

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

BMW OF NORTH AMERICA, LLC,
Petitioner,

v.

FORAS TECHNOLOGIES LTD.,
Patent Owner.

IPR2024-01346
Patent 7,627,781 B2

Before STEVEN M. AMUNDSON, RUSSELL E. CASS, and LISA A.
MURRAY *Administrative Patent Judges*.

CASS, *Administrative Patent Judge*.

JUDGMENT
Final Written Decision
Determining No Challenged Claims Unpatentable
35 U.S.C. § 318(a)

I. INTRODUCTION

A. Background and Summary

BMW of North America, LLC (“Petitioner”) filed a Petition (Paper 1, “Pet.”) requesting an *inter partes* review of claims 1–30 (“the challenged claims”) of U.S. Patent No. 7,627,781 B2 (Ex. 1001, “the ’781 patent”). We instituted an *inter partes* review under 35 U.S.C. § 314. Paper 7 (“Dec. on Inst.”). Patent Owner filed a Response (Paper 11, “PO Resp.”), Petitioner filed a Reply (Paper 14, “Pet. Reply”), and then Patent Owner filed a Sur-reply (Paper 19, “PO Sur-reply”). An oral hearing was held. Paper 26 (“Tr.”).

We have authority to enter this final written decision (“Decision”) under 35 U.S.C. § 318(a). The standard for review is set forth in 35 U.S.C. § 316(e), which provides that “the petitioner shall have the burden of proving a proposition of unpatentability by a preponderance of the evidence.” For the reasons provided below, we determine that Petitioner has not met this burden of establishing unpatentability of any of the challenged claims.

B. Real Parties in Interest

Petitioner identifies Bayerische Motoren Werke AG as an additional real party in interest. Pet. 72. Patent Owner asserts that it is the sole real party in interest. Paper 4, 2.

C. Related Matters

Patent Owner identifies the following current district court proceedings involving the ’781 patent: *Bayerische Motoren Werke AG, et al. v. Foras Technologies Ltd.*, No. 1:24-cv-00363 (E.D. Va.); and *Foras*

Technologies Limited v. Nissan Motor Company, Ltd., et al., No. 1:23-cv-00640 (W.D. Tex.). Paper 4, 2.

Petitioner identifies the following additional district court proceedings involving the '781 patent: *Foras Technologies Ltd. v. Toyota Motor North America, Inc.*, No. 2:23-cv-00321 (E.D. Tex.); and *Foras Technologies Limited v. Toyota Motor North America, Inc. et al.*, No. 2:23-cv-00150 (E.D. Tex.). Pet. 72.

The parties also identify the following USPTO proceeding involving the '781 patent: *Ex Parte* Reexamination Control No. 90/019,244, filed by Unified Patents. Pet. 72; Paper 4, 2.

This Decision is entered concurrently with a decision in IPR2024-01347 involving the same parties and related patent US 7,624,302.

D. Technology Overview

The crux of the dispute between the parties involves what it means to “assign[]” or “transfer[]” the “role of boot processor.” *See, e.g.*, Pet. Reply 2–8 (setting out the boot processor argument at a high level), 8–14 (setting out specific arguments with respect to the art); PO Sur-reply 5–7 (setting out the boot processor argument at a high level), 7–13 (setting out specific arguments with respect to the art). We will first provide some context as to what a boot processor does, then provide an overview of the '781 patent.

1. General Background on Boot Processors

A boot processor is responsible for turning on the computer system: testing internal components, checking memory integrity, identifying the system properties, loading the operating system, and initializing other resources. *See, e.g.*, Ex. 1023, 1:31–35; Ex. 1018, 1:35–40 (each of which are cited by Petitioner’s declarant, Dr. Brogioli, in Exhibit 1002 ¶ 44).

Multiprocessor systems present a unique issue—system startup generally only permits safe operation of one processor, as the multiprocessor coordination features and shared resources are not yet operational. Ex. 1014 ¶ 2. Because a multiprocessor system is unlikely to operate properly with each processor accessing resources at the same time, there needs to be a way to coordinate which processor performs the boot process, for which the prior art offers various solutions. For example, a boot processor may be selected based on which physical location it is attached to. Ex. 1014 ¶ 4 (stating that IA-32 systems select a boot processor “based on the physical location of the processor”); Ex. 1018, 2:1–6 (pre-selecting a boot processor based on physical slot/location).¹ Alternatively, a boot processor may be selected based on it performing a certain task first. Ex. 1014 ¶ 8 (stating some architectures use an “arbitrary” first-to-do-X process); Ex. 1018, 1:44–58, 2:7–14 (describing more “race” procedures). Some systems use specialized firmware. Ex. 1014 ¶ 6 (stating that Intel Itanium processors “rely on a multiprocessor safe firmware to select a [bootstrap processor]”).

However selected, the boot processor begins the boot tasks, which include building tables of various hardware devices so that the operating system knows what hardware it has available to it and where to find that hardware. Ex. 1023, 1:25–39; Ex. 1014 ¶ 3. The ’781 patent provides an example process for booting the operating system in an Itanium processor

¹ In computer hardware, there are *physical* and *logical* devices. For example, if physical CPU0 is in slot 0 in a motherboard and physical CPU1 is in slot 1, but then CPU0 is not working, CPU1 may be designated the first logical CPU, i.e. CPU0. Ex. 1011, 2:9–36. Thus, the actual physical processor that the operating system considers CPU0 is arbitrary and can be changed.

family (IPF) system. Ex. 1001, 11:60. Processors other than the main processors perform initial “sanity checks” and pass control to the boot processor.² *Id.* at 11:60–67. The boot processor then has access to ROM (read-only memory) to begin executing code, including code to find all of the processors and other hardware in the system. *Id.* at 11:65–12:12. Various firmware runs on the boot processor, including an extended firmware interface (EFI) layer, which begins loading the operating system. *Id.* at 12:8–26. The firmware creates ACPI (Advanced Configuration and Power Interface) tables so that the operating system can take control of the various hardware devices, including the processors. *Id.* at 12:24–31, 12:59–65. However, because these tables are used to *set up* the operating system, “they are never updated as things change.” *Id.* at 12:66–13:1. This is relevant because, as the system operates, the firmware is handling the swapping of processors when loss of lockstep occurs. *Id.* at 8:43–48 (describing firmware 18 accessing a device tree specifying which processor is the boot processor or spare processor), 9:3–10:42 (describing how the firmware and operating system interact to handle processor swapping). For example, the ’781 patent uses a device tree to keep track of the processors, including which is the boot processor and which is the spare (while keeping the spare hidden from the operating system by not listing it in the ACPI table). *Id.* at 13:20–55.

Not to be confused with ACPI, APIC is an Advanced Programmable Interrupt Controller. This is a controller for interrupt handling from various

² As we learned from Exhibit 1014, Itanium processors “rely on a multiprocessor safe firmware to select a [bootstrap processor].” Ex. 1014 ¶ 6.

sources, interrupts being a well-known way to notify the processor of some important task or event. *See, e.g.*, Ex. 1008, 268 (Intel developer manual’s chapter on APIC, listing various interrupt sources). The ACPI table will include the APIC ID of the various processors so that the system can use the APIC to route interrupts to the proper locations. *See id.* at 240 (processors adding their APIC ID to the ACPI table during boot), 244–245 (describing the APIC ID format); Ex. 1005 ¶ 26 (noting that “an APIC . . . processes external and normal interrupts that the processor . . . may receive at its interrupt pins”); Ex. 1011 ¶ 21 (describing “an ACPI . . . [A]PIC . . . table, where this table is a structure used by ACPI to describe interrupt routing”).

2. Overview of the ’781 Patent

The ’781 patent discloses a system comprising “a plurality of processors”; data storage for “storing information that assigns a role of boot processor to one of the plurality of processors and assigns a role of spare processor to another of the plurality of processors”; and “logic . . . responsive to detecting loss of lockstep for the boot processor, for transferring, during system runtime, the role of boot processor to the spare processor.” Ex. 1001, code (57).

The ’781 patent explains that “[t]he traditional response to detected LOL [loss of lockstep] has been to crash the system to ensure that the detected error is not propagated through the system.” Ex. 1001, 3:10–13. According to the ’781 patent, “crashing the system each time LOL is detected is not an attractive proposition,” especially for large systems having many processors. *Id.* at 3:16–22. As a solution, the ’781 patent describes techniques for “recovering from LOL detected for a boot processor in a

multi-processor system, in which a spare processor is utilized for such recovery” by assuming the role of boot processor. *Id.* at 4:34–42, 4:50–53.

The ’781 patent describes a multi-processor system storing information (“device tree”) that “identifies the processor in the system that is the boot processor and identifies the processor that is the spare for the boot processor.” Ex. 1001, 4:54–58. Notably, because ACPI tables “are never updated as things change,” the ’781 patent keeps track of the boot processor in the device tree. *Id.* at 12:59–13:35. The ’781 patent also explains that the system’s firmware can access such additional information, and “upon detecting LOL for the boot processor, determine[] the spare processor from the device tree and assign[] the spare processor the role of boot processor.” *Id.* at 4:58–62. According to the ’781 patent, “[t]he firmware then attempts to recover lockstep for the processor for which LOL was detected,” and “[i]f the recovery of lockstep is successful, the processor for which lockstep was recovered (i.e., the former boot processor) may be assigned the role of spare for the current boot processor.” *Id.* at 4:62–67. In this way, the ’781 patent explains, “the multi-processor system may recover from LOL detected for the boot processor without requiring a crash of the system (at least for certain types of ‘recoverable’ LOLs).” *Id.* at 4:67–5:3.

According to the ’781 patent, the spare processor for the boot processor may be a “hot” spare processor, which “is held in reserve, idling until such time as it assumes the role of boot processor responsive to detected LOL for the boot processor.” Ex. 1001, 5:7–10. Alternatively, the spare processor may be an “active” processor that is not held in reserve, but “is made available to the OS for executing instructions during normal system operation,” and “[u]pon LOL being detected for the boot processor, [it] is

dynamically transformed into a hot spare (i.e., it is idled) such that it can assume the role of the boot processor.” *Id.* at 5:10–16. The system in the ’781 patent performs the swap to a spare processor only for boot processors, not for non-boot processors. *See, e.g., id.* at Fig. 1 (item 103), 9:37–48 (process for non-boot processor), 10:43–59 (process for boot processor). The reason the system provides spares for boot processors, but not non-boot processors, is because keeping spares is expensive and to be avoided. *Id.* at 15:21–42. The reason the system uses spares at all, then, is because certain systems require keeping the boot processor active (as far as the system knows), which can be done by substituting in the spare. *Id.* at 10:43–60 (identifying the issue that some systems have with ejecting a boot processor), 5:39–41 (same), 16:16–19 (explaining how the ’781 patent fools the operating system by maintaining the ID of the original boot processor on the spare).

According to the ’781 patent, in response to detection of LOL for the boot processor, the system’s firmware “changes the assignment of the role of boot processor to the spare processor,” where “[t]his change in the role of boot processor occurs dynamically during system runtime, rather than requiring that the system be interrupted (e.g., shutdown and restarted, or crashed).” Ex. 1001, 18:31–36.

E. Challenged Claims

Claims 1–30 (all claims) are challenged. Claims 1, 9, 16, and 25 are independent. Claim 1 is illustrative of the claimed subject matter and is reproduced below.³

³ Paragraph labels provided by Petitioner have been added in brackets.

[1preamble] A system comprising:

[1a] a plurality of processors;

[1b] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors; and

[1c] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor,

[1d] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor.

Ex. 1001, 18:48–60.

F. Prior Art and Asserted Grounds

Petitioner's grounds rely on the following prior art references:

Name	Reference	Exhibit No.
Landry	US Pat. 5,408,647, iss. Apr. 18, 1995	1011
Bigbee	US Pub. 2003/0126498 A1, pub. July 3, 2003	1012
Safford	US Pub. 2004/0006722 A1, pub. Jan. 8, 2004	1006
Jin	US Pub. 2005/0086667 A1, pub. Apr. 21, 2005	1025
Fox	US Pub. 2005/0172164 A1, pub. Aug. 4, 2005	1005
Arai	US Pub. 2006/0036889 A1, pub. Feb. 16, 2006	1007

Petitioner asserts that the challenged claims would have been unpatentable on the following grounds:

Claim(s) Challenged	35 U.S.C. §	Reference(s)/Basis
1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, 30	103	Fox, Safford, Arai
1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, 30	103	Fox, Safford, Landry, Arai
3, 4, 19–21, 27, 28	103	Fox, Safford, Arai, Jin
3, 4, 19–21, 27, 28	103	Fox, Safford, Landry, Arai, Jin

Claim(s) Challenged	35 U.S.C. §	Reference(s)/Basis
8, 24	103	Fox, Safford, Arai, Bigbee
8, 24	103	Fox, Safford, Landry, Arai, Bigbee

Petitioner relies on the testimony of Dr. Michael Brogioli, Ph.D., a professor of Electrical and Computer Engineering who has experience in the fields of “computer architecture, computer hardware, computer firmware, and computer software systems.” Ex. 1002 ¶ 16. Patent Owner relies on the testimony of Mr. Nigel Jones, a consultant who “advises a variety of clients on engineering and intellectual property matters” and has experience “in writing low-level, real-time firmware designed to interact with the hardware features of a microprocessor.” Ex. 2004 ¶¶ 7, 9.

G. Overview of the Prior Art

1. Fox (Ex. 1005)

Fox is titled “Autonomous Fail-Over to Hot-Spare Processor Using SMI.” Ex. 1005, code (54). Fox is directed to “a method and system for dynamically activating a spare processor when processor failure is detected in a multi-processor data processing system.” *Id.* ¶ 2. Fox describes a method and system for dynamically replacing a failing processor using Systems Management Interrupt (SMI) functionality provided within basic input output system (BIOS) of the IA-32 architecture.⁴ *Id.* ¶ 15. At least one processor of the multiprocessor system (MP) is initially provided as a reserve (or hot-spare) processor that remains in an idle, off, or low-power mode. *Id.* While in that mode, the OS is prevented from initially utilizing

⁴ IA (Intel Architecture)-32 refers to an architecture used by Intel in its Pentium and certain other microprocessors. *Barron’s Dictionary of Computer and Internet Terms* 244 (12th ed. 2017) (Ex. 3001).

the hot-spare processor. *Id.* When a processor failure is detected, SMI code running on a good processor instructs the OS to hold off allocating processes to the failing processor. *Id.* Contemporaneously, the SMI (and OS) activates and completes an initialization of the hot-spare processor to prepare it to begin receiving the held-off processes. *Id.* Control is then returned to the OS, which updates the “active” processor list and allocates the threads that were running on the failing processor to the hot-spare processor. *Id.*

2. *Safford (Ex. 1006)*

Safford is titled “Method and Apparatus for Recovery from Loss of Lock Step.” Ex. 1006, code (54). Safford describes a computer system using lock-stepped processor cores that operate in a master/checker pair to improve reliability of processing assets. *Id.* ¶ 15. Each of two processors in the pair processes the same code sequences. *Id.* When a logic circuit compares the resulting outputs of the processors, any difference in the processor outputs indicates the existence of an error, prompting the logic circuit to initiate a sequence of steps to halt operation of the two processors. *Id.*

Figure 3 of Safford is reproduced below.

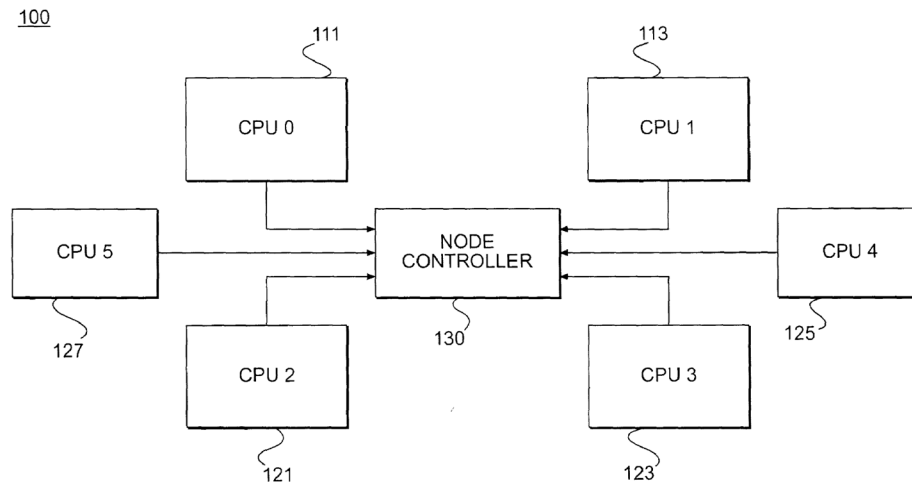


FIG. 3

Figure 3 depicts computer system 100 with processors 111, 113, 121, 123, 125, and 127 that operate as pairs when in lock step (i.e., processors 111 and 113 are a first pair; processors 121 and 123 are a second pair; and processors 125 and 127 are a third pair). Ex. 1006 ¶ 18. The processor pairs are coupled to a lockstep logic, connected to node controller 130. *Id.* According to Safford, “[t]he processor 125 may be designated as a ‘hot standby,’ and is sitting idle in lock step mode with the processor 127.” *Id.* ¶ 20. “Should one of the processors 111, 113, 121, and 123 suffer an error, the hot standby processors 125, 127 may be used to speed recovery from the resulting loss of lock step.” *Id.* ¶ 20.

3. *Arai (Ex. 1007)*

Arai is titled “High Availability Multi-Processor System.” Ex. 1007, code (54). Arai describes “a method and system for reserving a processor in a multiprocessor system for availability of replacement in response to detection of an error in a non-reserved processor.” *Id.* ¶ 2. According to Arai, “[w]hen a central processor in a multiprocessor system encounters an

error, it is very desirable to not lose the work being done on that processor and to move that work to another processor that is still operating in the system.” *Id.* ¶ 5.

In Arai, during system initialization, one or more logical processors may be reserved in an inactive state. Ex. 1007, code (57). In the event an error is detected on a logical or physical processor, the execution context of the processor experiencing the error is transferred to one or more of the reserved logical processors. *Id.* “Thereafter, the active processor is designated as inactive and replaced by the inactive processor to which the execution context has been transferred.” *Id.*; *see also id.* ¶ 19.

Figure 4b of Arai is reproduced below.

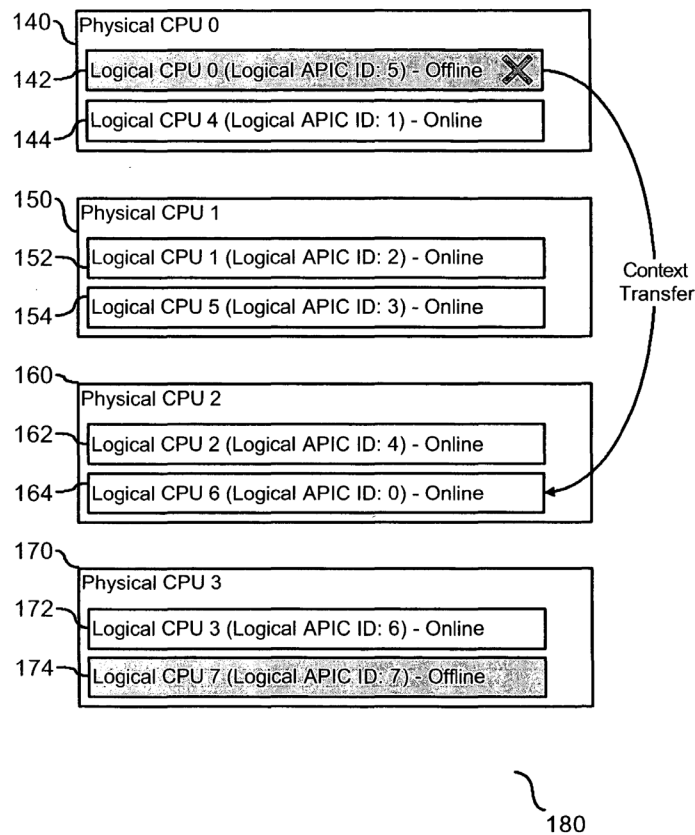


FIG. 4b

Figure 4b depicts the identification and status of each processor following occurrence of an error in a logical processor. Ex. 1007 ¶ 16. Following detection of an error with logical CPU₀ (142), logical CPU₀ is placed in an idle state and assigned the identification number of logical CPU₆ (164). *Id.* ¶ 23. At the same time, logical CPU₆ (164) is placed in an active state and assigned the identification number of CPU₀ (142). *Id.* In this way, in response to an error in an active processor, the execution context of the active processor may be transferred to an identified inactive processor, followed by activation of the processor that has received the execution context. *Id.* ¶ 25.

4. *Landry (Ex. 1011)*

Landry is titled “Automatic Logical CPU Assignment of Physical CPUs.” Ex. 1011, code (54). Landry relates to “startup logic for assigning logical central processing unit (CPU) designations among multiple CPUs.” *Id.* at 1:7–10.

Landry discloses a multiprocessor computer system that “includes fault tolerant power up logic for finding a functioning CPU to operate as logical CPU₀.” Ex. 1011, code (57), 2:9–11. When the system is powered up, all of the CPUs except the CPU in physical slot 0 (CPU P₀) are initially placed in a SLEEP mode. *Id.* at 2:15–18. The microprocessor in physical location 0 performs its power on self-test (POST) routine and tests its associated cache memory system. *Id.* at 2:18–20. If the CPU functions properly, the CPU is designated as CPU L₀, and the microprocessor retains this designation until the system is power cycled. *Id.* at 2:20–23. The CPU designated as CPU L₀ then awakens the remaining CPUs and boots up the rest of the computer system. *Id.* at 2:23–25.

If CPU P0 is not functioning properly, it is designated as inoperative. Ex. 1011, 2:26–27. The system then awakens CPU P1 and repeats the process of testing the CPU. If CPU P1 is operational, then it is designated as CPU L0, and it boots the remainder of the system. *Id.* at 2:27–31. On the other hand, if CPU P1 also fails, it is also given an inoperative designation. The computer system then turns on CPU P2, and repeats the process. *Id.* at 2:31–32. The process repeats until an operational microprocessor is found to perform the CPU L0 functions. *Id.* at 2:33–35. Consequently, if at least one of the microprocessors in the system is functioning, the computer system boots and then operates. *Id.* at 2:35–37. Thus, the multiprocessor system is not crippled by the failure of a single processor, enhancing the dependability of the computer system. *Id.* at 2:37–40.

5. *Jin* (Ex. 1025)

Jin is titled “Symmetric Scheduling for Parallel Execution.” Ex. 1025, code (54). *Jin* describes that system initiation may include the execution of certain drivers, such as EFI (Extensible Firmware Interface) drivers, prior to booting the operating system. *Id.* ¶ 2. *Jin* discloses a symmetric scheduling algorithm that “doesn’t differentiate between a bootstrap processor (BSP) and other processors in dispatching EFI drivers in parallel, thereby allowing efficient parallel operations.” *Id.* ¶ 18. According to *Jin*, “[t]he symmetric dispatch of drivers may include the operation of both a bootstrap processor that is responsible for the initiation of the system and other system processors, which may be referred to herein as application processors (APs).” *Id.* ¶ 19.

Jin describes “two phases of platform initialization in EFI-based BIOS (basic input output system)”: (1) Pre-EFI Initialization (PEI) phase, which is

“responsible for initializing the minimum system resources for enablement of the DXE phase”; and (2) Driver Execution Environment (DXE) phase, wherein “various drivers are executed collectively to initialize the platform into the final pre-boot state.” Ex. 1025 ¶¶ 22–25. According to Jin, “[t]he DXE core is responsible for discovering the drivers and executing the drivers in a correct order” and “thus may be considered to be a driver dispatcher.” *Id.* ¶ 25.

In Jin, during DXE core initialization, application processors are initialized to be “available at the very beginning of the DXE phase,” drivers are detected, and a discovered driver queue is built. Ex. 1025 ¶ 26. After DXE core initialization, “each processor, including the bootstrap processor and each application processor, attempts to identify a driver ready for execution from the discovered driver queue.” *Id.* “Once such a driver is identified, the processor executes the driver. After the execution of the driver, the processor returns to the discovered driver queue to attempt to obtain another driver ready for execution.” *Id.* According to Jin, this “process continues until all the drivers are dispatched from the discovered driver queue,” and “[a]t this point, the bootstrap processor is ready to boot the operating system (OS).” *Id.*

6. *Bigbee (Ex. 1012)*

Bigbee is titled “Method and Apparatus for Functional Redundancy Check Mode Recovery.” Ex. 1012, code (54). Bigbee describes data processing systems where “two or more physical processing elements or ‘cores’ are associated with and function as a single logical processor.” *Id.* ¶ 2. Bigbee discloses that “[t]he processor cores are operated in a functional redundancy check (FRC) mode in which identical instructions are provided

to each core and concurrently executed on identical data so that any error occurring within a single core will produce an inconsistent result as compared to the remaining cores associated with a given processor.” *Id.* “Specialized FRC logic performs this comparison and generates a signal such as an interrupt or exception once an inconsistent result has been detected.” *Id.*

II. ANALYSIS

A. Legal Standards Used in the Merits Analysis

“In an IPR, the petitioner has the burden from the onset to show with particularity why the patent it challenges is unpatentable.” *Harmonic Inc. v. Avid Tech., Inc.*, 815 F.3d 1356, 1363 (Fed. Cir. 2016) (citing 35 U.S.C. § 312(a)(3) (requiring *inter partes* review petitions to identify “with particularity . . . the evidence that supports the grounds for the challenge to each claim”)); *Dynamic Drinkware, LLC v. Nat’l Graphics, Inc.*, 800 F.3d 1375, 1378 (Fed. Cir. 2015) (discussing the burden of proof in *inter partes* review).

A claim is unpatentable under 35 U.S.C. § 103 if “the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious . . . to a person having ordinary skill in the art to which said subject matter pertains.” *KSR Int’l Co. v. Teleflex Inc.*, 550 U.S. 398, 406 (2007). The question of obviousness is resolved on the basis of underlying factual determinations, including: (1) the scope and content of the prior art; (2) any differences between the claimed subject matter and the prior art; (3) the level of skill in the art; and (4) when in evidence, objective indicia of non-obviousness. *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 17–18 (1966). To

support this conclusion, however, it is not enough to show merely that the prior art includes separate references covering each separate limitation in a challenged claim. *Unigene Labs., Inc. v. Apotex, Inc.*, 655 F.3d 1352, 1360 (Fed. Cir. 2011). Rather, obviousness additionally requires that a person of ordinary skill at the time of the invention “would have selected and combined those prior art elements in the normal course of research and development to yield the claimed invention.” *Id.*

Moreover, in determining the differences between the prior art and the claims, the question under 35 U.S.C. § 103 is not whether the differences themselves would have been obvious, but whether the claimed invention as a whole would have been obvious. *Litton Indus. Prods., Inc. v. Solid State Sys. Corp.*, 755 F.2d 158, 164 (Fed. Cir. 1985) (“It is elementary that the claimed invention must be considered as a whole in deciding the question of obviousness.”); *see also Stratoflex, Inc. v. Aeroquip Corp.*, 713 F.2d 1530, 1537 (Fed. Cir. 1983) (“[T]he question under 35 U.S.C. § 103 is not whether the differences *themselves* would have been obvious. Consideration of differences, like each of the findings set forth in *Graham*, is but an aid in reaching the ultimate determination of whether the claimed invention *as a whole* would have been obvious.”). As a factfinder, we also must be aware “of the distortion caused by hindsight bias and must be cautious of arguments reliant upon *ex post* reasoning.” *KSR*, 550 U.S. at 421.

B. Level of Ordinary Skill in the Art

The level of skill in the art is “a prism or lens” through which we view the prior art and the claimed invention. *Okajima v. Bourdeau*, 261 F.3d 1350, 1355 (Fed. Cir. 2001). “This reference point prevents . . . factfinders

from using their own insight or, worse yet, hindsight, to gauge obviousness.”
Id.

“The *Graham* analysis includes a factual determination of the level of ordinary skill in the art. Without that information, a . . . court cannot properly assess obviousness because the critical question is whether a claimed invention would have been obvious at the time it was made to one with ordinary skill in the art.” *Custom Accessories, Inc. v. Jeffrey-Allan Indus., Inc.*, 807 F.2d 955, 962 (Fed. Cir. 1986); *see also Ruiz v. A.B. Chance*, 234 F.3d 654, 666 (Fed. Cir. 2000) (“The determination of the level of skill in the art is an integral part of the *Graham* analysis.”).

Factors pertinent to a determination of the level of ordinary skill in the art include: (1) educational level of the inventor; (2) type of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of workers active in the field. *Env’t Designs, Ltd. v. Union Oil Co.*, 713 F.2d 693, 696–697 (Fed. Cir. 1983) (citing *Orthopedic Equip. Co. v. All Orthopedic Appliances, Inc.*, 707 F.2d 1376, 1381–82 (Fed. Cir. 1983)). Not all such factors may be present in every case, and one or more of these or other factors may predominate in a particular case. *Id.* Moreover, these factors are not exhaustive but are merely a guide to determining the level of ordinary skill in the art. *Daiichi Sankyo Co. v. Apotex, Inc.*, 501 F.3d 1254, 1256 (Fed. Cir. 2007). In determining the level of ordinary skill, we also may look to the prior art, which may reflect an appropriate skill level. *Okajima*, 261 F.3d at 1355.

Petitioner asserts that:

A person of ordinary skill in the art (“POSITA”) would have at least a bachelor’s degree in computer science, computer

engineering, or equivalent, and at least two years of prior work experience with computer architecture design, including fault-tolerant computer architecture design, as of the earliest priority date of the '781 Patent—October 25, 2004. [Ex. 1002] ¶¶28–30. Additional education could substitute for professional experience and vice versa. *Id.*

Pet. 5.

Patent Owner asserts that:

A person of ordinary skill in the art in the field of the '781 patent would have been a person with at least a bachelor's degree in Electrical Engineering, Computer Engineering, or equivalent, and at least two years of prior work experience with computer architecture, including experience relating to fault tolerant computer architecture, as of the earliest priority date of the '781 patent—October 25, 2004. If a person did not have a bachelor's degree, additional relevant and practical experience with fault tolerant computer systems could provide an acceptable substitute. Likewise, further education beyond a bachelor's degree could compensate for a deficiency in practical experience. *See* EX2004, Declaration of Nigel A. Jones ("Jones"), ¶ 26.

PO Resp. 3. Patent Owner further notes that its position is "similar, but not identical to, Petitioner's [position]" but that "the analysis [of the claims] is identical for both proposed determinations." *Id.* We agree that our analysis would not differ based on choosing one proposed level of skill over the other, but as the losing party, we select Petitioner's proposed level of skill.

C. Claim Construction

As stated in 37 C.F.R. § 42.100(b),

a claim of a patent . . . shall be construed using the same claim construction standard that would be used to construe the claim in a civil action under 35 U.S.C. 282(b), including construing the claim in accordance with the ordinary and customary meaning of such claim as understood by one of ordinary skill in the art and the prosecution history pertaining to the patent. Any prior claim construction determination

concerning a term of the claim in a civil action, or a proceeding before the International Trade Commission, that is timely made of record in the *inter partes* review proceeding will be considered.

37 C.F.R. § 42.100(b). Thus, under 37 C.F.R. § 42.100(b), we apply the claim construction standard set forth in *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312–13 (Fed. Cir. 2005) (en banc). As stated in *Phillips*, “the words of a claim ‘are generally given their ordinary and customary meaning’ . . . that the term would have to a person of ordinary skill in the art in question at the time of the invention.” *Id.* at 1312–13.

The parties do not explicitly offer claim constructions, but the dispute between the parties is premised on their reading of the claims on the prior art and each provides extensive arguments on that matter. We focus our analysis on claim 1 because Petitioner largely bases its obviousness grounds for the other independent claims on its claim 1 analysis. *See* Pet. 38–50 (analyses of claims 9, 16, and 25, each largely referring back to the claim 1 analysis). In addition, neither party argues nor bases its position on a difference between the independent claims. *See* PO Resp. 19–20 (referring back to claim 1 for its arguments on claims 9, 16, and 25); PO Sur-reply 16–17 (same); Pet. Reply 18 (not separately discussing claims 9, 16, or 25).

Petitioner’s position is that “transferring, during system runtime, the role of boot processor to said spare processor” (in limitation 1[c]) requires the system to substitute the spare processor for the boot processor during runtime—that the spare processor takes over the processing tasks of the boot processor. Tr. 23:1–2 (Petitioner’s counsel asserting, “[t]he claims simply recite switching the role of boot processor to the spare processor during system runtime”); *see also* Pet. Reply 3–8 (analyzing the claim language).

Patent Owner’s position is that the limitation requires the system to substitute the spare processor for the boot processor not only to take over for the boot processor during runtime, as Petitioner proposes, but also to be specifically designated the boot processor in the system so that it is recognized as such for a future reboot. Tr. 49:2–5 (Patent Owner’s counsel asserting that, “to be transferred the role of boot processor, [it] means that it has to be marked or flagged or dedicated to play the role of boot processor, which is to boot the system, load the operating system, initialize things . . . at some point.”), 50:18–51:9 (similar); *see also* PO Resp. 10–11 (asserting that “the ‘role of boot processor’ relates to what occurs in the system during the boot process”); PO Sur-reply 5–7 (further argument); *but see* Pet. Reply 2–3 (Ppetitioner asserting that Patent Owner’s position, “requiring the spare processor—the new boot processor—to perform system initialization after system reboot or reset,” is “wrong”), 13–14 (similar characterization of Patent Owner’s position). Reviewing the claims in light of the evidence, we determine that the correct reading is somewhere in between the parties’ positions.

Starting with claim 1, we note that it requires storing information that “assigns a role of boot processor” as well as logic “for transferring, during system runtime, the role of boot processor to [a] spare processor” when it is determined that there is potentially a problem with the processor. Notably, claim 1 specifically calls out a *boot* processor, not just any processor, and focuses on transferring the *role* of boot processor, not just its tasks, to a spare processor during system *runtime*. Thus, there are four critical components: (a) a *boot* processor, (b) a *spare* processor, (c) a transfer of the *role* of boot processor, and (d) the transfer to the spare occurring during

runtime. We review the specification for language discussing transferring the role of boot processor during runtime to a spare.

The '781 patent states that not all processors need to have a spare waiting or ready to be created in the event of loss of lockstep. For example, in the event of loss of lockstep in “non-boot processors . . . the lockstep recovery process . . . can be utilized without requiring a spare for lockstep recovery for those non-boot processors.” Ex. 1001, 15:32–42. That is, boot processors are treated differently from non-boot processors in that non-boot processors do not need a spare. *Id.* at 15:21–34 (explaining that holding spares for non-boot processors is “undesirably expensive and wasteful”); *see also id.* at 5:45–55 (“[T]he spare processor is used for recovering from [loss of lockstep] only in the case in which [loss of lockstep] is detected for the boot processor. For instances in which [loss of lockstep] is detected for non-boot processors, the lockstep may be recovered without the use of a spare.”), 8:57–9:3 (similar), 9:18–36 (similar). Non-boot processors are recovered without a spare, and are simply idled or ejected from the system until recovered. *Id.* at 9:37–10:42.

The reason why the '781 patent treats boot processors differently during runtime by providing a spare is because, “for various reasons, in certain system architectures[,] problems arise in attempting to idle (or eject) the boot processor from the system.” *Id.* at 5:39–43; *see also id.* at 10:45–50 (same). Thus, we find that the system in the '781 patent treats the boot processor differently during runtime because the *system architecture* it interfaces with treats processors identified as the boot processor differently. The process of recovering a boot processor in such systems requires the use of a spare. Therefore, because there is a problem *unique to boot processors*

during runtime unrelated to booting, the system in the '781 patent needs to keep track of which processor was the boot processor, for reasons other than booting.

This is why the '781 patent describes the firmware working in the background to manipulate the device tables to figure out if the problem is with the boot processor during runtime, and if so, identify a spare processor, free it up, and then modify the device tables so that the operating system thinks the spare processor is the boot processor. *See, e.g., id.* at 13:20–14:3 (describing a table or “device tree” of the available processors, their status, and their roles), 15:8–20 (describing how the swap is done through a “LID register”), 16:11–16 (describing how the LID register swap makes one CPU appear as another to the operating system), 15:60–16:4 (describing how the firmware can free up a working processor predesignated to be the spare by sending a “lie to the OS by indicating . . . that [the spare] is not functioning”); 17:32–55 (generally describing the swap-out procedure). In effect, the operating system continues to believe it is working on a certain processor, but in the background the firmware has swapped the processors, made possible by the fact that the operating system relies on the information provided by the firmware regarding the location of the system’s hardware. *Id.* at 13:1–2; *see also id.* at 9:42–48 (using ACPI allows firmware to cooperate with the operating system, such that “no processor or platform specific knowledge is required to be embedded in [the operating system]”), 16:11–21 (describing the mechanism for fooling the operating system as to which processor is which). In sum, the reason to go through this trouble is because runtime identification of a processor as a boot processor is

important in order to accommodate certain system requirements to treat boot processors differently.

Accordingly, when claim 1 recites “storage” and “logic” for transferring the “role of boot processor” during runtime to the spare processor, we determine that the claim is describing this transfer of *identification* of “boot processor,” which is required when transferring a boot processor’s functions to a spare because of the “problems [that] arise in attempting to idle (or eject) the boot processor.” *Id.* at 5:39–41. Thus, the claimed storage is the information used by the firmware and operating system to exchange this information identifying the boot processor, which is described in the specification as a device tree. *See, e.g., id.* at 13:20–35.

The ’781 patent makes clear that accurate identification of which processor is the boot processor remains critical during runtime. *See, e.g., id.* at 5:39–41. Thus, Patent Owner is correct in requiring the system to have a way to identify the boot processor and to transfer this identification during runtime. However, as made clear in the embodiments described in the ’781 patent discussed above, this identification is for runtime purposes to accommodate runtime problems, not for booting purposes. Patent Owner points to a few passages in the specification as allegedly supporting its position that the claims enable a system to designate a boot processor *for future reboots*. *See* PO Resp. 10–11 (citing Ex. 1001, 14:4–31, 13:36–55, 17:45–55); PO Sur-reply 5–7 (similar). None of these citations points to an embodiment that transfers the role of boot processor for the purpose of setting the processor for the next boot cycle. Although Patent Owner may be correct that the “primary function” of the boot processor is to boot the system (PO Sur-reply 5), the problem identified in the ’781 patent is a

problem relating to the runtime reassignment of the boot processor identity, to address a runtime need to identify the boot processor. *See, e.g.*, Ex. 1001, 10:45–48. Further, although the '781 patent has a passage that “discloses that changes to processor roles can change settings in NVRAM that are ‘honored on the next reset’” (PO Sur-reply 6 (quoting Ex. 1001, 14:28–29)), this passage is not specific to changing the role of boot processor but rather is a generic capability of the system “that controls how partitions are configured, memory interleave, CPU configuration, etc.” (Ex. 1001, 14:20–31). *See also id.* at 12:54–56 (similar). Still further, this feature is not described in the context of a boot processor being switched to a spare after loss of lockstep (as is claimed) but rather is in a section describing the features of a utility interface. *Id.* at 14:4–31. At best, Patent Owner points to features in the '781 patent that might be able to be used to force the boot processor selection on reboot, but the '781 patent does not describe or suggest an embodiment that uses these tools in this way.

Accordingly, based on our analysis of the claims and specification, we determine that “logic . . . for transferring, during system runtime, the role of boot processor” means logic that causes the system to identify the spare processor as the boot processor (e.g., in a device tree) which then causes the spare processor to appear as the boot processor to the system during runtime. Ex. 1001, 5:39–55, 10:43–67 (describing why boot processors are treated differently even after the boot process has completed); *compare id.* at 9:37–10:42 (process for recovering from loss of lockstep of a non-boot processor without using a spare), *with id.* at 10:43–11:42 (process for recovering from loss of lockstep of a boot processor using a spare because boot processors, during runtime, need special treatment); *see also* 15:21–42 (emphasizing

that spares are only needed for recovering boot processors). We construe transferring a “role of boot processor” to mean transferring an identification in the system of which processor is the boot processor. This construction comports with the embodiments described in the specification. We do not adopt an additional requirement that this procedure also requires designating which processor will boot the system in the next reboot, which is not disclosed in the specification.

Based on the clear support in the specification for the above interpretation, we see no need to resort to extrinsic evidence. *Seabed Geosolutions (US) Inc. v. Magseis FF LLC*, 8 F.4th 1285, 1287 (Fed. Cir. 2021) (“If the meaning of a claim term is clear from the intrinsic evidence, there is no reason to resort to extrinsic evidence.”) (citing *Profectus Tech. LLC v. Huawei Techs. Co.*, 823 F.3d 1375, 1380 (Fed. Cir. 2016) (“Extrinsic evidence may not be used ‘to contradict claim meaning that is unambiguous in light of the intrinsic evidence.’” (quoting *Phillips*, 415 F.3d at 1324))). In general, the cited portions of testimony from the declarants from both parties in the briefs do little more than point to the portions of the ’781 patent described above or argue about what a boot processor does during booting. *See, e.g.*, Pet. Reply 2–8. The specification makes clear that it is focused on the role of boot processor during runtime because of problems encountered during runtime; it is not focused on the boot processor during booting. Ex. 1001, 5:39–55, 10:43–67. Thus, the testimony about what a boot processor does during booting, or that the primary function of the boot processor is to boot, is not particularly pertinent, and does not undo the unambiguous discussion in the ’781 patent.

D. Availability of Exhibits 1008 and 1028

Exhibit 1008 is titled “IA-32 Intel Architecture Software Developer’s Manual” and Exhibit 1028 is titled “MultiProcessor Specification.”

Ex. 1008, 1; Ex. 1028, 1.⁵ Petitioner cites Exhibits 1008 and 1028 for its analysis of the claims, e.g., for what a person of ordinary skill in the art would understand a certain disclosure in the prior art to mean, but does not rely on them as a listed reference in the ground. *See, e.g.*, Pet. 22–23 (undisputed prior art Fox discloses an “IA-32” processor and Petitioner providing Exhibit 1008, the IA-32 developer’s manual, as evidence of a feature of the IA-32 processor). Patent Owner does not seek to exclude the references, but does assert that we should give “no weight” to these disclosures. The USPTO has established Rules for objecting to and challenging evidence. 37 C.F.R. § 42.64. This process sets various timeframes and permits briefing and curative actions, all of which are designed to crystallize the issues and ensure a full and fair opportunity for the parties to be heard. *See id.* Accordingly, we will not exclude Exhibits 1008 or 1028 on the basis of their authenticity, admissibility, or for any reason that should have been raised under the proper procedures. *See, e.g.*, PO Resp. 7–8; PO Sur-reply 7–8; *see also* Pet. Reply 11 (asserting that “Patent Owner has not filed any objection to or a motion to exclude these references.”). We consider Petitioner’s citations to these documents in our prior art analysis below. Notwithstanding, as is clear from our patentability analysis below, Exhibits 1008 and 1028 do not serve to demonstrate the unpatentability of any claim.

⁵ We follow the parties’ convention of citing to the later-added “Page X of Y” pagination rather than the native pagination.

*E. Asserted Obviousness of
Claims 1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, and 30
Based on Fox, Safford, Landry, and Arai*

Petitioner contends that claims 1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, and 30 would have been obvious over Fox, Safford, and Arai (Pet. 9–32, 38–61), and alternatively, over Fox, Safford, Landry, and Arai (*id.* at 33–37).⁶ Petitioner presents supporting testimony by Dr. Brogioli. Ex. 1002 ¶¶ 84–233. We will focus our discussion on independent claim 1, which illustrates a dispositive issue.

1. Overview of Petitioner’s Ground for Independent Claim 1

Petitioner presents an element-by-element analysis of claim 1, supported by testimony of Dr. Brogioli, as well as an analysis of a reason to combine the teachings of the references. Pet. 10–32, 36–37; Ex. 1002 ¶¶ 89–143, 151–158.

*a) Reason to Combine Fox, Safford, and
Arai and Fox, Safford, Landry, and Arai*

Petitioner asserts that a person of ordinary skill “would have found it obvious to combine Fox’s multiprocessor system with Safford’s lockstep processor and error detection logics and Arai’s processor-switching implementation details.” Pet. 10. Dr. Brogioli testifies that “[a person of ordinary skill] would have been motivated to implement the processors in Fox as lockstep processors to improve reliability and availability of Fox’s server system for critical processes, for which higher reliability and availability of the system are desired.” Ex. 1002 ¶ 90. He further testifies

⁶ For claims 2, 5–7, 9–18, 22, 23, 25, 26, 29, and 30, Petitioner combines its analysis of the two grounds. *See* Pet. 38–61.

that “[a person or ordinary skill] also would have been motivated to implement in Fox the error detection logics that detect loss of lockstep together with the lockstep processors as taught by Safford to implement lockstep functions and further improve availability of the system.” *Id.* He testifies that “[a person of ordinary skill] would have understood how to implement Fox’s processors as lockstep processors with error detection logics and expect success. [A person of ordinary skill] would only have applied well-known configurations and ordinary hardware and software configuration skills.” *Id.* ¶ 91. Dr. Brogioli provides additional reasons motivating the combination. *Id.* ¶¶ 92–99.

Petitioner contends that “Arai discloses switching the role of a failing processor with a spare processor.” Pet. 10. Dr. Brogioli testifies that “[a person of ordinary skill] would have been motivated to incorporate Arai’s teaching of transferring processor context from the failing processor to the spare processor not to interrupt the operating system and its running applications as desired by Fox.” Ex. 1002 ¶ 109. He reasons that “[a person of ordinary skill] would have understood that incorporating Arai’s processor-switching implementation details into Fox would provide the benefit of switching the role of the failing processor to the spare processor without affecting the operating system and running applications, which is expressly taught as desirable by both Fox and Arai and well known in the art.” *Id.* ¶ 110.

Petitioner asserts that a person of ordinary skill also would have been motivated to further combine these references with Landry. Pet. 33–35. Dr. Brogioli testifies that “Landry discloses a fault-tolerant boot process that dynamically designates a functional CPU as logical CPU0 that performs the

unique functions of a boot processor.” Ex. 1002 ¶ 147. He testifies further that “[a person of ordinary skill] would have understood Landry’s process would have been a suitable option for selecting a boot processor in Fox’s system consistent with the initialization protocol of the IA-32 multiprocessor system.” *Id.* He asserts that “[a person of ordinary skill] would have understood the benefits of implementing Landry’s boot processor selection process in Fox.” *Id.* ¶ 148. Among other reasons, “[a person of ordinary skill] would have further understood that using the logical CPU0 as the identification for the boot processor allows for transparent replacement of the physical processor experiencing a failure with another processor without interrupting the operating system.” *Id.*

b) [1preamble]: “A system comprising:”

Petitioner relies on Fox as disclosing the preamble of claim 1. Pet. 20, 36.⁷

c) [1a]: “a plurality of processors;”

Petitioner relies on Fox’s multiprocessor computer system to meet this limitation. Pet. 21, 36.

d) [1b]: “data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors; and”

In its first ground, Petitioner relies on Fox and Arai to meet this limitation. Pet. 22–24. Petitioner’s position is multilayered. First, Petitioner notes that Fox uses an IA-32 system, such that a person of ordinary skill in the art would have understood that one processor would

⁷ We do not express an opinion on whether the preamble recitations of the challenged claims are limiting.

have been designated the boot processor. *Id.* at 21 (citing, *e.g.*, Ex. 1005 ¶ 23), 22 (citing Ex. 1005 ¶¶ 6, 34; Ex. 1008, 238–239). Specifically, Fox states that “[t]he present invention provides a method for dynamically replacing failing processors in a[n] IA-32 configured multiprocessor computer system.” Ex. 1005 ¶ 23. The manual for the IA-32 system, provided by Petitioner, discloses a “BSP flag,”⁸ which can be found in the processor’s IA32_APIC_BASE register. Ex. 1008, 238; *see also id.* at 710 (showing that bit 8 of register address 27 is the BSP flag); Pet. 22 (asserting that “Fox’s IA-32 system would have set the BSP flag of the designated boot processor”). Petitioner also asserts that “Fox saves processor identifications in using a register or table (*e.g.*, BIOS table, ACPI table, MP table).” Pet. 22 (citing Ex. 1005 ¶¶ 26, 33, 35, 38, 41, 45–46). Then, Petitioner asserts that, although “Fox does not expressly disclose the processor register or table is stored in the system memory,” “Arai discloses this.” *Id.* at 23 (citing Ex. 1007 ¶¶ 19–22, 25–26). In particular, Petitioner asserts that “Arai discloses that during system initialization, the BIOS creates a table (*e.g.*, ACPI table or MP table) that identifies each processor in the system and its designated operational state, *i.e.*, active or disabled.” *Id.* Dr. Brogioli testifies that “[a person of ordinary skill] would have found it suitable to store Fox’s register or table that stores information assigning the roles of boot and spare processors in its system memory (‘data storage’), like in Arai.” Ex. 1002 ¶ 122.

In its second ground, Petitioner relies on Fox, Landry, and Arai to meet limitation [1b]. Pet. 36. Petitioner contends that “Fox in view of

⁸ BSP: Bootstrap processor. *See, e.g.*, Ex. 1008, 238.

Landry further renders obvious assigning a role of boot processor—logical CPU0—to one of said plurality of processors.” *Id.* Dr. Brogioli testifies that “a [person of ordinary skill] would have implemented Landry’s boot processor selection process in Fox and identified logical CPU0 in Fox as the boot processor (‘assigns a role of boot processor’).” Ex. 1002 ¶ 155.

Petitioner also refers back to the first ground, and asserts that there it explained that a person of ordinary skill in the art “would have understood that the identifications and software tags of the processors, including those of the boot processor and spare processor (‘information’), would be stored in Fox’s register or table” and “to store Fox’s register or table that stores information assigning the roles of boot and spare processors in its system memory (‘data storage’), like in Arai.” Pet. 36.

e) [1c]: “*logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor,*”

In its first ground, Petitioner relies on Fox, Safford, and Arai to meet this limitation. Pet. 24–30. Petitioner asserts that “Fox does not expressly disclose that its processors are lockstep processors or detecting loss of lockstep (‘LOL’). Safford does.” *Id.* at 24 (citing Ex. 1002 ¶¶ 64–65, 127–129).

Petitioner asserts that “Arai discloses replacing an active processor detected with an error with a reserved inactive processor in a multiprocessor system.” Pet. 28 (citing Ex. 1007, Abstr., ¶¶ 2, 8–9, 19–24). Petitioner explains that “Arai discloses that during system initialization, the BIOS creates a table that identifies each processor in the system and its designated operational state, i.e., active or disabled.” *Id.* (citing Ex. 1007 ¶¶ 19–22). Petitioner continues, “[i]f the BIOS determines the error requires

replacement of the processor, and if there is a reserved inactive processor, the identification numbers of the active processor in error and the reserved inactive processor are switched.” *Id.* (citing Ex. 1007 ¶¶ 22–23, 25).

Petitioner further asserts that Arai discloses “transfer[ring] of the execution context from the active processor that is experiencing the error to at least one of the inactive processors that has been designated to replace the active processor.” *Id.* at 29 (citing Ex. 1007 ¶¶ 22, 25, Fig. 4b) (internal quotation marks omitted) (alteration in original).

Dr. Brogioli testifies that “[a person of ordinary skill] would have understood that in Fox as modified by Safford, in response to detecting loss of lockstep for the boot processor, implementing Arai’s identification switching and context transfer in the switching process would have reassigned the processor logical identification and responsibilities of the failing boot processor pair to the spare processor pair (‘transferring . . . the role of boot processor to said spare processor’) while the operating system processes continue to execute (‘during system runtime’).” Ex. 1002 ¶ 137 (citing Ex. 1005 ¶¶ 8, 10, 13–14; Ex. 1007 ¶ 6). He further testifies that “[a person of ordinary skill] would have understood that incorporating Arai’s processor-switching implementation detail would improve availability and reliability of the multiprocessor system in Fox as modified by Safford” and “would have expected success in such modification because it merely involves using known techniques to improve a similar device in the same way taught in the prior art.” *Id.* ¶ 138.

Petitioner’s second ground relies on Fox, Safford, Landry, and Arai for limitation [1c] and refers to the above analysis for this limitation in Petitioner’s first ground and the above discussion of limitation [1b]. Pet. 37.

f) [1d]: “wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor.”

In its first ground, Petitioner relies on Fox and Safford to meet this limitation. Pet. 30–32. Petitioner asserts that “Fox discloses that at least one processor of the multiprocessor system is reserved as a hot-spare processor that remains in an idle, off, or low-power mode (‘held in reserve as a hot spare processor’),” and “when a processor failure is detected, . . . the BIOS switches the failing processor to the active hot-spare processor.” *Id.* at 30–31 (citing Ex. 1005, Abstr., ¶¶ 15, 40–41, Fig. 1). Petitioner asserts that Safford likewise “discloses that the spare processor pair 125/127 is designated as ‘a hot standby’ (‘hot spare’) and ‘sitting idle in lock step mode’ (‘held in reserve’),” and “when loss of lockstep is detected in the processor pair 111/113, the architecture state of processor pair 111/113 is copied to the hot standby processor pair 125/127, which becomes an active processor.” *Id.* at 32 (citing Ex. 1006 ¶¶ 18, 20–21). Dr. Brogioli testifies that “[a person of ordinary skill] would have understood that in Fox combined with Safford, Fox’s hot spare processor would be a hot spare processor pair ‘held in reserve’ in lockstep mode until loss of lockstep is detected for the boot processor pair, as taught in Fox and Safford.” Ex. 1002 ¶ 143.

Petitioner’s second ground relies on Fox, Landry, and Safford for limitation [1d] and refers to the above analysis for this limitation in Petitioner’s first ground and the above discussion of limitation [1b]. Pet. 37.

2. Patent Owner’s Contentions and Petitioner’s Reply

Patent Owner’s position is based on what it means to “assign[]” and “transfer[]” a “role of boot processor.” Patent Owner argues that merely

handing off the tasks being performed by the processor that happened to boot the system is not the same as “assigning the role of boot processor to the spare processor.” PO Resp. 9–11. In technical terms, as Patent Owner’s declarant, Mr. Jones, states: “Petitioner provides no evidence that changing the logical APIC ID of the processor in error to the logical APIC ID of the reserved processor has any impact on which processor performs any boot processor role.” Ex. 2004 ¶ 34. Patent Owner further explains that “APIC IDs in Arai are stored in RAM,” i.e., volatile memory storage. PO Resp. 10 (citing Ex. 1007 ¶¶ 20–21; Ex. 2004 ¶¶ 35–36; Ex. 2006, 103:14–21, 108:10–109:4). Patent Owner contrasts this temporary, task-based handover with what is allegedly disclosed in the ’781 patent, which stores the assignment “in NVRAM that [is] ‘honored on the next reset,’” and that the firmware itself “updates the device tree to identify the active spare as the boot processor, and performs any other configuration changes . . . to cause the OS to recognize the active spare as the boot processor.” *Id.* at 11 (citing Ex. 1001, 14:4–31, 13:36–66, 17:45–55). According to Patent Owner, this disclosure “suggests that updating the status is a separate step from transferring the role.” *Id.* In sum, Patent Owner’s position is that merely re-assigning the tasks the processor that happened to have booted the system is currently doing is different from re-assigning the specific role of booting the system.

Petitioner replies with two arguments. First, Petitioner asserts that “boot processor responsibilities are not confined to the ‘boot process’” (emphasis omitted); that is, the boot processor does other things, such as shutdown. Pet. Reply 3 (citing, *e.g.*, Ex. 1002 ¶ 46 (Petitioner’s declarant, Dr. Brogioli, asserting that the bootstrap processor “also controls all system

hardware, including [Application Processor] startup and shutdown”)). Second, Petitioner asserts that the “claims require transferring the role of the boot processor to a spare processor not ‘during the boot process,’ . . . but ‘during system runtime.’” *Id.* at 3 (emphasis omitted). In particular, Petitioner asserts that no role transfer needs to persist through a reset. According to Petitioner, “if the transfer requires the spare processor to perform initialization during the boot process after system reboots, the spare processor would only function as a boot processor after the operating system shuts down and restarts,” contrary to the claimed requirement to transferring the role of boot processor *during system runtime*. *Id.* at 4. Petitioner walks through the specification of the ’781 patent and asserts that “nowhere does the ’781 specification describe that transferring the role of the boot processor requires storing information of the new boot processor such that the information is persistently stored ‘for the next time the system reboots.’” *Id.* (quoting PO Resp. 13–14); *see also id.* at 5–8 (discussing specific passages of the ’781 patent).

Based on this interpretation of what it means to assign the role of boot processor, Petitioner reaffirms its position that the combination set out in the Petition (described above) meets the limitations of the claims. *See generally* Pet. Reply. 8–13.

In its Sur-reply, Patent Owner responds to Petitioner’s arguments, and summarizes its position by stating that “the ’781 Patent specification describes a specific approach by which a persistent identifier [is] used” and set during runtime to transfer the role of boot processor, which will be used during the boot process. PO Sur-reply 7; *see also id.* at 5–7 (argument regarding claim construction); *id.* at 7–9 (argument regarding the art).

The parties also argue over whether Petitioner and its declarant can rely on Exhibits 1008 and 1028, which are documents alleging to describe features of Intel processors. PO Resp. 7–8; Pet. Reply 11–12; PO Sur-reply 7–9.

3. *Analysis*

Although the Fox-Safford-Arai combination (with or without Landry) may swap processors that are failing, and perform the necessary bookkeeping to keep the operating system informed as to how to address the processors available, nothing in the Petition persuasively explains whether this generic processor swapping transfers the boot processor role. As we explained in our claim construction section, the term “boot processor” is not merely a label of convenience to describe the processor that happened to have earlier booted the system. The *role* of boot processor is specifically *transferred* to the spare; it is a piece of information tracked by the system. As we explain below, Petitioner does not adequately address the transfer of this information to make the system aware of which processor has the boot processor role.

Looking back to Petitioner’s analysis of “storage” limitation 1[b], Petitioner specifically identifies “the BSP flag of the designated boot processor” of the IA-32 system as teaching that the storage “assigns a role of boot processor.” Pet. 22. Petitioner here is relying on a register used in IA-32 processors to identify whether or not they are the boot processor. This register is used during booting so that only the boot processor boots the system. *See, e.g.*, Ex. 1008, 238–240, 710. This register does not appear to serve any further role, and Petitioner offers no persuasive evidence or argument that the processor register flag is exposed or copied to the

operating system or APIC table. Petitioner’s proposed combination does not involve updating this register, nor does Petitioner explain how, even if the register were updated, it would meet the “transferring” a “role of boot processor” limitation 1[c].

Instead, Petitioner’s discussion around transferring the role of boot processor is limited to identifying processors in the APIC table. *See* Pet. 26–30; Pet. Reply 8–12. Petitioner identifies Fox’s teaching to use interrupts (SMI) “in response to detecting the processor failure to facilitate the processor replacement,” wherein “[t]he SMI identifies to the OS and the BIOS the processor that is failing, and the BIOS, utilizing the SMI handler, takes control of the system and replaces the failing processor with the spare processor.” Pet. 26 (citing Ex. 1005 ¶¶ 15, 32, 41–42, 48). Notably, Fox here is agnostic as to the processor; it does not disclose transferring any flag or identifier specific to identifying a *boot* processor. Petitioner’s discussion regarding Arai is similar. *See* Pet. 28. Arai discusses a table created by BIOS that identifies the processor and its active/inactive state. *Id.* (citing Ex. 1007 ¶¶ 2, 8–9, 19–25). Petitioner does not explain how any of these disclosures mentions any flag or identifier regarding the *boot* processor.

Petitioner similarly argues that “a [person of ordinary skill in the art] would have understood that Fox’s IA-32 system would store the processor identifications and software flags, including those of the boot processor and the spare processor (‘information’), in such register or table accessible to the BIOS and operating system.” Pet. Reply 12 (citing Ex. 1005 ¶¶ 26, 33, 35, 38, 41, 45–46; Ex. 1008, 244, 256, 709–727; Ex. 1028, 38, 69, 72–73, 76; Ex. 1002 ¶¶ 119–120). However, aside from the previously discussed IA-32 register, used by the processors during the booting process to identify

whether they are the boot processor or not (Ex. 1008, 238, 710), Petitioner does not explain how such information is exposed to the BIOS or operating system, nor how it is changed during runtime, after its purpose has been fulfilled.

In sum, even if Petitioner's proposed combination swaps the spare processor for the processor that booted the system, Petitioner's proposed combination does not appear to transfer the *role* of boot processor to the spare processor. At best, Petitioner's proposed combination transfers the tasks of one processor to another. *See* Ex. 1005 ¶¶ 15 (Fox describing its disclosure as "updat[ing] the 'active' processor list" and "[re-]allocat[ing] the threads that were running on the failing processor"), 46 (transferring "processes (instructions, threads, etc.) initially allocated to the failing processor" to the spare processor)). Of the art of record, we are only aware of the '781 patent transferring a designation of which processor is the boot processor during the swap to a spare processor. *See, e.g.,* Ex. 1001, 10:43–55, 11:10–12, 16:19–21, 17:51–55; claims 1, 9, 16, 25. In conclusion, we find that Petitioner has not shown, by a preponderance of the evidence, how the prior art teaches "transferring" a "role of boot processor" as required by limitation 1[c].

F. Remaining Claims and Grounds

The reason why Petitioner's ground for claim 1 fails applies to each remaining claim and ground.

III. CONCLUSION

For the reasons given above, we determine that Petitioner has not established by a preponderance of the evidence that any claim of the '781 patent would have been obvious in view of the prior art cited.

The following table summarizes the outcome of this proceeding:

Claim(s)	35 U.S.C. §	Reference(s)/Basis	Claim(s) Shown Unpatentable	Claim(s) Not Shown Unpatentable
1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, 30	103	Fox, Safford, Arai		1, 2, 5–7, 9– 18, 22, 23, 25, 26, 29, 30
1, 2, 5–7, 9–18, 22, 23, 25, 26, 29, 30	103	Fox, Safford, Landry, Arai		1, 2, 5–7, 9– 18, 22, 23, 25, 26, 29, 30
3, 4, 19– 21, 27, 28	103	Fox, Safford, Arai, Jin		3, 4, 19–21, 27, 28
3, 4, 19– 21, 27, 28	103	Fox, Safford, Landry, Arai, Jin		3, 4, 19–21, 27, 28
8, 24	103	Fox, Safford, Arai, Bigbee		8, 24
8, 24	103	Fox, Safford, Landry, Arai, Bigbee		8, 24
Overall Outcome				1–30

IV. ORDER

In consideration of the foregoing, it is hereby

ORDERED that the record show that Petitioner has not established that any of claims 1–30 of the '781 patent are unpatentable;

FURTHER ORDERED that, because this is a Final Written Decision, parties to the proceeding seeking judicial review of the decision must comply with the notice and service requirements of 37 C.F.R. § 90.2.

IPR2024-01346
Patent 7,627,781 B2

FOR PETITIONER:

Lionel Lavenue
Kara Specht
Cory Bell
Xirui Zhang
Alissa Green
Deanna Smiley
FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, LLP
lionel.lavenue@finnegan.com
kara.specht@finnegan.com
cory.bell@finnegan.com
xirui.zhang@finnegan.com
alissa.green@finnegan.com
deanna.smiley@finnegan.com

FOR PATENT OWNER:

Brett Cooper
Robert Auchter
Seth Lindner
BC LAW GROUP, P.C.
bcooper@bc-lawgroup.com
robert@auchterlaw.com
slindner@bc-lawgroup.com