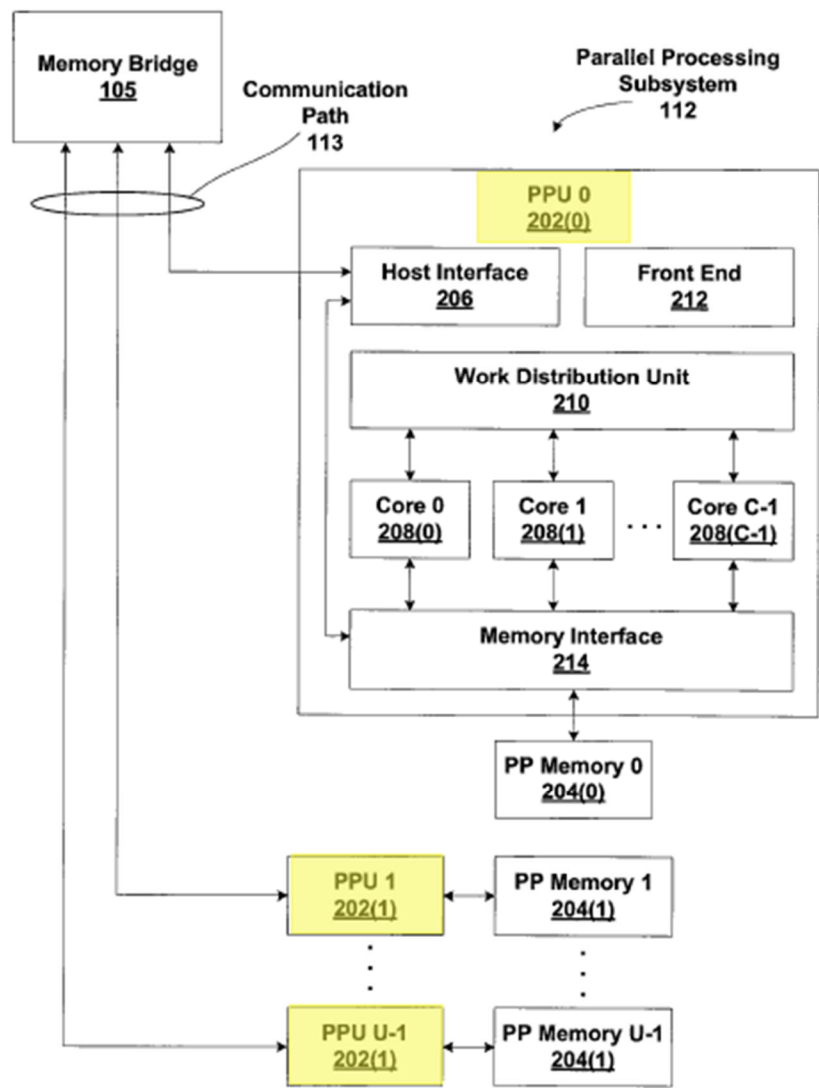


Figure 2

Ex. 1004, Fig. 2 (annotated). Figure 3 is further detail on the PPU 202:

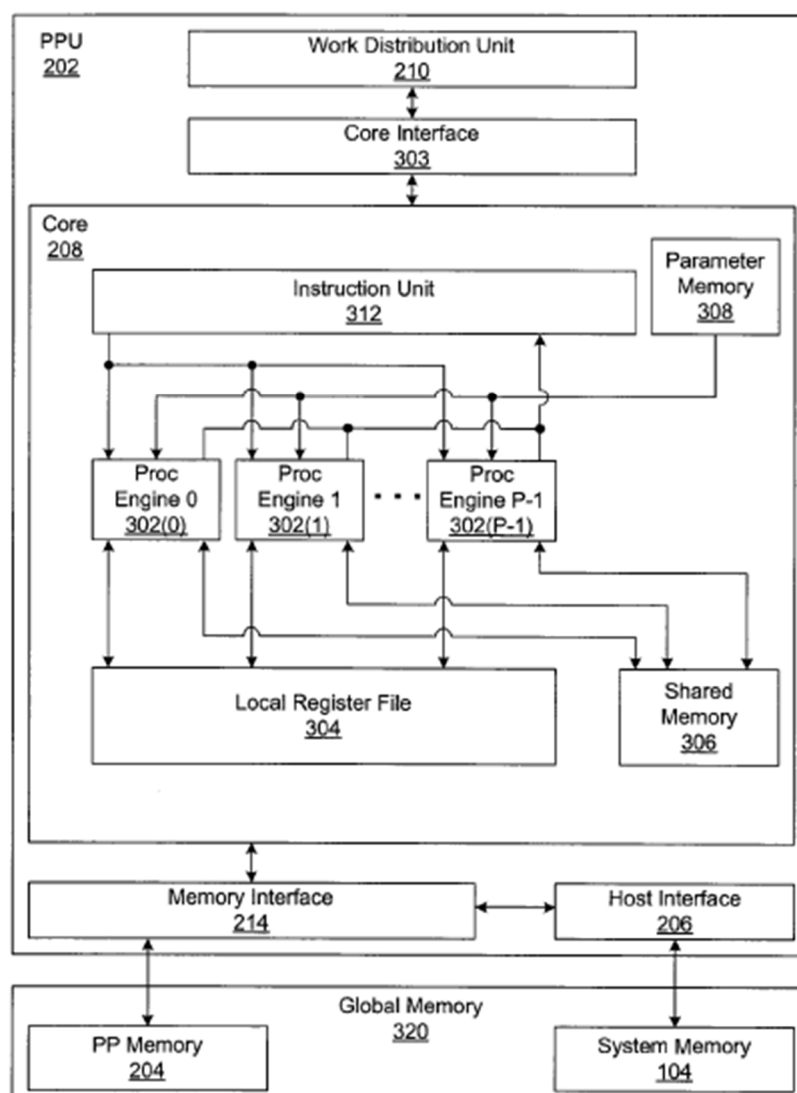


Figure 3

Ex. 1004, Fig. 3.

B. Jiao (U.S. Patent No. 8,319,774)

Jiao (Ex. 1005) is directed to a graphics processing system that is intended to address challenges in updating constants in a constant buffer, and the associated latencies encountered during a constant flush operation and the loading of new constant data. Ex. 2006, ¶57.

Figure 1 of Jiao shows the GPU in the context of the larger architecture:

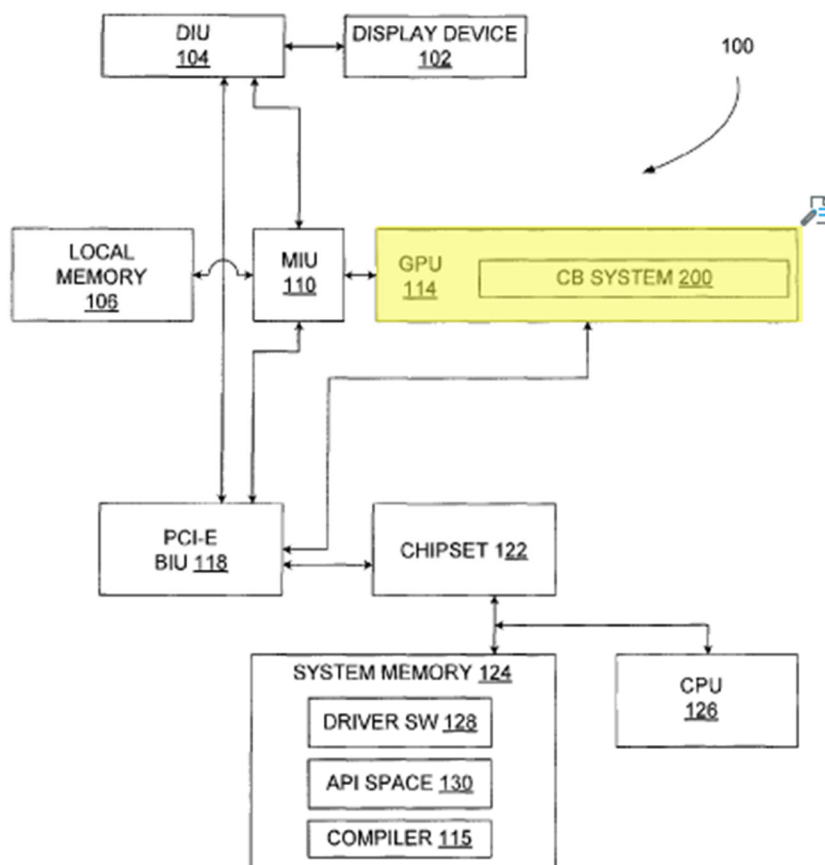


FIG. 1

Ex. 1005, Fig. 1 (annotated).

Figure 2 of Jiao is a high level diagram of the graphics processor 202 within GPU 114:

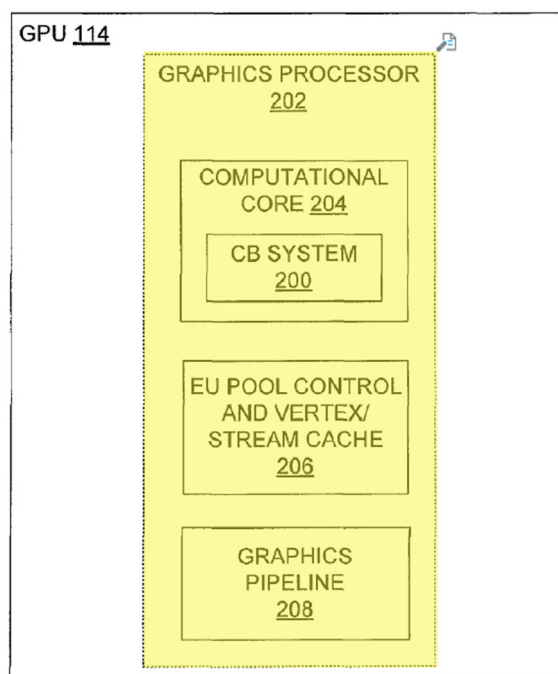


FIG. 2

Ex. 1005, Fig. 2.

Figure 4 gives more detail of graphics processor 202. Ex. 2006, ¶60. The key components are the Execution Units (EUs) 420 and EU Pool 414, the shader programs 304, 306, and 310, the EU Pool control 206, and the command stream processor 406. *Id.*

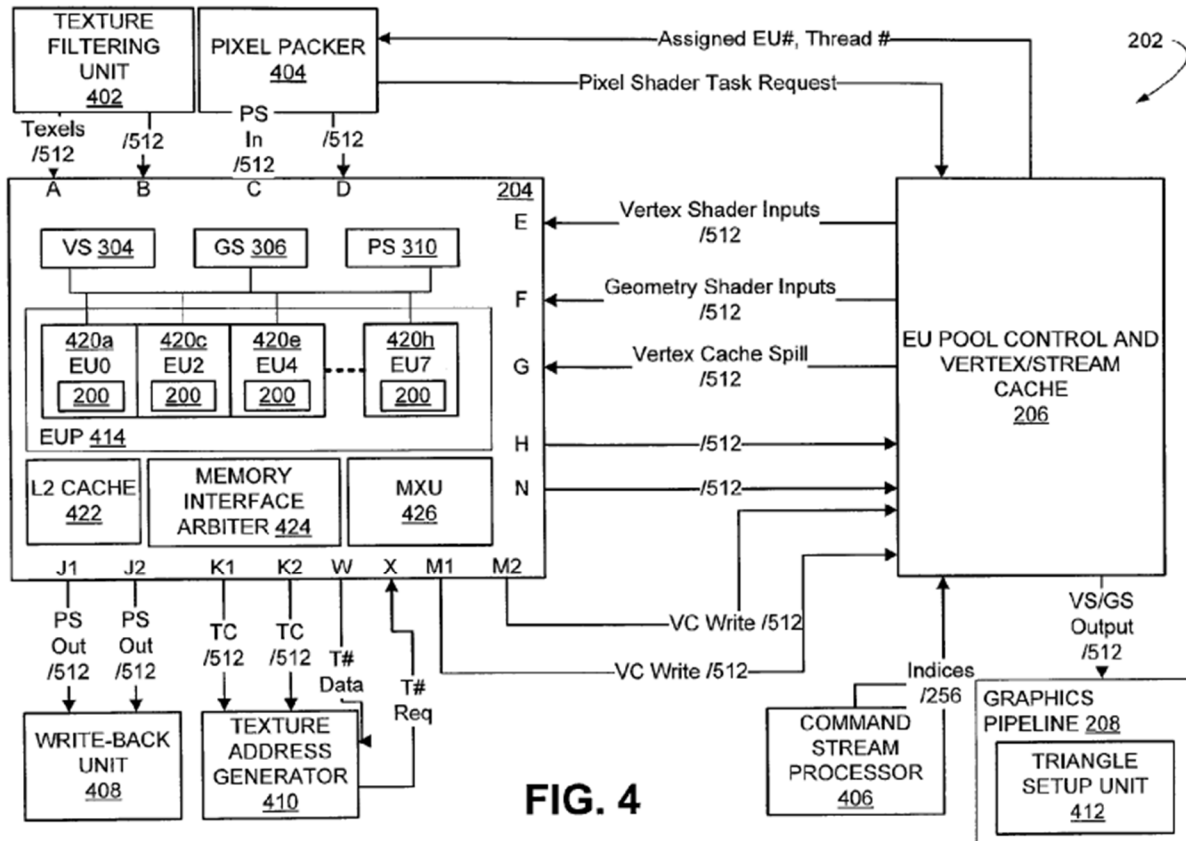


FIG. 4

Ex. 1005, Fig. 4. The shader programs — vertex shader 304, geometry shader 306, and pixel shader 310, perform pre-programmed actions on data, such as object shapes and vertex and pixel locations, that they receive during runtime. Ex. 2006, ¶60.

C. Stuttard (International Publication No. WO 00/62182)

Stuttard (Ex. 1006) discloses an allegedly improved “single-instruction, multiple data” (“SIMD”) architecture that divides the processing elements into processing blocks. Ex. 1006, 2:1-24; Ex. 2006, ¶61.

Figure 1 gives an overview of the architecture of Stuttard:

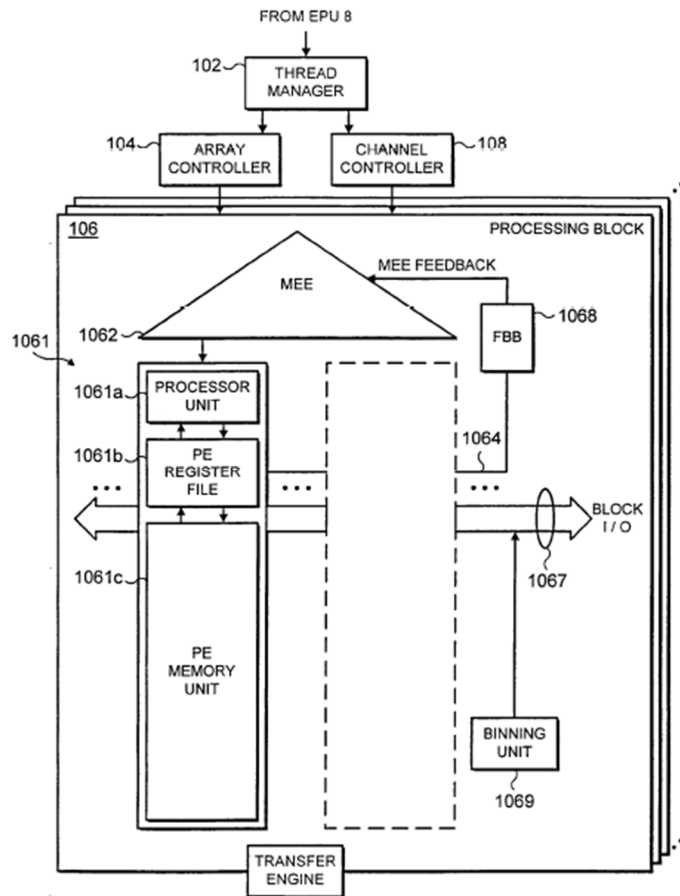


FIG. 3

Ex. 1006, Fig. 3. “The thread manager 102 is connected to receive control signals from the EPU 8. The control signals inform the thread manager as to when instructions are to be fetched and where the instructions are to be found. The thread manager 102 is connected to provide these instructions to the array controller 104 and to the channel controller 108. The array and channel controllers 104 and 108 are connected to transfer control signals to the processing blocks 106 dependent upon the received instructions.” Ex. 1006, 8:17-27. The processing core includes processing blocks 106, which comprise an array 1061 of processor elements (PEs),

a mathematical expression evaluator (MEE), and a binning unit. *Id.*, 8:15-9:3; Ex. 2006, ¶63.

Critically, the entirety of Stuttard can only operate on one instruction at a time, and one “thread,” or “instruction stream” over a period of time, as explained more fully below with respect to Ground 3. Ex. 1006, 4:17-23, 4:25-29, 10:20-24, 11:33-35; Pet., 57 (referring to “thread scheduler 1025, which uses the status information to determine which *thread* should be active *at any one time*.”), 62 (“Stuttard’s thread scheduler recalculates which *thread* should be active”); Ex. 2006, ¶64.

V. CLAIM CONSTRUCTION

Several terms were previously construed by the Board and at the ITC. Ex. 2006, ¶24. In addition, the following terms should be construed as follows:

A. “task”

Petitioner adopted the ITC construction of this task: “a unit of processing work or portion of an application to be executed.” Pet., 8. But an additional clarification that Petitioner and its expert substantively agree with is needed: a thread is not a task. Pet., 41; Ex. 1003 (Pedram Decl.), ¶102. The reason is simple: as a POSITA would know, and as the ’381 Patent states, a task corresponds more than one thread, which is why Petitioner and its expert have repeatedly stated threads are not tasks. Ex. 2006, ¶25.

The only expert to address this point is Dr. Hart, and he states that a POSITA would know that a task is “a unit of processing work decomposable into two or more related threads.” Ex. 2006, ¶¶25-29. A POSITA would understand that tasks are a higher level construct than threads, which are what are executed by the GPU. Ex. 2006, ¶¶28-29.

Petitioner and its expert, Dr. Pedram, agree that “[t]hreads are not tasks.” Ex. 1003, Par. 102; Pet., 41. They likewise differentiate between tasks and threads, and both give multiple examples of complex, multi-thread tasks. Pet., 41-42; Ex. 1003, ¶102; Ex. 2007, 10:14-23, 13:9-25 (tasks have “[cor]responding threads”). Petitioner provides 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 x 1080 LCD Display. Pet., 42 (citing Ex. 1003, ¶102). This parent task is ultimately decomposed into a plurality of threads (*e.g.*, vertex and covered pixel generation) and shader programs (*e.g.*, vertex processing, lighting, line segment processing, pixel processing, color and texture mapping). *Id.*; Ex. 2006, ¶¶28-29. The threads, when executed by the GPU, achieve the goal of the parent task, an example of which is Petitioner’s “draw a triangle” described above. *Id.*

The ’381 Patent states the same thing: “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.” Ex. 1001, 1:49-57 (emphasis added). Thus, the claimed task corresponds to *a plurality* of different threads. Ex. 2006, ¶¶26-28. Throughout the entirety of the ‘381 patent, it describes a task as comprising multiple threads, referring, for example, to “the threads of the first task,” and “the threads of the more-important (e.g., low-latency) task.” Ex. 1001 at 2:20- 37.

Therefore, “task” should be construed as “a unit of processing work decomposable into two or more related threads.”

B. “engine”

Properly construed, each claimed engine must be configured to independently (i) receive, from a scheduling module, at least one task, and (ii) load state data. Ex. 1001, Abstract, 2:10-19, 3:3-7, 4:48-51; 10:25-41, Claim 15; Figs. 2-4; Ex. 2006, ¶¶30-31.

These requirements are explicitly recited in the claims themselves and are confirmed by the specification. Ex. 2006, ¶32. Claim 15 requires “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[.]” Ex. 1001, Claim 15. Thus, a POSITA reading claim 15 would understand that any given engine of the plurality of engines must perform both of the claimed functions of (i) receiving a task from a scheduling module and

(ii) loading the state data associated with the task received by the engine. *See, e.g., id.*; Ex. 2006, ¶¶31-32. These requirements are reflected throughout the entirety of the specification, including, for example, Figure 4, which confirms and shows that the sending and receiving functions are performed for a specific task by a single engine, and are not distributed among different engines or functional blocks. *Id.*, ¶33.

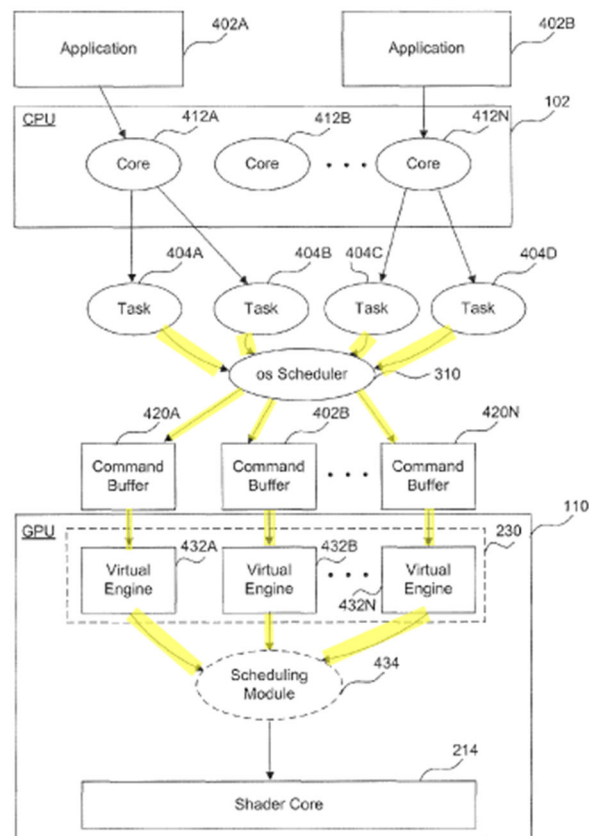


FIG. 4

Ex. 1001, Fig. 4 (annotated, where each path represents a task); *see also id.*, 9:34-36;6:65-7:19, 9:47-49, 10:25-39, Figs. 2, 3A, 3B. Furthermore, these two

requirements are supported by the '381 Patent's disclosure and depiction of the engines, and their receipt of tasks in a parallel configuration, which a POSITA would understand to mean that each engine loads the state data for whichever task it receives. *E.g., id.*, 4:48-51, 6:1-6; Ex. 2006, ¶34. For example, the '381 Patent expressly provides that each engine must function independently:

Each virtual engine is configured to receive data-parallel processing tasks (e.g., graphics-processing tasks and general-compute tasks) and independently execute these tasks on the single shader core.

Ex. 1001, 4:48-51.

C. “configured to”

The Federal Circuit has held that the phrase “configured to” means “designed to” or “made to,” which is narrower and more specific in scope than “capable of” or “suitable for.” *Aspex Eyewear, Inc. v. Marchon Eyewear, Inc.*, 672 F.3d 1335, 1349 (Fed. Cir. 2012). A claim reciting an apparatus or system that is “configured to” perform a specified function is not satisfied if the system is merely “capable of” the recited function, which would eliminate any meaningful limit to the words “configured to,” but means that it is specifically “arranged or set-up to perform a specified function,” “set up for operation especially in a particular way,” or “actually programmed or equipped with software to” perform the recited function. *Mitek Sys., Inc. v. TIS America Inc.*, 2014 WL 3891237, at *2 (D. Del. Aug. 6, 2014);

Intellectual Ventures I LLC v. Altera Corp., 2013 WL 3913646 (D. Del. July 26, 2013); *SIPCO, LLC v. Abb, Inc.*, 2012 WL 3112302, at *11 (E.D.Tex. July 30, 2012).

In the context of claim 15, the claim language “a shader core configured to ...” means it is insufficient that the shader core is simply “capable of” executing a first task and a second task simultaneously. Ex. 2006, ¶36. Rather, the shader core must be specifically programmed or designed to cause a first task and a second task to be executed simultaneously – and not just any two tasks, but a first task and a second task from *the same plurality of tasks* received by the plurality of engines. *Id.*

Indeed, that is exactly what Petitioner conceded in the ITC, where it stated that the claim language requires “the design of an apparatus to perform an action and not merely for the action to occur some of the time.” Ex. 2003 (ITC Init.Det.), 97-98; Ex. 2004 (Pet. Post-Hearing Br.), 67-68. Further, as Petitioner argued in the ITC, because the claim requires the system to be “configured to” execute a graphic task while executing a general compute task, it is insufficient to point to “the result of chance or probability, which only requires capability.” *Id.*, 67. As Petitioner argued there, “configured to” refers to the system’s “design . . . not its use at any given point in time.” *Id.* Petitioner argued that “the shader core must be specifically programmed or designed to cause a first task and a second task to be executed simultaneously—and not just any two tasks, but a first task and a second task from the same plurality of tasks received by the plurality of engines. If a first task and a

second task from a plurality of tasks are occasionally executed at the same time, and it happens as a matter of probability or chance, this is nothing more than a shader core that is capable of executing a first task and a second task simultaneously.” *Id.*, 67-68.

D. “each of the plurality of tasks”

In the ITC, Petitioner “assert[ed] that each individual engine ‘must be configured to load state data for each of the pluralities of tasks that are received.’” Ex. 2003, 99-100; Ex. 2004, 73-76. Patent Owner agrees that each engine must be configured to load state data, although does not agree that each item of state data must correspond to a plurality of tasks, *i.e.*, more than one task, as was rejected in the Initial Determination (“ID”) in the ITC the parties are currently litigating. Ex. 2003, 99-100; Ex. 2006, ¶37.

VI. PETITIONER HAS FAILED TO SHOW THAT ANY CHALLENGED CLAIMS ARE OBVIOUS UNDER ANY GROUND

A. Claim 19

Petitioner alleged claim 19 was invalid in the earlier ITC Investigation, but the ITC rejected Petitioner’s defense. Petitioner has likewise failed to meet its burden here.

Claim 19 depends from claim 15 and requires that (1) the “first task” executed by the claimed shader core is “a graphics-processing task”; and (2) the “second task”

that the shader core executes *while* also executing the first task is “a general-compute task.”

Ground 1 relies on Wilt for these limitations, but the Petitioner failed even to allege—never mind prove—that Wilt discloses a shader core that performs these different types of tasks *at the same time* as required by claim 19.

Grounds 2 and 3 fall ever shorter. Petitioner failed to identify in either Jiao (Ground 2) or Stuttard (Ground 3) a single instance of “a general-compute task” executed by the alleged shader core, much less one that is executed by that shader while it also executes a graphics processing task as claim 19 requires.

Also, Petitioner’s single reference obviousness for grounds 2 and 3 fail to repair deficiencies with an explicit analysis that provides reasons why a POSITA would be motivated to modify the reference, and why there is a reasonable expectation of success. *Ball Aerosol & Specialty Container, Inc. v. Ltd. Brands, Inc.*, 555 F.3d 984, 993 (Fed. Cir. 2009). Nor did Petitioner’s expert provide sufficient evidence supporting his obviousness opinion. *See, e.g., TQ Delta, LLC v. Cisco Sys.*, 942 F.3d 1352, 1358 (Fed. Cir. 2019). Thus, Petitioner’s arguments cannot support obviousness. *Chemours Co. FC, LLC v. Daikin Indus.*, 4 F.4th 1370, 1377 (Fed. Cir. 2021).

1. Ground 1 Fails

a. Petitioner Did Not Allege That Ground 1's "Single PPU" Theory Invalidates Claim 19

Petitioner's Ground 1 forwards two alternate theories for identifying the claimed shader core in Wilt. Under its "Single PPU Theory," Petitioner argues that a single PPU is the claimed shader core. Pet., 26, 28.

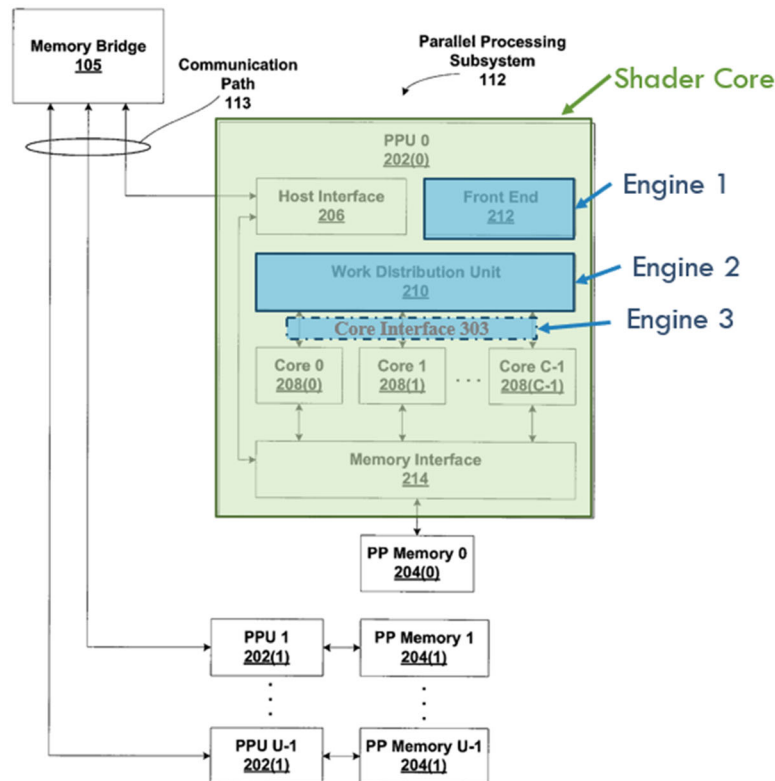


Figure 2

Ex. 1004, Fig. 2 (annotated with Single PPU theory).

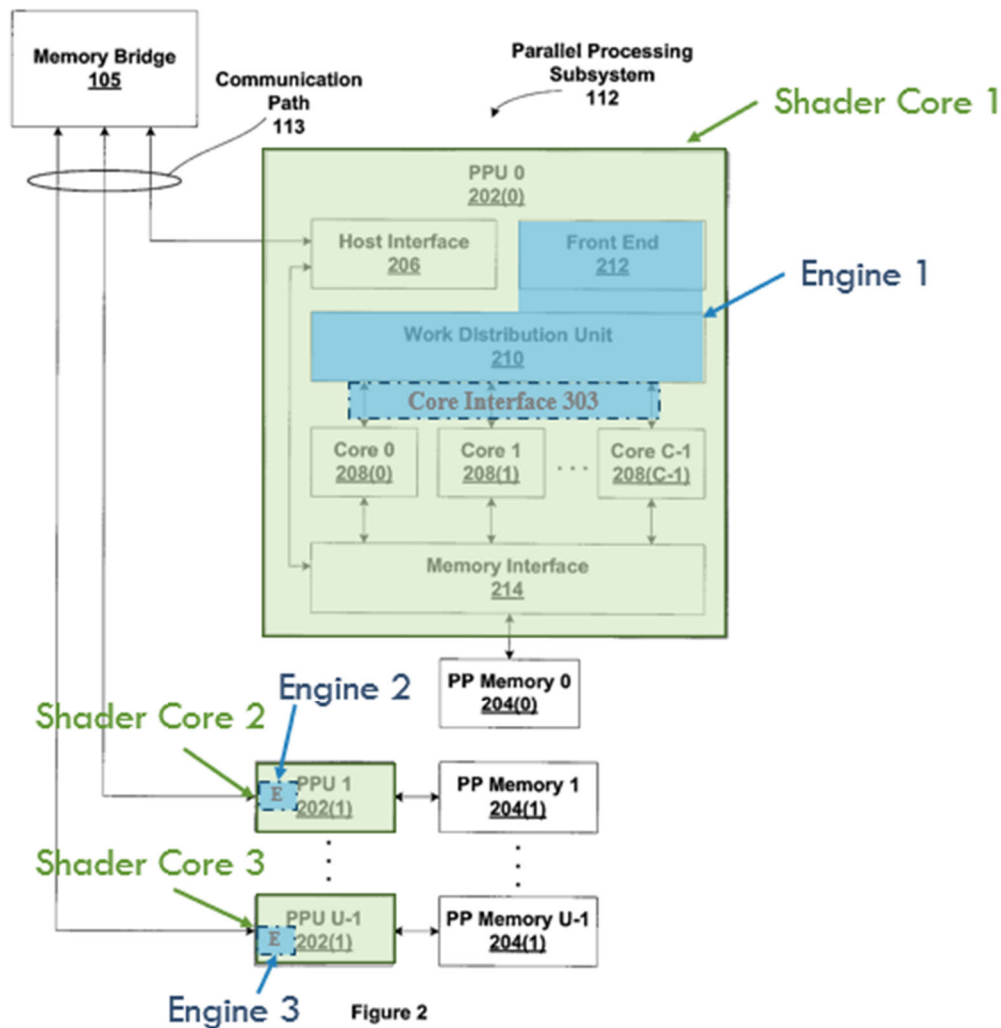
But Petitioner does not allege that claim 19 is unpatentable under this theory, never alleging that a single PPU (the alleged shader core) can execute both graphics processing and general-compute tasks, much less that it can execute one task type

while executing the other type as required by claim 19. Ex. 2006, ¶83. At most, the Petition alleges Wilt discloses “PPUs for graphics processing and PPU for general-purpose processing.” Pet., 35. Even if true, this would not meet the limitations of claim 19 under the “single PPU” theory and the Petition does not argue otherwise. Ex. 2006, ¶83.

And even if Petitioner now tried to (improperly) expand the scope of the IPR beyond the allegations of the Petition, Wilt in fact does not disclose a single PPU which executes a graphics-processing task while executing a general-compute task. Ex. 2006, ¶84. At best, Wilt discloses only PPU that are exclusively for graphics processing or exclusively for general-compute PPU. *Id.*; Ex. 1004, 4:55-58.

b. Ground 1’s “Multiple PPU” Theory Does Not Invalidate Claim 19

Ground 1’s second theory is its “Multi-PPU Theory,” where Petitioner argues that all the Wilt PPU are combined to form a single shader core. But this theory similarly fails because although the Petition alleges that the Mutli-PPU shader core can perform the two different types of tasks, it does not allege, much less prove, that it can do one *while* also doing the other, as claim 19 requires. Ex. 2006, ¶85.



Ex. 1004, Fig. 2 (annotated with Multi-PPU theory).

Nor could the Petitioner reasonably do so, because Wilt discloses only that its system is capable of executing what Petitioner terms general-compute threads, but not that they are executed *while* executing graphics-processing threads. Ex. 2006, ¶86. Although Wilt states that some PPUs can be used as graphics processors and some can be used for general-purpose computations, it does not disclose sending both types of tasks to its PPUs at the same time, let alone that both types of tasks

would be executed at the same time by the PPUs. *Id.*; Ex. 1004, 4:46-60. Nor has Petitioner made such argument.

Furthermore, Petitioner must prove that Wilt's system is actually configured to (*i.e.*, “programmed to” or “designed to”) execute a graphics-processing task while executing general-compute task. Ex. 2006, ¶86. As Petitioner previously argued in the ITC, such execution occurring by happenstance is not enough to show the system is “configured” to do so. *See* Section VI.C.1.a. Petitioner has not even attempted such a showing here.

2. Ground 2 Fails

Petitioner has not identified a single Jiao disclosure describing that its purported shader core can execute general-compute tasks, much less executing them *while* executing a graphics-processing task as claim 19 requires. Ex. 2006, ¶118. Accordingly, Petitioner relies only on obviousness grounds to argue that Jiao invalidates claim 19. Pet., 51. But this argument fails for at least two reasons: (1) the tasks that Petitioner argues would be obvious to execute with “graphics-processing” tasks are *not* the required “*general-compute*” tasks; and (2) there is no evidence that it was obvious or even possible for Jiao's purported shader core to execute these (wrong) tasks while also executing “graphics-processing” tasks. Ex. 2006, ¶¶119-123.

First, Petitioner’s obviousness theory does not identify or rely on any “general-compute” tasks. *Id.*, ¶121. Petitioner merely cites to Jiao’s statement that “arithmetic logic units and other units corresponding to functionality pertaining to instruction decoding, operand fetch, branch computation, instruction execution, shift/logic, load/store, among other processing functionality or logic” (Ex. 1005, 11:4-8) and also programmable-execution units discloses general-compute tasks. But neither Petitioner nor its expert explained how this disclosure relates to the ability to execute general-compute tasks, or why this functionality would not be provided for the purpose of executing graphics processing tasks (which are the subject of Jiao), rather than general-compute tasks (which are not). Ex. 2006, ¶¶121-122. Nor does Jiao provide any such explanation. *Id.* Indeed, in the ITC, the ALJ noted that Jiao is *not* set up to handle general compute tasks given that its only shaders are only vertex, geometry, and pixel shaders used in *graphics* processing. *Id.*

Dr. Pedram further confirmed that the alleged tasks generated by these shaders are not alleged to be general-compute tasks. *See* Ex. 2007 (Pedram IPR Depo. Tr.), 118:23-119:5.

Second, in claim 19, the shader core must execute the claimed general-compute tasks. But as Dr. Pedram confirmed, in Jiao those alleged tasks are carried out by the EUDP, not the computational element 420 that comprise the alleged

shader core. *See* Ex. 2007, 118:23-120:11 (“If it’s general-compute task, it will be dispatched to EUDP to perform it . . . Q. Does Jiao ever state that EUDP 514 can comprise the computational elements 420? A. No.”). There is no allegation, much less disclosure in Jiao, that the alleged general-compute tasks are executed by the shader core, as Dr. Pedram agrees. Ex. 2006, ¶ 121.

Accordingly, the ALJ in the 1318 Investigation correctly found that Petitioner failed to prove Jiao rendered this claim obvious under the same arguments presented here because Petitioner failed to show that the EUs of Jiao could function in the way disclosed by claim 19, or could be made to do so with modification. Ex. 2003, 161-162. Further, there is no motivation for a POSITA to use Jiao for compute tasks alongside graphics rendering tasks. *Id.*

3. Ground 3 Fails

Ground 3 relies on Stuttard’s disclosure of “general system control” tasks as the claimed “general-compute” tasks. Pet., 60-61. But general ***system control*** tasks are not general-***compute*** tasks. Ex. 2006, ¶¶166-167. The only similarity is that both types of tasks have the word “general” in their names, which does not relate to the task type. *Id.*

The ’381 Patent describes general-compute tasks as “mathematical algorithms, physics simulations, or the like.” Ex. 1001, 9:24-26; Ex. 2006, ¶167. As Petitioner’s expert testified, “general-compute task” must be “involve[]

computation,” and gave examples such as “addition, subtraction,” and methods of performing mathematics in computers. Ex. 2007, 114:1-12. He also conceded that general system control tasks are not necessarily general-compute tasks unless they specifically involve computation operations. Ex. 2007, 147:2-148:10.

But Stuttard does not describe its system control tasks as general-compute tasks or as involving computation. Ex. 2006, ¶167. Indeed, the only exemplary system control tasks described in Stuttard and upon which the Petition relies are handling data input/output (“I/O”) operations, such as transferring data to and from memory. Ex. 1006, 10:23-31. Petitioner’s expert admitted that these types of tasks are *not* general-compute tasks because they are not computational in nature. Ex. 2007, 148:11-149:12, 151:1-12 (“[By themselves, if you just say, ‘I went to memory -- external memory through the DDR interface to bring in some data item and write it in the shared memory or in the local memory or some processing unit in my system,’ the fact that you went there and you brought it in, *that’s not a computational task*, no. It’s not.”).

B. Claim 20

Petitioner alleged claim 20 was invalid in the earlier ITC Investigation, but the ITC rejected Petitioner’s defense. Petitioner has likewise failed to meet its burden here.

Challenged claim 20 depends from claim 15 and further requires that the shader core “is configured to receive the plurality of tasks based on a priority associated with each of the plurality of tasks.” But the Petition’s Grounds 1, 2, and 3 fail to identify in any of the references a single priority associated with a task, much less a task priority that affects how the shader core receives the tasks:

- Ground 1’s Wilt does not disclose priorities at all;
- Ground 2’s Jiao discloses priorities only for state data, not for tasks; and
- Ground 3’s Stuttard similarly does not disclose priorities for tasks, it only discloses priorities for threads, which Petitioner admits are not tasks.
- For all grounds, Petitioner’s single reference obviousness arguments are insufficient. *See* Section VI.A, *supra*.

1. Ground 1 Fails

Ground 1 relies on Wilt for claim 20’s limitations, but Wilt does not even mention task priorities. Accordingly, Petitioner resorts to a series of (incorrect) inferences, arguing that Wilt’s reference to “popping” a PPU context is an indirect disclosure of what it calls “context swapping,” and that this unmentioned “context swapping” in turn indirectly discloses that the PPUs are “configured to receive the plurality of tasks based on a priority associated with each of the plurality of tasks.” Pet., 35. Petitioner fails to meet its burden for numerous reasons. Ex. 2006, ¶¶88-93.

First, claim 20 requires task priority, not context swapping. *Id.*, ¶88.

Second, to the extent Petitioner is arguing that “context swapping” inherently teaches or makes obvious the claimed task priorities, Wilt mentions neither task priorities nor context swapping, and provides that its PPU contexts pertain to CPU threads, not GPU tasks. Ex. 2006, ¶89.

Third, even if Wilt did describe “context swapping,” the practice of context swapping does not necessarily involve or require the use of task priorities, to affect how the purported shader core receives tasks or otherwise. Ex. 2006, ¶90. Context swapping is a technique used when switching the processing of one thread to another thread. Context swapping preserves the state of the thread being paused so that when the thread resumes execution, it can pick up where it left off. *Id.* Therefore, context swapping, even if it were disclosed, would not be a disclosure of task priority. *Id.*

Petitioner’s expert admitted that there are reasons and ways of performing context swapping that did not involve task priority, such as swapping when the current thread needs data that is presently unavailable. Ex. 2007, 60:10-62:16. Another example of when context swapping is used is when a thread process is interrupted, the status of the thread process is using context swapping. Ex. 2006, ¶91. When the interrupt is resolved, the thread process can resume its execution at the very same point at which the interrupt occurred due to context swapping. *Id.*

2. Ground 2 Fails

Ground 2 relies on Jiao’s disclosure of “a plurality of constants with a priority level.” Pet., 52. But these priorities are explicitly for *constants*, not for *tasks* as required by claim 20. Unlike claim 20, Jiao uses the priority level of the constants or “constant blocks” to put the constant blocks—not tasks—onto “fast” or “slow” access paths. Ex. 2006, ¶¶127-129. Constant blocks are data. *Id.*

As the ALJ in the ITC has held, priority for *constants* does not disclose *task* priority. Ex. 2003, 163-164. Indeed, Petitioner’s expert conceded that constants are not tasks, they are merely data. Ex. 2007, 74:17-75:21.

Further, Jiao does not disclose that these constant block priorities impact, in any way, the priority of tasks being received by the shader core. Ex. 2006, ¶128. Jiao states only that “[t]he priorities assigned to the constant buffer slots *assist hardware in best utilizing any dedicated constant data access paths/mechanisms*, if present.” *Id.*, 12:15-17 (emphasis added). Thus, the constant block priorities are unrelated to task priority.

3. Ground 3 Fails

Similarly, Ground 3’s Stuttard does not describe priorities for a *task*; it only describes priorities for a *thread*. Pet., 62-63; e.g., Ex. 1006, 23:18-28, 23:35-24:6 (all discussing priority of threads only); Ex. 2006, ¶171. Fatally, Petitioner and its expert concede that a thread is *not* a task. Pet., 41; Ex. 1003, ¶102.

Ground 3 nonetheless assumes that a high-priority thread must be the result of a high-priority task, and therefore switching threads based on thread priority somehow discloses that “the shader core is configured to receive the plurality of tasks based on a priority associated with each of the plurality of tasks,” as required by claim 20. There is no evidence supporting this assumption. Ex. 2006, ¶¶171-173.

To the contrary, Petitioner’s own expert conceded that high priority threads were not necessarily the result of some (undisclosed) high priority task. Ex. 2007, 152:3-17. Rather, according to Dr. Pedram, the threads that make up a single task could themselves have different priorities, some high and some low. *Id.* Thus, a POSITA would understand that thread priority may change depending on certain conditions that have nothing to do with task priority. Ex. 2006, ¶¶172-173. For example, threads may have dependencies that impact their ability to be processed, which may impact their priority level. When a thread pauses, Stuttard describes changing which thread is executing depending on if that thread is “yielding” or “not yielding.” Ex. 1006, 24:22-26. This further indicates that the priority of a parent task does not relate to child thread priority. Ex. 2006, ¶173.

Accordingly, Stuttard does not disclose any relationship between the priority of threads and the priority of tasks, and the thread priority described in Stuttard is not used to determine the order in which tasks, or even threads, are received.

C. All Challenged Claims

The failings of Petitioner established in the foregoing sections apply specifically to claims 19 and 20, respectively. The following sections concern limitations that are common to all the challenged claims and, therefore, establish reasons that the Petition has failed to prove unpatentable all the challenged claims 15-20.

1. Ground 1 Fails For All Challenged Claims

Ground 1 fails for all challenged claims because Wilt does not teach the following limitations required by each challenged claim:

- a shader core that receives tasks, and executes a first task while executing a second task;
- plurality of engines separate from the shader core; and
- engines that receive tasks.

a. Wilt Does Not Disclose Simultaneously Executing Different Tasks

All claims require “a plurality of engines configured to receive . . . a plurality of tasks,” and “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.” Ex. 1001, Claims 15-20. Petitioner has at best alleged simultaneous execution of different threads; but without more, nothing

indicates that any two or more threads would have been decomposed from different parent tasks, and not the same parent task, and therefore nothing indicates that the systems of Wilt (or any other reference) are configured to simultaneously execute different tasks. Ex. 2006, ¶¶69-76.

Petitioner argues that work distribution unit 210, front-end unit 212, and core interface unit 303, individually or collectively, receive and process tasks from software programs running on the host CPU. Pet., 26-27. However, under both Single PPU and Multi-PPU theories, Further, Petitioner has not shown that the shader core is configured to execute a first task while executing a second task. *Id.*, ¶74.

Although Wilt states that “tasks” are being received and executed by cores 208 of the PPUs, they are not the tasks claimed by the ’381 Patent. *See generally* Ex. 1004, 4:1-23; Ex. 2006, ¶70. A POSITA would understand Wilt’s cores 208 do not actually receive and execute the claimed tasks because the highest order item that any component of Wilt’s system receives are commands or command streams, *i.e.*, threads (the ’381 Patent defines a thread as a sequence of instructions, *i.e.*, a sequence of commands). *Id.*; Ex. 1001, 1:49-51. For example, Wilt discloses that CPU 102, the “master processor of system 100, . . . issues **commands** that control the operation of PPUs 202.” Ex. 1004, 4:61-5:4; *see also id.* (“CPU 102 writes **a stream of commands** for each PPU 202”), 10:21-25 (“[a] driver program executing

on CPU 102 of FIG. 1 writes **commands** defining the CTA to a pushbuffer (not explicitly shown) in memory (e.g., system memory 104), from which the **commands** are read by a PPU 202.”) (emphases added). Further, these commands issued by CPU do not ever become the claimed tasks as they make their way through Wilt’s system. Ex. 2006, ¶71. Whatever work is initiated by the CPU, that work is decomposed into smaller and smaller components until it is executed by the alleged shader core, which executes threads. *Id.*

Wilt also explicitly provides that three alleged engines (and the collective engines under the Multi-PPU theory) receive only commands, and in turn, so does the alleged shader core. Ex. 1004, 3:60-64 (“For example, commands related to processing tasks may be directed to a front end unit 212 while commands related to memory operations (e.g., reading from or writing to PP memory 204) may be directed to a memory interface 214.”), 4:7-15 (work distribution unit “receives commands”), 8:15-19 (“[C]ore interface 303 receives . . . commands defining how the data is to be processed.”), 10:32-34; Ex. 2007, 67:15-23 (Dr. Pedram agreeing that “commands” are received by the PPU, front end unit, work distribution unit and core interface); Ex. 2006, ¶72.

A POSITA would understand Wilt’s commands are, at best, a subcomponent of a task as claimed by the ’381 Patent. *Id.*, ¶73. Wilt’s commands “defin[e] how the data is to be processed (e.g., what **program** is to be executed).” Ex. 1004, 8:17-

19 (emphasis added). Meanwhile, the '381 Patent provides that “[t]o perform a single graphics-processing task, a GPU may execute a plurality of different threads (sequence of instructions),” wherein “[e]ach thread may comprise *a shader program*.” Ex. 1001, 1:49-52 (emphasis added). Therefore, in the '381 Patent terminology, Wilt's command streams are threads. Ex. 2006, ¶73. As a result, as also explained above in Section V.A, a claimed task decomposes into several threads, *e.g.*, several shader programs, that perform a specific function. And so Wilt's commands are at best equivalent to a single shader program, or thread. *Id.* And receipt and execution of different threads is insufficient to disclose receipt and execution of different tasks, as the threads could have come from the same parent task. *Id.*

Petitioner has also failed to explain how its Single PPU or Multi-PPU theory executes a first task while executing a second task. *Id.*, ¶74. With respect to Petitioner's Single PPU theory, Wilt explains that “PPU 202 includes a core 208 (or multiple cores 208) configured to execute a large number of *threads* in parallel, where the term ‘thread’ refers to an instance of a particular program executing on a particular set of input data.” Ex. 1004, 5:61-65, 6:16-27; Ex. 2006, ¶75. A thread is not a task, and therefore does not satisfy the claims requirement of executing at least two tasks. *See* Section V.A, *supra*. However, nothing in Wilt discloses that the threads executed by a single PPU correspond to multiple tasks. Ex. 2006, ¶75.

Nor can it be assumed that any two (or more) threads will correspond to two different tasks. *Id.*

Petitioner and its expert explicitly admit that a thread is not a task. Pet., 41; Ex. 1003, ¶102. Dr. Pedram, could not, under oath, state that a thread can be a task, and he further explained that a POSITA would not know how to trace a thread back to its parent task—even if a task could be decomposed into just one thread only—without substantial technical details, none of which are present in the asserted references. Ex. 2007, 14:17-15:6, 23:12-27:4 (stating, *e.g.*, that he would not “be able to map the thread to the task it was decomposed . . . from” because “[s]ometimes it’s not doable. And there’s a lot of information needed about the microarchitecture, about the hardware, the machine, the compiling that’s being used, the programming involvement, different things. So generally, it’s a difficult task. It may be possible in some cases. Sometimes not. I don’t know.”). Dr. Pedram further testified that “even in the case that a task would result in a single thread, the task and the thread are not the same construct,” because threads are what are “executed in the end on [a] piece of hardware,” whereas tasks have to be “[t]ransformed in such a way that it can . . . get mapped into hardware that implements it. *Id.* at., 21:21-22:24.

Absent additional disclosure explaining which tasks certain threads correspond to, it cannot be assumed that any two (or more) threads will correspond to two different tasks. Ex. 2006, ¶75. Dr. Pedram agreed, confirmed that merely

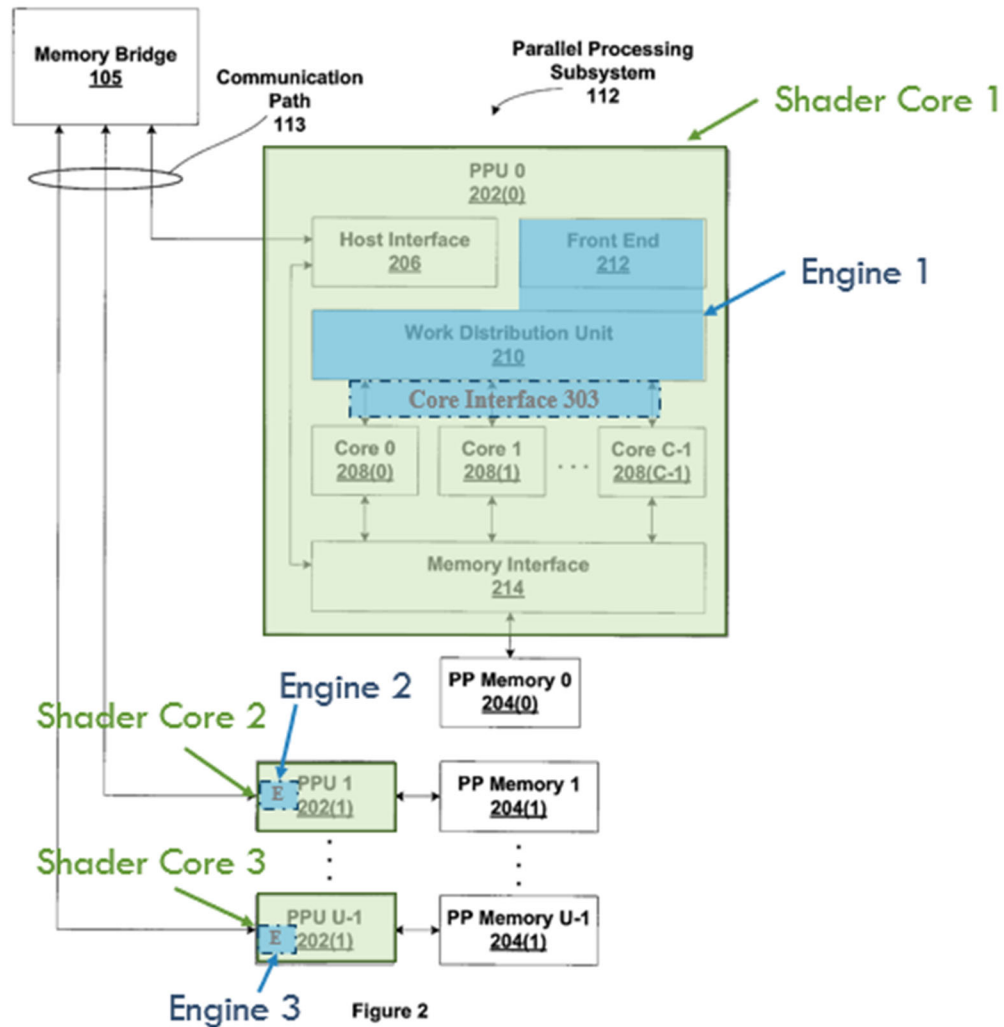
processing multiple threads simultaneously says nothing about whether the system can execute multiple tasks simultaneously. He testified “if a system can process multiple threads at the same time, it does not necessarily care about whether those threads belong to the same higher-level task or multiple different higher-level tasks.” Ex. 2005 (Pedram ITC Dep.), 305:18-306:9. Therefore, processing multiple threads simultaneously is not sufficient to disclose the requirement to “execute a first task from the plurality of tasks while executing a second task.”

Moreover, the Petition has not shown that the system is “configured to” execute threads that were decomposed from two separate parent tasks. Ex. 2006, ¶76. Any sort of “happenstance” or “probability” based argument, which is not supported in the first instance, would be insufficient to meet the requirement of “configured to,” as Petitioner itself argued in the ITC. *See* Section V.C.

b. The Multi-PPU Theory Does Not Disclose or Render Obvious the Claims

Regarding the Multi-PPU Theory, no POSITA would understand that front end, work distribution unit, and core interface of a single PPU is a collectively single “engine,” as Petitioner argues. Indeed, Dr. Pedram testified that he would not consider these three items as “a single item.” Ex. 2007, 72:2-73:9. Nor would any POSITA. Ex. 2006, ¶77. Nor could he provide a reason that any POSITA would group them together as one “engine.” Ex. 2007, 72:2-73:9. Instead, he repeatedly

referred to it as a “box drawing exercise” undertaken solely to invalidate the claims, with no technical basis in reality. *Id.*, 69:18-70:3, 71:13-72:1. Therefore, this theory, which Petitioner’s Multi-PPU Theory hinges on, fails to disclose or render obvious claim 15, and subsequently all the dependent claims.



Ex. 1004, Fig. 2 (annotated with Multi-PPU Theory).

Similarly, the Multi-PPU Theory as modified by the Institution Decision fails because a POSITA would not understand the set of cores 208 within and across all

PPUs to constitute a “shader core.” Ex. 2006, ¶78. Cores across different PPUs are not related to each other in any way, and such a box-drawing crosses PPU boundaries, which has no logical basis. Ex. 2007, 41:15-44:5 (“there is no point-to-point connection between these [PPUs]”). Dr. Pedram could not opine that the PPUs communicate with each other. *Id.*, 41:25-42:20, 44:3-7. Additionally, each core only receives commands from the alleged engines within its own PPU, as Dr. Pedram admitted. Ex. 2007, 87:24-88:7. This demonstrates that it makes no sense to consider all cores across all PPUs as one entity, let alone one shader core, as such shader core resource sharing is one of the defining features of the shader core and its relationship with the engines. Ex. 1001, 7:64-8:43; Ex. 1005, Figs. 2-3.

c. Wilt’s PPU(s) Do Not Receive Tasks From the Alleged Engines Because the Alleged Engines Are Inside the PPU(s)

All claims require “a plurality of engines” to be separate from the “shader core.” Ex. 1001, Claims 15-20.

Under both the Single PPU Theory and the Multi-PPU Theory, described above in Section VI.A.1, in which either a single PPU or the combination of all PPUs are the alleged shader core, the alleged shader core encompasses the alleged engines, contrary to the claims. Pet., 26. That is explicitly Petitioner’s *argument*: “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.*

Claim 15, by contrast, requires the shader core to receive tasks *from* engines. Ex. 2006, ¶80. A plain and ordinary reading means the shader core cannot logically receive something from within itself, so the engines must be outside the shader core. *Id.* Allowing the shader core to encompass the engines would also render redundant the separate “receive” steps of the engine and the shader core. *Id.* The specification confirms that the engines and the shader core are separate:

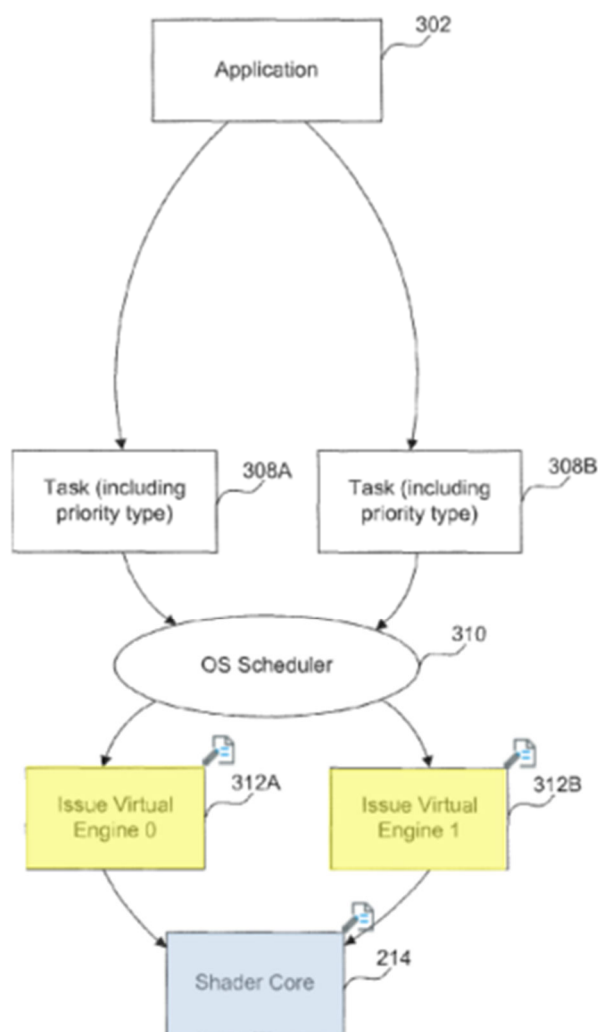


FIG. 3A

Ex. 1001, Fig. 3A.

Prosecution disclaimer applies as well. *Speedtrack, Inc. v. Amazon*, 998 F.3d 1373, 1379 (Fed. Cir. 2021) (“Prosecution disclaimer can arise from both claim amendments and arguments.”); Ex. 2006, ¶81. The applicant argued that because internal components of the alleged shader core in a prior art reference were identified

by the examiner as the claimed “engines,” the read failed because the alleged shader core cannot “receive the plurality of tasks from at least one of the plurality of engines” as required by the claims, because the “internal components of shader core 130 and shader core 130 are one in the same.” Ex. 1002 (’381 FH), 164-165. As a result, the examiner concluded the prior art did not disclose “the shader core configured to receive the plurality of tasks from at least one of the plurality of engines.” *Id.*, 158.

**d. Rewriting Petitioner’s Theory to Contradict
Petitioner’s Explicit Theory Is Improper**

Petitioner explicitly asserted that the first processing unit was subsystem 112, and that each PPU, or the plurality of PPUs, within subsystem 112 was the claimed shader core. Pet., 25, 28; Ex. 1003, ¶¶69, 76-77. The Institution Decision, however, altered Petitioner’s theories, effectively rewriting one PPU as the “first processing unit,” with the cores within that PPU as the claimed shader core for the Single PPU theory, and the collection of PPUs in combination as the “first processing unit,” with the cores collectively throughout all of the PPUs as the claimed “shader core” for the Multi-PPU theory. Institution Decision (“ID”), 19. The Institution Decision justified this rewriting because “it is unclear how subsystem 112 is used” in Petitioner’s original theories. *Id.*

Any fault in Petitioner’s theories should not be corrected by the Institution Decision, and ambiguity of disclosure should not be a basis to invalidate issued patent claims. “The Board must base its decision on arguments that were advanced by a party” *Oren Techs., LLC v. Proppant Express Invs. LLC*, No. 2019-1778, 2021 U.S. App. LEXIS 21859, at *13 (Fed. Cir. July 23, 2021); *see also id.* at *14 (“We have held that the Board erred when it relied on a prior art reference that was unasserted for meeting a particular limitation, even though the passage from the reference that the Board relied on for disclosing the limitation had been block quoted in the petition and institution decision, in a different context.”); *Koninklijke Philips N.V. v. Google LLC*, 948 F.3d 1330, 1335 (Fed. Cir. 2020) (“Congress chose to structure a process in which it’s the petitioner, not the Director, who gets to define the contours of the proceeding.”); *In re Magnum Oil Tools Int’l, Ltd.*, 829 F.3d 1364, 1381 (Fed. Cir. 2016).

2. Ground 2 Fails For All Challenged Claims

Ground 2 fails for all challenged claims because Jiao does not teach the following limitations required by each challenged claim:

- a shader core that receives tasks, and executes a first task while executing a second task;
- plurality of engines separate from the shader core;
- engines that receive tasks; and

- a scheduling module that is *not* associated with the first processing unit.

a. Jiao’s Shaders Do Not Disclose or Render Obvious the “Plurality of Engines”

All claims require “a plurality of engines configured to receive . . . a plurality of tasks,” which is separate from the shader core. Ex. 1001.

i. Jiao’s Shaders Do Not Receive Tasks, or Even Threads or Instructions

Petitioner’s theory is that Jiao’s alleged engines, vertex shader 304, geometry shader 306, and pixel shader 310, receive graphics *data* (e.g., graphics primitives, which are shapes) to produce pixel data. Pet., 39; Ex. 2007, 75:22-77:11, 79:25-80:16, 95:14-96:44. But as Dr. Pedram conceded, graphics data, including graphics primitives, are not tasks. Ex. 2006, ¶96.

Petitioner claims that Jiao’s shaders receive tasks from driver software 128. Although Jiao discloses that driver software “communicates instruction sets or commands,” it never states that those instruction sets or commands are provided to the shaders. Ex. 1005, 5:17-23. All Jiao discloses is that those instruction sets or commands are provided generally to “registers in GPU 114.” *Id.*

Instead, the shaders only receive data. Ex. 2006, ¶98. Jiao describes “primitives” and constants, called “vertex information” being passed to vertex shader 304. *Id.* For example, the Petition and the Institution Decision rely largely on a portion of Jiao that states: “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.” Ex. 1005, 7:57-60. But this merely says that when driver software receives a command, driver software causes *vertex information* (not tasks) to be sent to vertex shader. Ex. 2006, ¶98. It does not state that the vertex shader receives commands. *Id.* Dr. Pedram admitted that the “vertex information” are primitives, not tasks. Ex. 2007, 90:17-22 (agreeing that “primitives are the vertex information that Jiao describes as being passed to the vertex shader in order to allow the vertex shader to perform operations such as transformations, skinning, and lighting.”). Thereafter, “vertex shader 304 passes *data* to the geometry shader 306,” which “receives, as inputs, vertices for a full primitive. Ex. 2006, ¶99. The geometry shader 306 receives, as inputs, vertices for a full primitive,” and “also receives constants.” Ex. 1005, 6:28-32 (emphasis added); Ex. 2007, 95:22-96:24 (geometry shader receives “data” and “the right bunch of constants that it needs”). “[G]eometry shader 306 then outputs information to a rasterizer 308 The pixel shader 310 is invoked for each pixel covered by the primitive,” and “also receives constants.” Ex. 1005, 6:34-44, 8:7-10 (“The Z-values are passed back to the pixel shader 310.”). Dr. Pedram could not identify any disclosure where Jiao’s shaders *receive* tasks as opposed to data in the form of primitives or state data. Ex. 2007, 90:5-15, 90:23-113:11; *see also id.*, 79:9-80:7,

80:11-16. Thus, in each instance, Jiao's shaders are receiving *data*, not tasks. Ex. 2006, ¶99.

Dr. Pedram's theory seems to be that the shaders must be configured somehow, so they must receive tasks. *Id.*, ¶100. But that is insufficient for several reasons, including the fundamental one that Jiao does not describe how the shaders receive their configuration, let alone that they receive them from a scheduling module. Nor could Dr. Pedram cite where Jiao actually states that the shaders receive tasks, in contrast with the explicit discussion in Jiao that the shaders receive primitives and constants; he could only state that in addition to primitives and constants, "there *should be* some other information because the vertex shader could perform in a reconfigure manner a number of different operations" but when asked to identify where in Jiao states that shaders receive tasks, he could only point to "vertices," "primitives," "constants," and the like. Ex. 2007, 90:16-98:2. As Dr. Pedram could not disagree with, there are numerous ways to configure a shader program, and not all of them require transmission of tasks to the shaders via a scheduling module. *See* Ex. 2007, 100:4-107:20, 111:24-113:11; Ex. 2006, ¶100.

Any alleged tasks that are processed by the alleged shader core are *generated* in the first instance by the shaders. Pet., 47 (citing Ex. 1005, 9:35-38); Ex. 2006, ¶101. They are not received by the shaders, further confirming what the rest of the disclosure also states. *Id.* For example, Column 6 of Jiao describes the shaders

receiving data, and then follows that discussion with the statement “[h]aving described components that generate tasks for execution units in the computational core 204...” Ex. 1005, 6:49-52. The previous paragraphs were about the operation of the shaders, not any other element that generate tasks and send them to the shaders. *See also 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.”).

ii. Jiao’s Thread Slots are Not Engines

Petitioner’s alternative theory regarding thread slots as being included in the claimed plurality of engines also fails. Pet., 40; Ex. 2006, ¶102. Jiao’s only disclosure regarding thread slots provides, in accord with their name, that “EU pool control unit 206 assigns *a thread* to one of the *empty* thread slots.” Ex. 1005, 9:18-21 (emphasis added). Thus, each thread slot, at best, ever receives only one thread. Ex. 2006, ¶102. Otherwise, EUP could assign threads to any thread slot, not merely the empty ones. *Id.* Further, the storage of one thread does not disclose the storage of tasks, nor the storage of multiple tasks because Petitioner has not, nor can, sufficiently tie these threads to at least two tasks. *Id.*

Additionally, the thread slots are within the execution units of EU pool, the alleged shader core. Ex. 2006, ¶103. This means that the alleged shader core cannot receive anything from the thread slots because the thread slots are internal

components to the alleged shader core. *Id.* As explained above in Section VI.C.1.c, such structure conflicts with the claim language.

iii. Jiao’s Shaders are Part of the Shader Core

Petitioner alleged EU Pool 414 is the claimed shader core. Pet., 47-49. But Jiao’s shaders are within EUP 414, in contradiction to the claim language requiring they be separate. Jiao provides the shaders are within EUP 414: (“In general, an embodiment of the graphics processor 202 utilizes programmable vertex 304, geometry 306, and pixel shaders 310. ***Rather than*** implementing the functions or operations of these components as ***separate*** fixed function units with different designs and instruction sets, ***the operations are instead executed by the pool 414*** of execution units 420a, 420b . . . 420m with a unified instruction set.”). Ex. 1005, 9:25-31. This disclaimed structure means that the alleged shader core cannot receive anything from the alleged engines. See Section VI.C.1.c, *supra*.

iv. Jiao’s Shaders Are Shader Programs, Not Engines

Jiao’s shaders are ***shader programs*** not the claimed engines nor even components of the claimed first processing unit. Ex. 2006, ¶105. Jiao states: “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.” Ex. 1005, 6:59-62. A POSITA would understand that shader programs are components of tasks, which are distinct from the claimed engines. See Ex. 1001,

1:51-66, 8:33-38 (a single graphics processing task typically comprises various shader programs); Ex. 2006, ¶105. Jiao further distinguishes its shaders as being programmable, and describes them as having “operations” that are not “separate fixed function units with different designs and instruction sets.” Ex. 1005, 9:26-31. This is evidence that the shaders are components of tasks, because tasks are composed of threads, which are composed of instructions. Ex. 2006, ¶105. A POSITA would understand operations to be threads or instructions. *Id.*

Further, Jiao’s shader programs are launched/issued like the ’381 Patent’s shader programs because Jiao discloses that the operations of the shader programs “are *executed* by the pool 414 of execution units.” Ex. 1005, 9:26-31, 2:16-20 (describing shader programs as “running” on a computational core); Ex. 2006, ¶106. The ’381 Patent also describes shader programs as being executed by the shader core. Ex. 1001, 1:58-59 (“[t]o efficiently execute the various threads (e.g., shader programs) of a single graphics-processing task”), 8:64 (“when data is needed by any of these components to execute a shader program”).

Further, the shader programs are not engines because their inputs and outputs are connected to each other in a serial fashion. Ex. 2006, ¶107. Dr. Pedram agrees that the workflow is sequential, transitioning the output of one shader to the input of the next. Ex. 2007, 109:6-10. Jiao describes them as being part of a “graphics pipeline,” where data is first passed to the vertex shader, then the geometry shader,

and lastly the pixel shader. Ex. 1005, 6:10-44, Fig. 3. Therefore, they do not independently receive tasks from a scheduling module. *See* Ex. 2007, 109:6-10.

b. Jiao’s Alleged Scheduling Module Is Associated with the First Processing Unit

All claims require a “scheduling module” which is associated with the second processing unit, and not the first processing unit. Ex. 1001; ID, 9.

Petitioner’s alleged engines (shaders) and the alleged scheduling module (EU Pool control unit) are part of the same GPU. Ex. 1005, Fig. 2. This is contrary to the ID’s construction of the term “associated with,” which requires that the scheduling module *not be in a first processor*. ID, 9. Additionally, Petitioner’s alleged first processing unit (computational core) and second processing unit (command stream processor) are also within GPU. Ex. 1005, Figs. 2-4; Ex. 2006, ¶108. However, EUP control unit is not “associated with” the second processing unit because it is a separate component from command stream processor, and Jiao discloses only that command stream processor can send data to EUP control unit. Ex. 1005, 7:19-20 (“The command stream processor 406 provides triangle vertex indices to the EU pool control unit 206.”), Fig. 4; Ex. 2006, ¶108. If this satisfied the claim requirement of being “associated with,” then EUP control unit would also be associated with the first processing unit given that EUP control unit also sends data to the shaders, which are components of the first processing unit. *Id.*, Ex. 1005,

Fig. 4 (arrow notating sending “shader inputs”). However, the applicant distinguished prior art by arguing that the claimed scheduling module cannot be associated with the same processing unit that the claimed engines are associated with – namely, the claimed first processing unit. Ex. 1002, 182-183. The applicant explained that prior art did not disclose the claimed invention because “[t]hread scheduler 210 and the hardware units are not associated with different processing units; rather, they are associated with the same processing unit—namely shader core 130.” *Id.*

c. Jiao’s Alleged Shader Core Does Not Receive “The Plurality of Tasks”

All claims require “a shader core configured to receive the plurality of tasks from at least one of the plurality of engines.” Ex. 1001.

But here, the alleged tasks executed by the shader core are the ones generated in the first instance by the shaders. Pet., 47 (citing Ex. 1005, 9:35-38); Ex. 2006, ¶109. The Institution Decision held that the tasks received by the shader core were not required to be the same tasks received by the engines, even though the claim language uses the word “the” plurality of tasks. ID, 26-27, 30-31. The rationale was that in the ’381 Patent, there is “input logic” between the engines and the shader core that “prepares” the task for issuance to the shader core, and thus processes it in some way, and therefore allowing for some modification of the tasks. *Id.* But as depicted

in Figure 2 of the '381 Patent, not all tasks undergo this allegedly transformation by the input logic, *e.g.*, the tasks from the background engines and the real-time low-latency engine go straight to the arbitration logic. Ex. 1001, Fig. 2; Ex. 2006, ¶111.

Neither does the input logic “generate” tasks, as Jiao’s shaders do, nor do they “process” them as the Institution Decision found. ID, 30; Ex. 2006, ¶112. Instead, “[i]nput logic arbitrates which tasks are issued to shader core 214.” Ex. 1001, 7:20-21. For example, components of input logic identifies shader programs and data relevant to the tasks: “[v]ertex analyzer 208 identifies shader programs associated with a graphics-processing and/or general-compute task and schedules when each shader program can be launched in shader core 214 based on input and output data that will be available” and “[s]can converter 212 traverses the primitives to determine pixels to be processed by shader core 214.” Ex. 1001, 7:42-46, 7:57-58. Meanwhile, arbitration logic merely provides tasks to the shader core. *Id.*, 7:32-33, 7:60-62. None of these components generate tasks, nor do they modify or process, tasks. Ex. 2006, ¶112.

Nor has it been shown that the tasks generated by Jiao’s shaders are merely modified or even processed versions of any alleged tasks received by the shaders (which do not exist in Jiao in the first case, as explained above). Ex. 2006, ¶113. Even if Petitioner were correct that: (i) the shaders received configuration instructions from a scheduling module, (ii) once configured, the shaders receive

primitives and, (iii) on the basis of those primitives, the shaders generate tasks, those tasks are not and cannot be merely variations of the configuration instructions, certainly are not in any way analogous to the examples of tasks of the '381 Patent. *Id.*

Even if pointing to the generated shader tasks were sufficient, the Petition has not shown that the system is “configured to” execute shader tasks that were decomposed from two separate parent tasks. Ex. 2006, ¶114. Any sort of “happenstance” or “probability” based argument, which is not supported in the first instance, would be insufficient to meet the requirement of “configured to,” as Petitioner argued in the ITC. *See* Section V.C, *supra*.

d. Jiao Does Not Disclose Receiving and Simultaneously Executing Tasks

All claims require “a shader core configured to . . . execute a first task from the plurality of tasks while executing a second task.” Ex. 1001.

To the extent that Jiao’s shader programs generate anything besides data, they generate threads at best, not tasks. Ex. 2006, ¶115. Notably, as Petitioner admits, Jiao refers to the tasks generated by the shaders as both tasks and threads. Pet., 43 (citing Ex. 1005, 8:65-67, 9:10-13). While Petitioner argues the “tasks or threads” of Jiao are like tasks in the '381 Patent, that is incorrect; they are what the '381 Patent would call threads. Ex. 2006, ¶115. For example, Jiao discloses that the

“tasks” generated by the shaders are assigned to “available threads” within the various execution units 420, including in “thread slots.” Ex. 1005, 9:39-54. “The EU pool control unit 206 particularly knows which threads have been assigned tasks and are occupied.” *Id.* As Petitioner admits, a thread is not a task. Pet., 41. Nor has Petitioner sufficiently tied these threads to multiple tasks. Ex. 2006, ¶115. Thus, to the extent there is any ambiguity, Jiao’s “tasks or threads” should be interpreted as threads, and Petitioner fails to meet its burden to prove otherwise. *Id.*

Additionally, the word “task” as used in Jiao does not refer to what the ’381 Patent refers to as “tasks,” for the additional reason that in Jiao, multiple so-called “tasks” can be associated with a single context, indicating they are actually what the ’381 Patent would call threads. Ex. 2006, ¶116. The ’381 Patent describes each thread is associated with *a* set of locally stored state data (such as texture handles, shader constants, etc.), which is called a context. Ex. 1001, 1:53-57. Meanwhile, Jiao describes *multiple* of its “tasks” as having a single context. Ex. 1005, 13:52-53 (“the execution unit 420 is ready to receive tasks that use the new constant context (712)”), 14:29-31 (“the counter 904 in the constant reference table 910 keeps track of the number of outstanding tasks for a given shader constant context”); 14:34-36 (“When the counter 904 reaches zero, the EU pool control logic 605 is notified that all tasks for the context have completed and the EU pool 414 is flushed.”). This further indicates that what Jiao calls “tasks” are actually threads. Ex. 2006, ¶116.

3. Ground 3 Fails For All Challenged Claims

Ground 3 fails for all challenged claims because Stuttard does not teach the following limitations required by each challenged claim:

- a shader core that receives tasks, and executes a first task while executing a second task;
- plurality of engines separate from the shader core;
- engines that receive tasks; and
- a scheduling module that is *not* associated with the first processing unit.

a. Stuttard Does Not Disclose Receiving and Simultaneously Executing Tasks

All claims require “a plurality of engines configured to receive . . . a plurality of tasks,” and “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.” Ex. 1001.

Petitioner argues that “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 and the processing blocks 106 are the shader core. Pet., 56. At best, the alleged engines and shader core in Stuttard’s system receive only threads, not tasks. Petitioner’s theories fail for two reasons: first, Stuttard

cannot operate on multiple tasks simultaneously, as it cannot operate even on multiple *threads* simultaneously; second, even if that is incorrect, Petitioner fails to show (nor can it show) that the threads were decomposed from more than one task. Thus, Stuttard fails to disclose that the alleged shader core executes a first task and a second task simultaneously.

i. Stuttard Receives and Executes Threads, Not Tasks

Stuttard's graphics processing elements receive and process threads only, not tasks. Ex. 2006, ¶¶134-135. Stuttard makes no disclosure of tasks at all, except to state that generally that threads "work[] co-operatively with other threads to perform a task or tasks," which only serves to emphasize the distinction between task and thread, as argued above. Ex. 1006, 10:10-12. Stuttard describes what multithreading and multitasking are, yet never mentions that it is capable of either. *Id.*, 10:12-15. All of Stuttard's disclosures regarding how its components work relate to threads, which Petitioner has not sufficiently tied to multiple tasks. Ex. 2006, ¶135.

The instruction fetch unit, EPU, and thread scheduler also do not send the claimed tasks to the alleged engines. Ex. 2006, ¶136. Stuttard explains that "[i]nstructions for *a thread* are fetched from local memory or the EPU 8 by the fetch unit 1023." Ex. 1006, 11:12-14 (emphasis added). This indicates that the instruction

fetch unit fetches, as implied by its nomenclature, are instructions. Ex. 2006, ¶136. Without more, this cannot disclose the sending of entire tasks to the alleged engines.

Stuttard does not disclose that the thread scheduler or EPU sends tasks. Ex. 2006, ¶¶137-139. Thread scheduler calculates which thread should be active. *Id.*, 23:35-24:12. EPU also calculates which thread should be active, and sends control signals to thread manager. *Id.*, 8:17-19, 25:19-24. This *selection* does not disclose engines receiving anything, not to mention tasks, from a scheduling module. Ex. 2006, ¶138. But even if it did, what is being received by the alleged engines are not tasks, but threads. *Id.* Stuttard explains very clearly how thread scheduler and/or EPU operates if a scheduling trigger occurs: “First, it stops the active **thread** from running, and waits a cycle for any semaphore decrements to propagate.” *Id.*, 25:1-3. Then, thread scheduler activates **a thread** based on parameters should as the thread’s priority, whether the thread is yielding, etc. *Id.*, 25:5-17. Thus, both thread scheduler and EPU, at best, are capable of only activating one thread for execution. Ex. 2006, ¶138.

Control signals sent by EPU are even further removed from the claimed tasks because control signals are **signals** that set up communication channels and control the flow of data. Ex. 2006, ¶139. “The control signals inform the thread manager as to when instructions are to be fetched and where the instructions are to be found.” *Id.*, 8:19-21.

Further, Stuttard discloses that the alleged engines only receive threads, not tasks. Ex. 2006, ¶140. For example, the binning unit sorts primitive data, which are not tasks. Ex. 1006, 21:15-21. The channel controller and the array controller receive only instructions, not tasks. *Id.*, 26:3-7. Similarly, the cache memory unit “stor[es] instructions fetched for each thread.” *Id.*, 23:5-6. Each thread processor only receives a thread, not tasks. *Id.*, 25:26-29.

Stuttard provides that the processing elements within the processing blocks are “controlled to carry out a series of instructions in an instruction stream,” which “are known as ‘threads’.” Ex. 2006, ¶141; Ex. 1006, 10:7-10. Further, Stuttard explains that thread scheduler determines which *thread* should be active and being processed. *Id.*, 23:34-25:24.

The result of all of this is that the alleged shader core (processing blocks) do not receive the claimed tasks, but at most threads. And so as explained below, although Stuttard does not disclose simultaneously executing different threads, even if it did, it would not disclose simultaneously executing different tasks, because it cannot be assumed that different threads correspond to different parent tasks.

ii. Stuttard Does Not Simultaneously Execute Different Threads

Petitioner mainly relies on one sentence in Stuttard to demonstrate that Stuttard can simultaneously execute different threads, and therefore, in its view, also

simultaneously execute different tasks: “At any particular moment in time, each thread may be assigned to any one of a number of tasks.” Ex. 1006, 11:19-21. But as Dr. Pedram conceded, the threads being discussed here are the threads as they exist in the cache of thread manager—not what is being executed by the processing blocks. *See id., id.* Fig. 4; Ex. 2007, 125:2-7 (agreeing that “in the citation on page 11 [of Stuttard] that you point to that is describing in the thread manager, the eight threads that are discussed are items B0 through B7 in the TM cache 1024”). Stuttard contains no disclosure that these threads will each be executed simultaneously by the processing blocks, the alleged shader core. Ex. 2006, ¶142. Petitioner’s theory relies on the unstated proposition that the multiple threads in the cache correspond to threads in the thread processor which then correspond to the multiple processing blocks. *Id.*, ¶143. But not only is that never stated in Stuttard, Dr. Pedram does not even agree with it. He testified that the eight thread processors in Figure 4 do *not* necessarily correspond to the eight threads in the cache. Ex. 2007, 243:24-244:3.

Indeed the *only* discussion in Stuttard describing how threads in the portion Petitioner so heavily quotes would actually be executed *in the processing blocks* is that “the same instruction stream is supplied to all processing blocks of the system.” Ex. 2006, ¶144. There simply is no discussion stating that the processing blocks could each correspond to different threads in the cache 1024. Indeed, in the entirety of Figure 4, the same instruction stream is supplied to all processing blocks, as made

clear in the paragraph immediately after the quote Petitioner points to—“this particular example” clearly refers to the only “example” referenced, Figure 4:

In the example shown in Figure 4, the thread manager includes one thread processor 1026 for each thread. 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. In this particular example the same instruction stream is supplied to all of the processing blocks in the system.

Ex. 1006, 11:27-35.

Similarly, Stuttard’s Abstract states that dividing the processing elements into a “plurality of processing blocks” is specifically to allow them to be “operable to process respective groups of *data items*”—not threads. *Id.*, Abstract; Ex. 2006, ¶145. Indeed, the “Summary of the invention” discloses “a data processing apparatus comprising *an array of processing elements*, which are operable to process respective data items in accordance with *a common received instruction*, wherein the processing elements are operably divided into a plurality of processing blocks having at least one processing element, the processing blocks being operable to process respective groups of *data items*.” *Id.*, 2:25-33 (emphasis added); Ex. 2006, ¶146.

Stuttard's system is "a data processing apparatus comprising *a SIMD* (single instruction multiple data) array of processing elements, wherein the processing elements are operably divided into a plurality of processing blocks." Ex. 1006, 2:17-23 (emphasis added). A SIMD is only capable of processing a single thread at any given moment in time. Ex. 2006, ¶147. Because a thread cannot correspond to more than one task, to the extent Stuttard is interpreted as disclosing executing complete tasks at all (which Patent Owner disagrees with) is only capable of processing a single task at one time. *Id.*

Stuttard makes clear in all disclosures that it is only capable of processing the same instruction across all processing blocks. *Id.*, ¶148, Ex. 1006, 2:25-34 (all processing blocks operate on a "*common received instruction*"). Stuttard's system includes a thread manager, which controls the entire array of processing blocks. *Id.*, 8:16-24, 10:5-17, Fig. 3. When describing how the thread manager operates, Stuttard provides: "At a given time, *only one thread is executing*, and the scheduling of the time multiplexing between threads is determined by the dynamic conditions of the program execution." *Id.*, 23:18-21. Stuttard further provides that the thread scheduler, of which there is only one, "determine[s] *which thread* should be active at any one time." *Id.*, 12:15-18. This is reiterated in a section describing the function of the thread scheduler in more detail, entitled "Determining *which thread* should be active." *Id.*, 23:34. The section explains that the thread scheduler will recalculate

which thread should be active given certain circumstances. *Id.*, 23:34-25:17. Further, the purpose of the thread processors “are to allow high level flow control to be performed for *a thread*, (such as looping and conditional branches), and to assemble instructions to assemble instructions to be issued to the array controller 104 and channel controller 108.” *Id.*, 26:3-7; Ex. 2006, ¶¶150-153. All of these disclosures clarify that Stuttard has only a single active thread at one point in time. Ex. 2006, ¶148.

As to Petitioner’s apparent supposition that because there are multiple processing blocks, they must each process different threads simultaneously, there are several benefits to grouping the array of processing elements that is unrelated to the ability to process multiple threads or tasks at one time. Ex. 2006, ¶154. One benefit is allowing the processing blocks to operate on different regions of the display, *i.e.*, different data groups. *Id.* Stuttard’s system includes binning units whose purpose “is to sort primitive data by region, since the data is generally not provided by the host system in the correct order for region based processing.” Ex. 1006, 14:32-35. During this process, the binning units within each processing block work collectively to retrieve primitive data from local memory, and sorts the primitives by region. *Id.*, 21:15-21. After binning, “all of the PEs in one processing block contain[ing] data concerning respective primitives occurring in a single region.” *Id.*, 22:4-10.

Dividing the array of processing elements into processing blocks also allows for redundancy. Ex. 2006, ¶155. A large array of processing elements “can result in a large die size for the manufacture of the device in a silicon device.” Ex. 1006, 9:22-28. Accordingly, it is preferable to have redundant processing elements such that if one die area is faulty, another can be used in its place. *Id.*

iii. Even If Stuttard Executes Multiple Threads Simultaneously, It Does Not Execute Different Tasks Simultaneously

Additionally even if Stuttard’s system was capable of executing more than one thread at a time, absent additional disclosure explaining which parent task certain threads correspond to, it cannot be assumed that any two (or more) threads will correspond to two different parent tasks. Ex. 2006, ¶156. Dr. Pedram agreed, testifying “if a system can process multiple threads at the same time, it does not necessarily care about whether those threads belong to the same higher-level task or multiple different higher-level tasks.” Ex. 2005, 305:18-306:9. Therefore, processing multiple threads simultaneously is not sufficient to disclose the requirement to “execute a first task from the plurality of tasks while executing a second task.” Ex. 2006, ¶156.

Pointing to the capability of multi-threading is also insufficient because the Petition has not shown that the system is “configured to” execute threads that were decomposed from two separate parent tasks. *Id.*, ¶157. Any sort of “happenstance”

or “probability” based argument, which is not supported in the first instance, would be insufficient to meet the requirement of “configured to.” *See* Section V.C, *supra*.

b. The Alleged Engines Are Within the Alleged Shader Core

All claims require “a plurality of engines” which is separate from the shader core. Ex. 1001.

Petitioner’s alleged engines conflict with the claim language and run afoul of prosecution history disclaimer, where the applicant distinguished the prior art on account of the prior art’s shader core encompassing the engines. *See* Section VI.C.1.c, *supra*. Petitioner argues that the plurality of processing blocks constitutes the shader core. Pet., 59. However, as depicted in Fig. 3, binning unit 1069, which is an alleged engine, is part of Petitioner’s alleged shader core. Ex. 2006, ¶158. Therefore, Petitioner’s theory regarding “the plurality of engines” fails because the alleged engines are at least partially within the alleged shader core. *Id.*

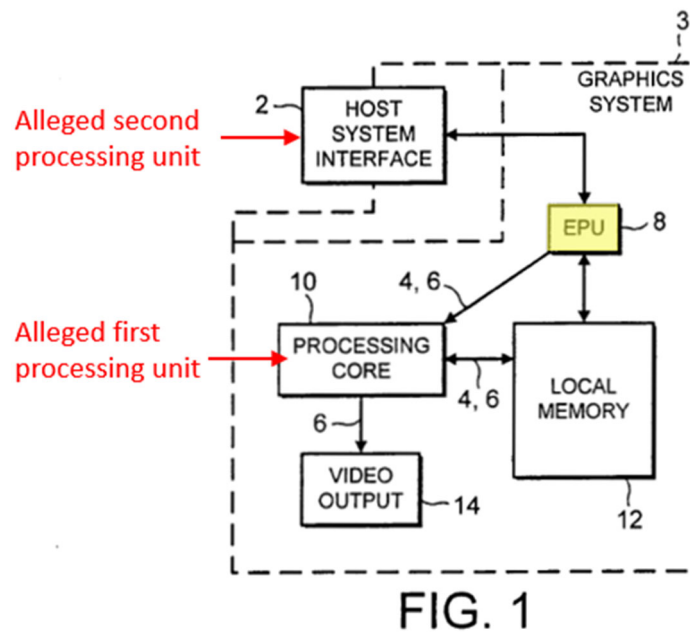
c. The Alleged Scheduling Module Is Not “Associated With” The Second Processing Unit

All claims require a “scheduling module” which is associated with the second processing unit, and not the first processing unit.” Ex. 1001; ID, 9.

Petitioner has two theories for the claimed “scheduling module”: (i) instruction fetch unit and/or thread scheduler, and (ii) the EPU. Pet., 56.

The Institution Decision correctly reasoned that, pursuant to the prosecution history, instruction fetch unit and thread scheduler are in processing core, the first processing unit, and thus are not “associated with a second processing unit.” ID, 34.

The EPU is also not a scheduling module because it is also not associated with a second processing unit. Ex. 2006, ¶¶159-160. Petitioner identifies the host system as the claimed second processing unit that the EPU is associated with. Pet., 56. However, the EPU is at best merely connected to the host system. Ex. 2006, ¶159. That is insufficient to be “associated with it.” *Id.*



Ex. 1006, Fig. 1 (annotated).

In order to be “associated with,” the claimed scheduling unit must be within the claimed second processing unit. Ex. 2006, ¶160. That is clear from the

prosecution history in which the applicant successfully argued past a Jiao reference by arguing that the alleged scheduling module and the alleged engines “are not associated with different processing units; rather they are associated with the same processing unit—namely, shader core 130” specifically because both are “within shader core 130.” Ex. 1002, 183; Ex. 2006, ¶160. Thus, the applicant equated “associated with” with “within.” *Id.*

Separately, if EPU 8 were associated with the host system by virtue of its connection to the host, it is also associated with the alleged first processing unit, the processing core 10, for the same reason. Ex. 2006, ¶161. But the applicant argued past the reference during prosecution by explaining that the claim does not cover a situation in which the alleged scheduling module and engines are “associated with the same processing unit.” *Id.* Under Petitioner’s read, EPU 8 would be associated with the alleged first processing unit, as would be the alleged engines. *Id.* Thus EPU 8 cannot be a scheduling module under the claims.

d. The Alleged Engines Are Not Engines

Petitioner’s alleged engines also cannot be the claimed plurality of engines because they are not parallel, but serialized; Dr. Pedram agrees. Ex. 2007, 165:1-3. Petitioner’s alleged engines perform different functions at different stages in a serialized manner. Ex. 2006, ¶¶162-163. Components of the thread manager

receives threads first, before array controller, channel controller, or binning unit. Ex. 1006, 8:17-27.

Further, each of the engines do not receive tasks or load state data. Ex. 2006, ¶164. For example, binning unit receives only “primitive data,” which are not tasks. *Id.*, 14:32-35, 21:15-21. And status block receives “status information,” which is also not a task. *Id.*, 12:11-14. Nor does every alleged engine load state data – the Petition contains no allegation that binning unit, cache memory unit, or thread processors load state data. *Id.*, 163:21-164:9 (binning unit and components of the thread manager do not load state data); Pet., 56-58.

VII. CONCLUSION

For any of the foregoing reasons, the Board should determine that Petitioner has failed to prove that the challenged claims of the '381 Patent are unpatentable.

Dated: January 17, 2024

Respectfully submitted,

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)

Michael T. Renaud (Reg. No. 44,299)

Adam S. Rizk (Reg. No. 66,867)

MINTZ, LEVIN, COHN, FERRIS,

GLOVSKY AND POPEO, P.C.

One Financial Center

Boston, MA 02111

Telephone: 617-542-6000

Facsimile: 617-542-2241

E-mails: WAMeunier@mintz.com;

MTRenaud@mintz.com

ARizk@mintz.com

Reza Dokhanchy (Reg. No. 62,795)
MINTZ, LEVIN, COHN, FERRIS,
GLOVSKY AND POPEO, P.C.
3580 Carmel Mountain Road, Suite 300
San Diego, CA 92130
Telephone: 858-314-1596
RDokhanchy@mintz.com

CERTIFICATE OF WORD COUNT

Pursuant to 37 C.F.R. §42.24(d), Patent Owner hereby certifies, in reliance on the word count of the word-processing system (Microsoft Office Word 2010) used to prepare this preliminary response, that the number of words in this paper is 13,970. This word count excludes the tables of contents, tables of authorities, certificate of word count, certificate of service, and table of exhibits.

Dated: January 17, 2024

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)

CERTIFICATE OF SERVICE

I certify that a copy of Patent Owner's Response and accompanying ATI Exhibits 2006 through 2007 is being served by electronic mail on the following counsel of record:

Lead Counsel

Jeffrey Johnson
Reg. No. 53,078
ORRICK, HERRINGTON &
SUTCLIFFE, LLP
609 Main Street, 40th Floor
Houston, TX 77002-3106
Telephone: (713) 658-6400
Facsimile: (713) 658-6401
Email: 3J6PTABDocket@orrick.com;
Realtek-AMD_OHS@orrick.com

Backup Counsel

Christopher J. Higgins
Reg. No. 66,422
ORRICK, HERRINGTON &
SUTCLIFFE, LLP
1152 15th Street, N.W.
Washington, DC 20005-1706
Telephone: (202) 339-8400
Facsimile: (202) 339-8500
Email: OCHPTABDocket@orrick.com;
Realtek-AMD_OHS@orrick.com

Steve Baik
Reg. No. 42,281
WHITE HAT LEGAL
P. O. Box 382
San Jose, CA 95002
Telephone: (650) 618-5282
Email: sbaik@whitehat.legal

Dated: January 17, 2024

/s/ William A. Meunier

William A. Meunier (Reg. No. 41,193)