IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Michaelis *et al.*	§	Attorney Docket No.: 185945.012500
	§	
U.S. Patent No. 7,502,958	§	Customer No.: 67395
	§	
Issue Date: March 10, 2009	§	
	§	
Filing Date: October 25, 2004	§	
	§	
For: SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION	§	

REQUEST FOR *EX PARTE* REEXAMINATION OF U.S. PATENT 7,502,958

Mail Stop "*Ex Parte* Reexam"
Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

Pursuant to the provisions of 35 U.S.C. §§ 301-307, Unified Patents, LLC ("Requester") hereby requests an *ex parte* reexamination of claims 1-25 (the "Challenged Claims") of U.S. Patent 7,502,958 (the "'958 Patent", Ex. 1001), which issued on March 10, 2009, based from U.S. Patent Application 10/973,076 (the "'076 Application"), filed October 25, 2004. The '958 Patent was originally assigned to Hewlett-Packard Development Company, L.P. as recorded in the U.S. Patent and Trademark Office ("USPTO") at reel/frame 037039/0001 and is currently assigned to Foras Technologies LTD. as recorded in the USPTO at reel/frame 058992/0571.

This Request presents prior art references and analyses that are noncumulative of the prior art that was before the Examiner during the original prosecution of the '958 Patent. As demonstrated herein, the Challenged Claims are invalid over these references. Requester therefore requests that an order for reexamination and an Office Action rejecting claims 1-25 be issued.

Ex Parte Patent Reexamination Filing Requirements

Pursuant to 37 C.F.R. § 1.510(b)(1), statements pointing out at least one substantial new question of patentability based on material, non-cumulative reference patents and printed publications for the Challenged Claims of the '958 Patent are provided in Section I of this Request.

Pursuant to 37 C.F.R. § 1.510(b)(2), reexamination of the Challenged Claims (1-25) of the '958 Patent is requested, and a detailed explanation of the pertinence and manner of applying the cited references to the Challenged Claims is provided in Section II of this Request.

Pursuant to 37 C.F.R. § 1.510(b)(3), copies of every patent or printed publication relied upon or referred to in the statement pointing out each substantial new question of patentability or in the detailed explanation of the pertinence and manner of applying the cited references are provided as Exhibits [1004]-[1017] (with Exhibits [1004]-[1007] forming the basis for the SNQs presented herein).

Pursuant to 37 C.F.R. §1.510(b)(4), a copy of the '958 Patent is provided as Exhibit 1001 of this Request, along with a copy of any disclaimer, certificate of correction, and reexamination certificate issued corresponding to the patent.

Pursuant to 37 C.F.R. § 1.510(b)(5), the attached Certificate of Service indicates that a copy of this Request, in its entirety, has been served on Patent Owner at the following correspondence address of record, in accordance with 37 C.F.R. § 1.33(c):

Foras Technologies Ltd.
c/o 56436 - Hewlett Packard Enterprise
3404 E. Harmony Road
Mail Stop 79
Fort Collins, CO 80528

Also submitted herewith is the fee set forth in 37 C.F.R. § 1.20(c)(1).

Pursuant to 37 C.F.R. § 1.510(b)(6), Requester hereby certifies that the statutory estoppel provisions of 35 U.S.C. § 315(e)(1) and 35 U.S.C. § 325(e)(1) do not prohibit Requester from filing this *ex parte* patent reexamination request.

TABLE OF CONTENTS

TABLE OF EXHIBITS

Exhibit	Description
Ex. 1001	U.S. Patent No. 7,502,958 to Michaelis et al. (the "'958 Patent")
Ex. 1002	U.S. File History of the '958 Patent ("U.S. File History")
Ex. 1003	Declaration of Andrew Wolfe ("Wolfe Decl.")
Ex. 1004	U.S. Patent Application Pub. No. 2003/0126498 to Bigbee et al. ("Bigbee")
Ex. 1005	U.S. Patent No. 7,085,959 to Safford ("Safford")
Ex. 1006	U.S. Patent Application Pub. No. 2003/0051190 to Marisetty et al. ("Marisetty")
Ex. 1007	U.S. Patent No. 5,491,788 to Cepulis et al. ("Cepulis")
Ex. 1008	U.S. Patent No. 7,356,733 to Michaelis et al. (the "'958 Idling Reference")
Ex. 1009	U.S. Patent No. 7,818,614 to Michaelis et al. (the "'958 Recognizing Reference")
Ex. 1010	Volume 2: IA-64 System Architecture, *Intel IA-64 Architecture Software Developer's Manual* (1.1 rev. 2000) (the "'958 IA-64 Manual")
Ex. 1011	*Advanced Configuration and Power Interface Specification* (2.0 rev. 2000) (the "'958 ACPI Specification")
Ex. 1012	Intel, *Extensible Firmware Interface Specification*, Version 1.10 (Dec. 1, 2002) ("Intel EFI Specification")
Ex. 1013	Wendy Bartlett, et. al., *Commercial Fault Tolerance: A Tale of Two Systems, IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 1 (January-March 2004) ("Bartlett")
Ex. 1014	U.S. Patent No. 7,191,328 to Hobson ("Hobson")
Ex. 1015	Bozidar Levi, *UNIX Administration, A Comprehensive Sourcebook for Effective Systems and Network Management* (2002) ("Levi")
Ex. 1016	*IA-64 System Abstraction Layer Specification* (2000) ("Intel SAL Specification")
Ex. 1017	*Intel Multiprocessor Specification*, Version 1.4 (May 1997) ("Intel Multiprocessor Specification")
Ex. AA	Appendix A: Claim chart comparing claims to Bigbee, as set forth in Grounds 1 and 2.
Ex. BB	Appendix B: Claim chart comparing claims to Safford in view of Marisetty and Cepulis, as set forth in Grounds 3 and 4.

I. SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

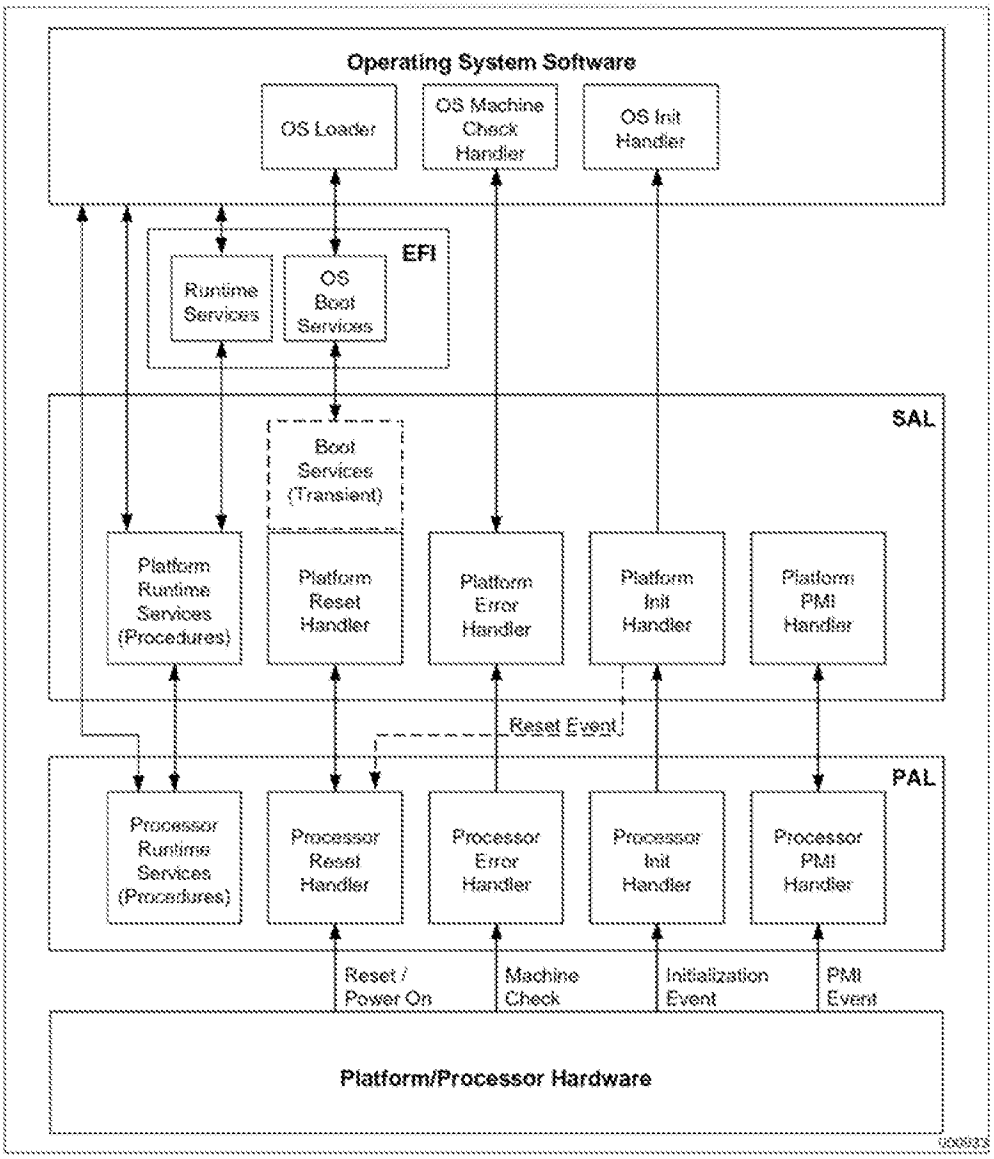
A. U.S. Patent 7,502,958

1. Technical Background

Since the 1960s and 1970s, computer system designers, such as IBM, Tandem, and Stratus, have been developing and releasing fault-tolerant systems with high availability. *Bartlett* (Ex. 1013) at 87 and 89. For example, in 1964, IBM shipped the S/360, which "was designed with robust error checking and was the first commercial computer to include Error Correction Code (ECC) in memory." *Id.* As another example, the Tandem NonStop system shipped in 1976 and included "processor pairs [that] are an effective design for recovering from transient problems." *Id.* at 91. Tandem and IBM both continually improved these systems, and by the early 1990s Tandem had moved to "tight lockstepping of a pair of MIPS chips" with a variant "being developed for future Intel Itanium(R)-based platforms." *Id.* at 91-92. For its part, in 2001 IBM introduced its "Autonomic Computing initiative," which focused on self-healing that included "dynamic CPU sparing, a process by which a failed CPU can be transparently replaced by a spare." *Id.* at 95. These systems were often operating-system centric, with "the operating system provid[ing] its own layer of fault detection and fail-fast response [and supplying] the infrastructure that allows applications to survive single processor failures." *Id.* at 89.

Around the same time that IBM and Tandem were iterating on their high-availability systems in which the OSs managed the hardware, Intel was working on an initiative that attempted to address some of the challenges with OS management of hardware and released the IA-64 System Architecture at the beginning of 2000.[1] *See '958 IA-64 Manual* (Ex. 1010) § 1.7, *see also Intel EFI Specification* (Ex. 1012) at 1-6. The IA-64 System Architecture included a new IA-64 Firmware that included a "Processor Abstraction Layer (PAL)," a "System Abstraction Layer (SAL)," and a "Extensible Firmware Interface (EFI)." *'958 IA-64 Manual* (Ex. 1010) § 1.7, *see also*§ 11.1, fig. 11-2.

[1] Note: The official release of the EFI 1.10 Specification was on December 12, 2002. *Intel EFI Specification* (Ex. 1012) at iii.



Id., fig. 11-2.

"PAL [was a] firmware layer that [abstracted] . . . processor implementation-specific features and [was] independent of the number of processors. SAL [was a] platform specific firmware component that [isolated] OS and other higher level software from implementation differences in the platform. EFI [was a] platform binding specification layer that [provided] a legacy free API interface to the OS Loader." *Intel SAL Specification* (Ex. 1016) § 1.2.

"Itanium-Based Platforms EFI [executed] as an extension to the SAL execution environment with the same rules as laid out by the SAL specification." *Intel EFI Specification* (Ex. 1012) § 2.3.3. The Intel EFI Specification "[required] that functionality defined in a number of other existing specifications be present on a system that [implemented the] specification." *Intel*

EFI Specification (Ex. 1012) at References-5. The Advanced Configuration and Power Interface (ACPI) Specification was one such required specification and at the time was described as "the current state-of-the-art in the platform-to-OS interface" and fully [defining] the methodology that allows the OS to discover and configure all platform resources." *Id.* "The ACPI (Advanced Configuration and Power Interface) specification and the SAL (System Access Layer) specification [were considered] two examples of new and innovative firmware technologies that were designed on the premise that OS developers prefer to minimize runtime calls into firmware." *Id.*

One of the primary goals of the IA-64 System Architecture's PAL firmware was to "provide a consistent firmware interface to abstract processor implementation-specific features." '958 *IA-64 Manual* (Ex. 1010) § 11. Similarly, one of the primary goals of the EFI was "abstraction of the OS from the firmware" to allow the OS "to be constructed with far less knowledge of the platform and firmware that underlie those interfaces." *Intel EFI Specification* (Ex. 1012) at 1-6. In other words, Intel proposed "a backward-compatible solution in which the EFI layer is configured to emulate certain aspects of legacy BIOS operation." *Hobson* (Ex. 1014) at 1:43-46.

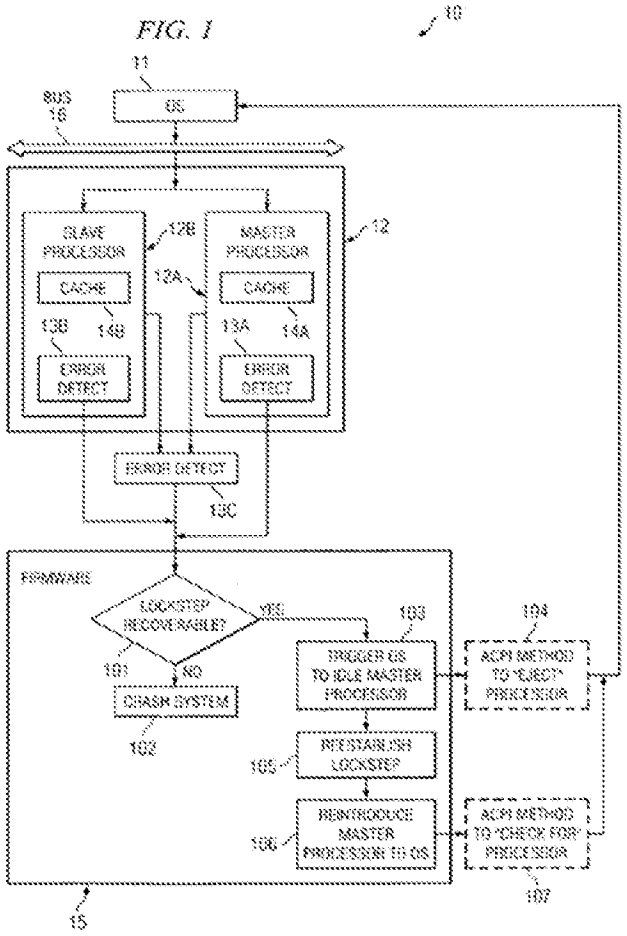
Within this historical context, on October 25, 2004, Applicants for the '958 Patent filed a patent application claiming that they had invented a novel process for handling a "loss of lockstep for a pair of processors" that included (1) "triggering, by firmware, an operating system to idle the processors," (2) "recovering, by the firmware, lockstep between the pair of processors," and "triggering, by the firmware, the operating system to recognize the processors as being available for receiving instructions." *'958 Patent* (Ex. 1001), abstract.

However, as noted above and shown in greater detail below, detecting and recovering from loss-of-lockstep was not new, abstracting and emulating hardware management processes from an OS in a manner that allowed an OS to continue running through recovery was not new either, and pushing operating system and processor implementation-specific features to firmware was commonplace. Furthermore, as explained in detail herein, it would have been obvious to try to combine these limitations in the manner claimed in the '958 Patent. And, as also explained herein, not only was the claimed invention of the '958 Patent obvious, the inventors of the '958 Patent were not the first to attempt to claim this obvious combination. In fact, Intel described this obvious combination to the U.S. Patent Office before the inventors of the '958 Patent. *See*, e.g., Bigbee (Ex. 1004).

2. Summary

The '958 Patent describes systems and methods that use firmware to recover from loss of lockstep "in a manner that allows for recovery from certain detected errors without requiring that [an operating system] be implemented with specific knowledge for handling such recovery." '958 *Patent* (Ex. 1001) at 5:63-66. According to the '958 Patent, "in lockstep processing two processors are paired together, and the two processors perform exactly the same-operations and the results-are compared (e.g., with an XOR gate). If there is ever a discrepancy between the results of the lockstep processors, an error is signaled." *Id.* at 2:16-20. The '958 Patent uses "Loss of lockstep" (or "LOL") broadly to "refer to any error in the pair of lockstep processors." *Id.* at 2:27-28. An exemplary configuration of the '958 Patent's system is illustrated in fig. 1, which is reproduced below for convenience.

FIG. 1



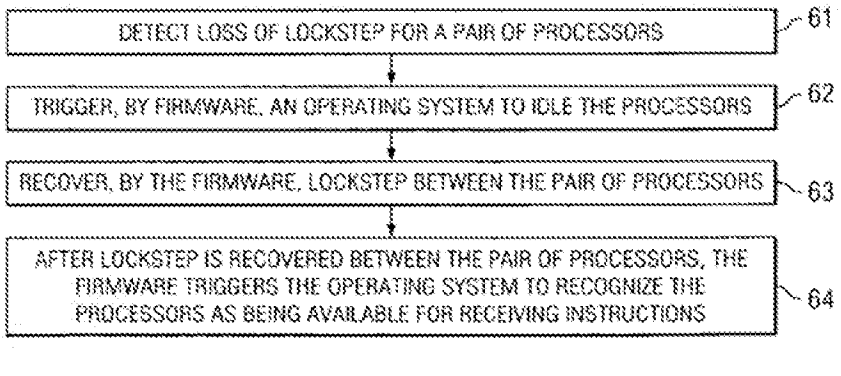
Id., fig. 1.

According to the '958 Patent, "[t]he traditional response to detected LOL has been to crash the system to ensure that the detected error is not propagated through the system." *Id.* at 2:66-3:2.

According to the '958 Patent, "prior solutions attempting to resolve at least some detected [silent data corruptions] without requiring [a] system to be crashed have been Operating System ('OS') centric," which "requires a lot of processor and platform specific knowledge to be embedded in the OS." *Id.* at 3:17-23. However, the '958 Patent states that at least one prior firmware-based solution "[attempts] to recover from a LOL without involving the OS in such [a] recovery procedure." *Id.* at 3:29-30. In the mentioned solution, "upon LOL being detected, firmware is used to save the state of one of the processors in a lockstep pair (the processor that is considered 'good') to memory, and then both processors of the pair are reset and reinitialized." *Id.* at 3:31-35. According to the '958 Patent, this solution "makes the processors unavailable for an amount of time without the OS having any knowledge regarding this unavailability, and if the amount of time required for recovery is too long, the system may crash." *Id.* at 3:36-40.

In an attempt to differentiate from these traditional responses to loss of lockstep, the '958 Patent claims that its method is able to use firmware in a manner that allows for recovery from loss of lockstep "without crashing the system" and "without requiring that [an OS] be implemented with specific knowledge for handling such recovery." *Id.* at 5:56 and 5:63-66. The manner in which the '958 Patent uses firmware to recover from loss of lockstep is shown in the figure below:

FIG. 6



Id. at fig. 6.

The '958 Patent generally focuses on implementation details that are specific to the Advanced Configuration and Power Interface (ACPI) 2.0 Specification and the Itanium Processor Family (IPF), which is described as "a 64-bit processor architecture co-developed by Hewlett-Packard Company and Intel Corporation." *Id.* at 5:5-7, 6:2-4, 6:60-61. The '958 Patent claims that "[t]he quintessential model of the traditional IPF architecture is given in the *Intel IA-64 Architecture Software Developer's Manual, Volume 2: IA-64 System Architecture, in section 11.1*

Firmware Model." *Id.* at 7:24-29. The '958 Patent states that "the IPF processor architecture comprises such supporting firmware as Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI)" and uses the terms "PAL," "SAL," and "EFI" interchangeably with the term "firmware." *Id.* at 5:23-26, 7:29-32.

In addition to claiming general firmware-based systems and methods that use firmware to recover from loss of lockstep, the '958 Patent claims a method in which "a hot spare processor [is utilized] for recovering from LOL for [a] system's boot processor." *Id.* at 4:1-3, *see, e.g.*, claims 23-25.

For the sake of reference, the claims for which reexamination is requested are reproduced below. Claims 1, 13, 19, and 23 are the independent claims, while the remaining challenged claims depend directly or indirectly from these claims.

1. A method comprising:

detecting loss of lockstep for a lockstep pair of processors;

using firmware to trigger an operating system to idle the lockstep pair of processors, recover lockstep for the lockstep pair of processors, and trigger the operating system to recognize the lockstep pair of processors having recovered lockstep; and

responsive to said detecting loss of lockstep, using said firmware to determine if lockstep is recoverable for the lockstep pair of processors, wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.

2. The method of claim 1 wherein said using firmware to trigger an operating system to idle the lockstep pair of processors further comprises:

using the firmware to trigger the operating system to idle a master processor of the lockstep pair of processors.

3. The method of claim 2 wherein said using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep further comprises:

using the firmware to trigger the operating system to recognize the master of the lockstep pair of processors.

4. The method of claim 1 wherein said detecting loss of lockstep comprises:

detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon.

5. The method of claim 1 wherein said detecting loss of lockstep comprises:

detecting a precursor to lockstep mismatch.

6. The method of claim 1 wherein said detecting loss of lockstep comprises:

detecting lockstep mismatch between the lockstep pair of processors.

7. The method of claim 1 further comprising:

if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable.

8. The method of claim 1 further comprising:

if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable.

9. The method of claim 1 further comprising:

determining if said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor.

10. The method of claim 9 wherein if determined that said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor, further comprising:

swapping the role of system boot processor to another processor in the system.

11. The method of claim 10 further comprising:

after said swapping, recovering the lockstep for the lockstep pair of processors.

12. The method of claim 11 further comprising:

after recovering lockstep for the lockstep pair of processors, holding the recovered pair of processors in idle as a spare for the processor to which the role of system boot processor was swapped.

13. A system comprising:

an Advanced Configuration and Power Interface (ACPI)-compatible operating system;

master processor operable to process instructions scheduled by said ACPI-compatible operating system;

slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master processor; and

firmware operable, responsive to detection of loss of lockstep between the master and slave processors, to use an ACPI method to trigger the operating system to idle the master processor, attempt to recover lockstep between the master and slave processors, and if recovery of lockstep is successful then use an ACPI method to reintroduce the master processor to the operating system.

14. The system of claim 13 further comprising:

error detection logic for detecting loss of lockstep between the master and slave processors.

15. The system of claim 13 further comprising:

error detection logic for detecting a precursor to loss of lockstep.

16. The system of claim 15 wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep.

17. The system of claim 15 wherein said error detection logic comprises a parity-based mechanism.

18. The system of claim 13 further comprising error detection logic for detecting lockstep mismatch between the master and slave processors.

19. A system comprising a pair of lockstep processors and computer-executable firmware code stored to a computer-readable medium, the computer-executable firmware code comprising:

firmware code, responsive to detection of loss of lockstep between the pair of lockstep processors, for determining if the lockstep is recoverable for the pair of lockstep processors, wherein said firmware code for determining if the lockstep is recoverable further comprises firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors; and

firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating system to idle the processors, attempting to recover lockstep between the pair of processors, and if lockstep is successfully recovered between the pair of processors, triggering said operating system to recognize the processors as being available for receiving instructions.

20. The system of claim 19 wherein the code for attempting to recover lockstep further comprises:

code for resetting the pair of processors.

21. The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering the operating system to idle the processors comprises:

code for using an ACPI method to idle the processors.

22. The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering said operating system to recognize the processors as being available for receiving instructions comprises:

code for using an ACPI method to cause the operating system to check for the processors.

23. A method comprising:

establishing, in a multi-processor system, a hot spare processor for a system boot processor;

detecting loss of lockstep for a lockstep pair of processors in said multi-processor system;

determining if said lockstep pair of processors for which the loss of lockstep is detected is the system boot processor;

if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor, copying a state of the system boot processor to the hot spare processor; and

if determined that said lockstep pair of processors for which the loss of lockstep is not the system boot processor, then triggering an operating system to a) idle the lockstep pair of processors, b) attempt to recover lockstep between the lockstep pair of processors, and c) if lockstep is successfully recovered between the lockstep pair of processors, trigger said operating system to recognize the processors as being available for receiving instructions.

24. The method of claim 23 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising:

attempting to recover lockstep between the lockstep pair of processors after copying the state of the system boot processor to the hot spare processor.

25. The method of claim 24 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising:

if lockstep is successfully recovered between the lockstep pair of processors, establishing the lockstep pair of processors for which lockstep was recovered as a hot spare processor.

3. Priority

The '958 Patent resulted from the allowance of Application No. 10/973,076, which was filed on October 25, 2004. While the '958 Patent includes cross-references to applications related by subject matter, the '958 Patent does not claim priority to any other applications.

4. Prosecution History

The '958 Patent was allowed after two rounds of prosecution, during which Applicants moved the claims to allowance by amending the independent claims to recite using firmware to determine whether lockstep mismatch had occurred. In particular, independent claims 1 and 19 were amended as described below, and independent claims 13 and 23 were allowed in their original form after the Applicant argued that:

- the cited references did not disclose "triggering an operating system to idle a lockstep pair of processors for which a loss of lockstep has been detected" (*U.S. File History* (Ex. 1002) at 152),
- the "node controller" in Safford (Ex. 1005) is "not an operating system" (*Id.* at 154), and
- Safford "merely disables the processor pair by taking it offline, this may lead to the operating system timing out and crashing" [2] (*Id.* at 154).

After, the Applicant amended the claims to recite:

- using said firmware to determine if lockstep is recoverable for the lockstep pair of processors (*Id.* at 104),
- using said firmware to determine if a lockstep mismatch has occurred between the lockstep pair of processors. *Id.*

During the first round of prosecution, claims 1-35 were rejected under 35 U.S.C. § 103(a) in view of Marshall (U.S. Patent 5,915,082) titled "Error detection and fault isolation for lockstep processor systems" and Safford (U.S. Patent 7,085,959) titled "Method and apparatus for recovery from loss of lock step." *See id.* at 163.

The Applicant's first response did not amend the independent claims and argued against the Examiner's assertion that the art teaches the **signaling of an operating system to idle a lockstep pair of processors** while acknowledging that the art discloses a processor that signals to a node controller that a processor pair should be taken off-line.[3] *See id.* at 136-154. The Applicant's response specifically argued that the node controller in the art is not an operating system since, "in fig. 3 [of Safford,] the node controller 130 is illustrated as an independent block alongside the CPU

[2] However, note that the point of Safford is to enable continued operation of the system during recovery from loss of lockstep. *See, e.g., Safford* (Ex. 1005) at 2:1-9, 3:28-31.

[3] Unless otherwise specified, all **bold** and *italics* emphases and annotations below have been added. Text in *italics* is used to signify claim language and/or add emphasis.

hardware blocks, and does not appear to comprise an operating system running on a CPU."[4] *Id.* The Applicant's response further argued that **taking a processor off-line is different than idling a processor since taking a processor off-line could crash a system**.[5] *Id.* Applicant's response indicated that Applicant's claims solve this crashing issue. *Id.*

In response to the second action, the Applicant amended independent claims 1 and 21 (now claim 19) to include allowable subject matter. *See id.* at 103-112. Specifically, independent claim 1 was amended to include using firmware to **determine if lockstep is recoverable and to determine if lockstep mismatch has occurred**. *Id.* Independent claim 19 was **amended to include using firmware to determine if lockstep mismatch has occurred**. *Id.*

Independent claim 19 was subsequently amended by way of examiner's amendment to include a pair of lockstep processors in order to overcome a potential § 101 issue. *See id.* at 82.

Three notices of allowances were received during prosecution; however, none of the notices provided any reasons for allowance. *See id.* at 12-15, 49-52, 77-89.

B. Level of Ordinary Skill in the Art

A Person of Ordinary Skill in The Art ("POSITA") would have had, as of October 25, 2004: (1) at least a bachelor's degree in electrical engineering, electronics engineering, computer engineering, or a related scientific field; and (2) at least one to two years of experience in the computer science field, including chip design and software engineering, with additional education substituting for less experience and vice versa. *Wolfe Decl.* (Ex. 1003) ¶ 48.

C. Claim Construction

1. Overview

The claims should be construed according to their broadest reasonable interpretation. "During reexamination ordered under 35 U.S.C. 304, and also during reexamination ordered under 35 U.S.C. 257, claims are given the broadest reasonable interpretation consistent with the specification and limitations in the specification are not read into the claims (*In re Yamamoto*, 740 F.2d 1569, 222 USPQ 934 (Fed. Cir. 1984)). Requester reserves the right to advocate a different claim interpretation in any other forum.

[4] However, note that each of the '958 patent's illustrations of an OS is an independent block set alongside blocks representing lockstep-processor pairs.

[5] However, note that the point of Safford is to enable continued operation of the system during recovery from loss of lockstep. *See, e.g.*, *Safford* (Ex. 1005) at 2:1-9, 3:28-31.

2. "boot processor" (Claims 9, 10, 12, and 23-25)

Claims 9, 10, 12, and 23-25 recite a "boot processor," which should be given the broadest reasonable interpretation that is both "consistent with the ordinary and customary meaning of the term" and "consistent with the use of the term in the specification and drawings." *See* MPEP § 2111. Under this standard, a boot processor is a common component that was found in computer systems at the time of the alleged invention of the '958 Patent, and would have been understood by a POSITA to include **any processor designated for handling bootstrap functions**. *See Wolfe Decl.* (Ex. 1003) ¶ 53. Furthermore, a boot processor can also be referred to as a bootstrap processor (BSP) or by a computer-readable designator (e.g., "logical CPU 0"). *Id.*

As noted, a boot processor was a critical device for any computing system and was a common component included in computing systems. *See Wolfe Decl.* (Ex. 1003) ¶ 51. Indeed, "[t]he execution of the bootstrap program is always automatically initiated when the system is powered-on or when a system hardware reset is applied. It is also initiated when the system is rebooted from the system console." *Levi* (Ex. 1015) § 4.2.1. Furthermore, "[t]he origin of the word boot (as in, 'to boot the system') is bootstrapping, which is the process of bringing a computer system to life and ready to use. ('Bootstrapping' is actually the 'nerd word' for starting up a computer.)." *Id.*

As of the priority date, a POSITA would have understood that in multi-processor system architectures, a boot processor may either be selected by default or through a selection process during bootup. For example, as noted in the '958 IA-64 Manual, which the '958 Patent incorporates by reference, "firmware selects a Bootstrap processor (BSP) in multi-processor (MP) configurations early in the boot sequence." *'958 IA-64 Manual* (Ex. 1010) § 24.1.1. And in some systems (e.g., fault-tolerant systems), the processor designated as the boot processor may change over time. For example, the Intel MultiProcessor Specification explains how an operating system can determine and track the ID of a designated BSP:

> While all processors in an MP-compliant system are functionally identical, one of the processors will be designated as the bootstrap processor (BSP) at system initialization by the system hardware or by the system hardware in conjunction with the BIOS. The rest of the processors are designated as the application processors (APs). The BSP is responsible for booting the operating system. Once the MP operating system is up and running, the BSP functions as an AP. Usually a processor is designated as the BSP because it is capable of controlling all system hardware, including AP startup and shutdown. The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the

> BSP operating as the last running processor during system shutdown. The BSP is not necessarily the first processor, especially in fault-tolerant MP systems in which any available processor can be designated as the BSP.

Intel Multiprocessor Specification (Ex. 1017) § B.1.

In some multi-processor system architectures, a boot processor may be referred to by a logical, rather than a physical, position (e.g., logical CPU 0 or CPU L0 rather than physical CPU 0 or CPU P0) that is indicative of its designation for controlling other system hardware since the role of boot processor may often be assigned to a processor regardless of its physical position within a system. For example, during boot-processor selection,

> If CPU P0 is not functioning properly and cannot perform even the most basic functions, it is designated as inoperative. The hardware 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. 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. The process repeats until an operational microprocessor is found to perform the CPU L0 functions.

Cepulis (Ex. 1007) at 2:52-64.

The proposed interpretation of "boot processor" is consistent with the specification of the '958 Patent, which references the boot process of an Itanium Processor Family (IPF) system:

> The boot-up process of a traditional IPF system, for example, proceeds as follows: when the system is first powered on, there are some sanity checks (e.g., power on self-test) that are performed by microprocessors included in the system platform, which are not the main system processors that run applications. After those checks have passed, power and clocks are given to a boot processor (which may, for example, be master processor 12A).

'958 Patent (Ex. 1001) at 7:40-47.

As shown, the broadest reasonable interpretation for boot processor that is both "consistent with the ordinary and customary meaning of the term" and "consistent with the use of the term in the specification and drawings" is any processor designated for handling bootstrap functions including any processor that is referred to as a boot processor, that is referred to as a bootstrap processor (BSP), or that is defined by its logical designation as a boot processor (e.g., logical CPU 0 or CPU L0).

D. List of Prior Art Patents and Printed Publication

Reexamination of the Challenged Claims is requested in view of the following references:

Exhibit 1004 (U.S. Patent Application Publication No. 2003/0126498 to Bigbee et al. ("Bigbee")): Bigbee is a prior art printed publication because it was filed as an application on January 2, 2002, and published on July 3, 2003, which is before the earliest claimed effective filing date of October 25, 2004.

Bigbee is prior art at least under pre-AIA 35 U.S.C. § 102(a) and pre-AIA 35 U.S.C. § 102(b).

Exhibit 1005 (U.S. Patent No. 7,085,959 to Safford ("Safford")): Safford is a prior art patent because it was filed as an application on July 3, 2002, and published as U.S. Patent Publication No. 2004/0006722 on July 8, 2004, which is before the earliest claimed effective filing date of October 25, 2004. Safford issued on August 1, 2006.

Safford is prior art at least under pre-AIA 35 U.S.C. § 102(a).

Exhibit 1006 (U.S. Patent Application Publication No. 2003/0051190 to Marisetty et al. ("Marisetty")): Marisetty is a prior art printed publication because it was filed as an application on September 27, 1999, and published on March 13, 2003, which is before the earliest claimed effective filing date of October 25, 2004.

Marisetty is prior art at least under pre-AIA 35 U.S.C. § 102(a) and pre-AIA 35 U.S.C. § 102(b).

Exhibit 1007 (U.S. Patent No. 5,491,788 to Cepulis et al. ("Cepulis")): Cepulis is a prior art patent because it was filed as an application on September 10, 1993, and issued on February 13, 1996, which is before the earliest claimed effective filing date of October 25, 2004.

Cepulis is prior art at least under pre-AIA 35 U.S.C. § 102(a) and pre-AIA 35 U.S.C. § 102(b).

As shown below, Requester submits that these prior art references raise new "substantial [questions] of patentability" because "the [teachings] of the (prior art) patents and printed publications [are] such that a reasonable examiner would consider the [teachings] to be important in deciding whether or not the [claims are] patentable." *See* MPEP § 2242. Because these patents or printed publications present substantial new questions of patentability, reexamination should be ordered and the challenged claims should be canceled.

E. Discretionary Denial Is Not Warranted

There is no reason to deny reexamination based on any discretionary factor. 35 U.S.C. § 325(d) states that:

> Notwithstanding sections 135(a), 251, and 252, and chapter 30, during the pendency of any post-grant review under this chapter, if another proceeding or matter involving the patent is before the Office, the Director may determine the manner in which the post-grant review or other proceeding or matter may proceed, including providing for the stay, transfer, consolidation, or termination of any such matter or proceeding. In determining whether to institute or order a proceeding under this chapter, chapter 30, or chapter 31, the Director may take into account whether, and reject the petition or request because, the same or substantially the same prior art or arguments previously were presented to the Office.

Of the references listed above, Bigbee, Safford, and Marisetty were cited during prosecution of the '958 Patent. U.S. Patent No. 6,920,581, which issued from Bigbee, was also cited during prosecution of the '958 Patent. However, out of these cited references, only Safford was applied in a substantive rejection of the claims of the '958 Patent.

Each of Bigbee, Safford, and Marisetty is being presented herein "in a new light, or in a different way, as compared with its use in the earlier examination(s), in view of a material new argument or interpretation presented in [the instant] request." *See* MPEP § 2242. Thus, while Bigbee, Safford, and Marisetty have been previously cited, the instant request presents new combinations, arguments, and basis for invalidity not previously considered by the Office. Thus, consideration of these references in the instant request is proper. *See id.*

Each of Bigbee, Safford, Marisetty, and Cepulis is relied upon herein in proposed rejections and present new, non-cumulative technological teachings that were not previously considered and discussed on the record during prosecution of the application that resulted in the '958 Patent or during prosecution of any other prior proceeding involving the '958 Patent. *See* MPEP § 2216. The instant request presents Bigbee, by itself, and Safford in combination with Marisetty to disclose, teach, or suggest each of the limitations of the Challenged Claims. The instant request presents new, non-cumulative technological teachings of Cepulis to place Safford in "a new light or a different way that escaped review during earlier examination." *See* MPEP § 2242.

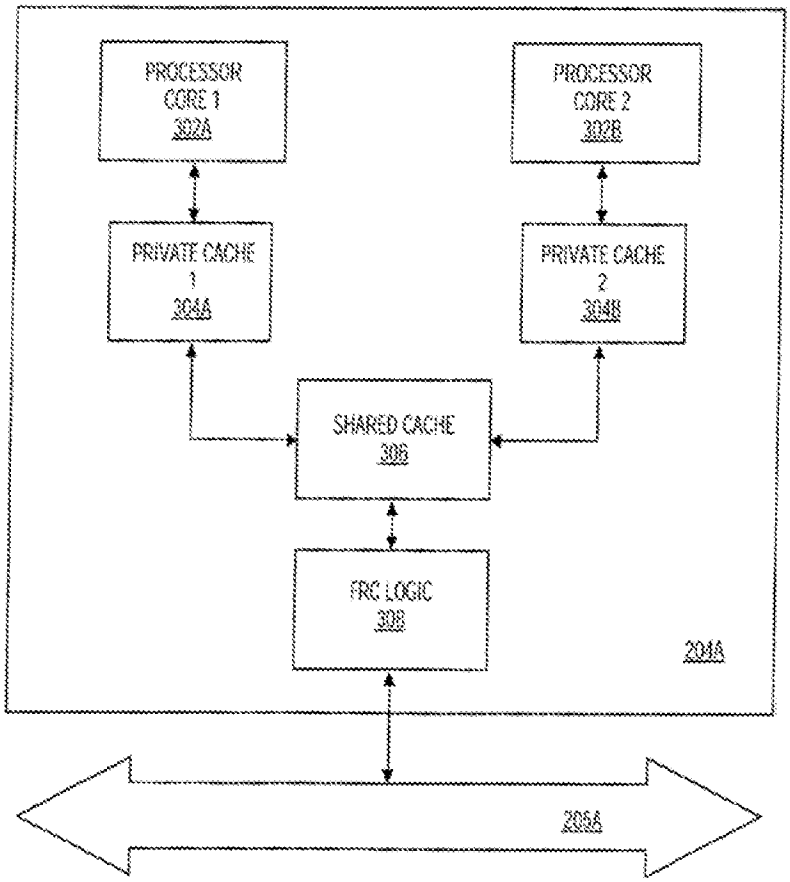
Requester submits that, because substantial new questions with respect to all cited references are presented herein that have not been previously presented to or adjudicated by the U.S. Patent Office, 35 U.S.C. § 325(d) does not warrant denial of this Reexam Request.

F. Ground 1: Bigbee Presents Substantial New Questions of Patentability For Claims 1-8 and 13-22 under 35 U.S.C. § 102

1. Overview of Bigbee

Bigbee generally describes systems and methods for firmware-based recovery from loss of lockstep in the same way as the '958 Patent, as demonstrated below in § II.B. *See Bigbee* (Ex. 1004), claim 28 and ¶ [0002]. Bigbee improves upon prior operating-system-based recovery methods that "[burden] the operating system" by requiring processor and platform specific knowledge to recover from errors related to loss of lockstep. *See id.* ¶ [0004]. Bigbee improves upon prior firmware-based recovery methods that don't account for "operating systems [having] a maximum tolerable interrupt latency before system errors or failures occur." *Id.* ¶ [0003]. Furthermore, the systems and methods of Bigbee use firmware to recover from loss of lockstep in a manner that isolates "operating [systems] from implementation-specific platform differences." *See id.* ¶ [0018].

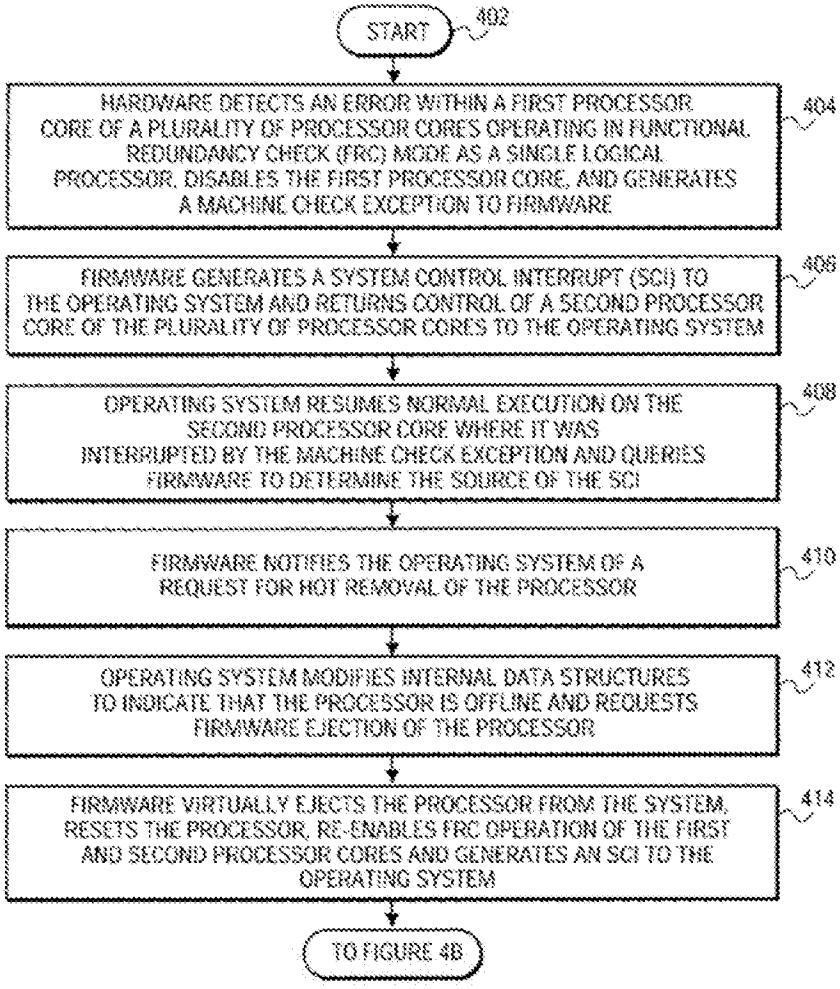
Bigbee's data processing systems include "two or more physical processor cores operating **lock step** in FRC mode as a single logical processor from the point of view of [an] operating system." *Id.* ¶ [0014]; *see also*, fig. 3.



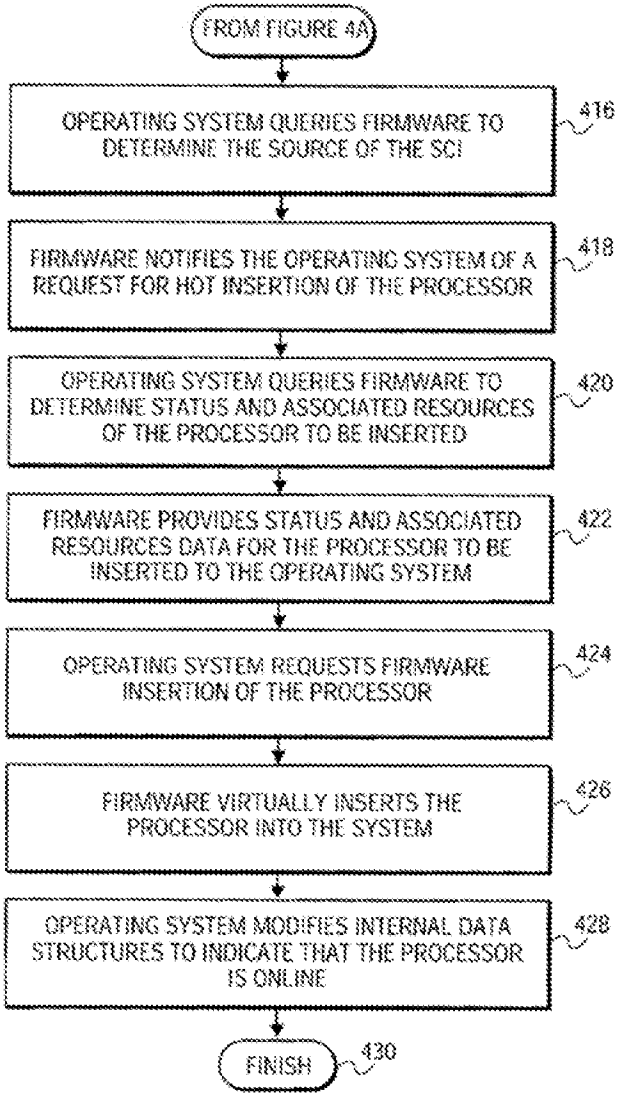
Id., fig. 3.

Bigbee describes a pair of processor cores that "are operated in a . . . 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.* ¶ [0002]. Bigbee describes using firmware to "[generate] a signal such as an interrupt or exception once an inconsistent result has been detected." *See id.* ¶ [0002] and [0034], *see also* claim 28.

Bigbee describes using firmware to communicate with an operating system during recovery from loss of lockstep in a manner that enables the operating system to survive the loss of lockstep without crashing. *See id.*, FIGS. 4A and 4B. For example, Bigbee describes using firmware to (1) trigger an operating system to idle a processor that has lost lockstep (steps 406 and 410) and (2) recover the processor from loss of lockstep (step 412). *See id.*, fig. 4A.



Id., fig. 4A. Bigbee further describes using firmware to trigger an operating system to recognize a recovered lockstep processor (e.g., steps 418-428). *See id.*, fig. 4B.



Id., fig. 4B.

Bigbee generally focuses on implementation details associated with "advanced configuration and power interface (ACPI)-compliant data processing [systems]" and "the Itanium™ family of processors." *Id.* ¶¶ [0016] and [0033]. Additionally, Bigbee defines "firmware" to include "an extensible firmware interface (EFI), a system abstraction level (SAL), and a processor abstraction level (PAL)." *Id.* ¶ [0018].

Bigbee is analogous art to the '958 Patent. *See Wolfe Decl.* (Ex. 1003) ¶ 61. For example, Bigbee is from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) and is reasonably pertinent to the problem faced by the inventors of the '958 Patent. See Bigbee (Ex. 1004) ¶¶ [0002]-[0004] and [0018], *see also Wolfe Decl.* (Ex. 1003) ¶ 61. As noted above, the '958 Patent is directed to solving at least two problems: (1) using

firmware to recover from loss of lockstep in a manner that avoids crashing an operating system and (2) recovering from loss of lockstep in a manner that does not require an operating system to have platform-specific recovery knowledge. *'958 Patent* (Ex. 1001) at 3:22-28. In particular, the '958 Patent notes that the problem with an "OS-centric type of solution" is that it "requires a lot of processor and platform specific knowledge to be embedded in the OS," which creates "such a large burden that most commonly used OSs do not support lockstep recovery." *Id.* Similarly, Bigbee's proposed systems are directed to addressing "significant drawbacks" associated with operating-system based recovery, including that "it burdens the operating system, limits the number and types of operating systems which may be utilized with a particular hardware platform, limits the ability of the platform to evolve, and prevents differentiation in terms of the underlying platform implementation." *Bigbee* (Ex. 1004) ¶ [0004], *see also Wolfe Decl.* (Ex. 1003) ¶ 61.

Bigbee anticipates each and every element of claims 1-8 and 13-22. *See Wolfe Decl.* (Ex. 1003) ¶¶ 75-155. Therefore, Bigbee presents a substantial new question of patentability. A reasonable examiner would find the teachings of this reference important in determining whether claims 1-8 and 13-22 are patentable because Bigbee was never discussed or otherwise critically considered during prosecution. Appendix A shows in greater detail how Bigbee discloses each and every element of claims 1-8 and 13-22.

G. Ground 2: Bigbee Presents Substantial New Questions of Patentability For Claims 1-8 and 13-22 under 35 U.S.C. § 103

As explained above, Bigbee anticipates each and every element of claims 1-8 and 13-22. To the extent Patent Owner argues that Bigbee does not expressly disclose each and every element, Requester submits that any differences between Bigbee and the claims of the '958 Patent would have been minor and obvious in view of the knowledge of a person having ordinary skill in the art at the time of the earliest priority date of the '958 Patent. *See Wolfe Decl.* (Ex. 1003) ¶¶ 75-155.

H. Ground 3: Safford in view of Marisetty Presents Substantial New Questions of Patentability for Claims 1-8 and 13-22

1. Overview of Safford

Safford generally describes a multi-processor system, like the one disclosed in the '958 Patent as demonstrated below in §§ II.D and II.E, that is capable of recovering from a loss of lockstep detected in connection with a certain processor pair. *Safford* (Ex. 1005), Abstract. As explained in Safford, "to improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a master/check pair." *Id.* at 2:55-57. "Each of two

processors in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit located near external interfaces of the two processors." *Id.* at 2:57-60. "Any difference in the processor outputs indicates the existence of an error." *Id.* at 2:60-63. The configuration of Safford's system, whose processors operate in lockstep, is illustrated in fig. 2 and is reproduced below for convenience.

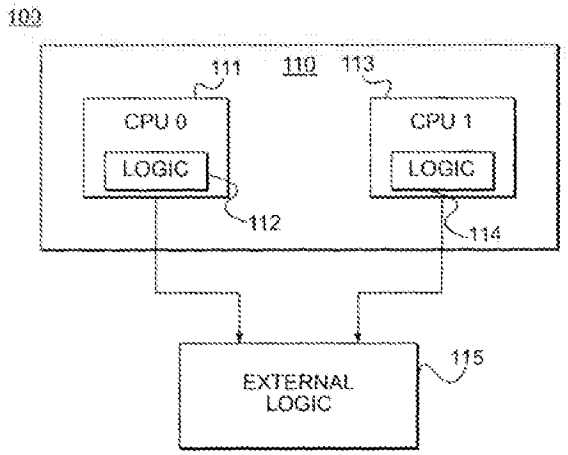


FIG. 2

Id., fig. 2

To recover from loss of lockstep, Safford's system follows the process illustrated in fig. 4, which is reproduced below for convenience.

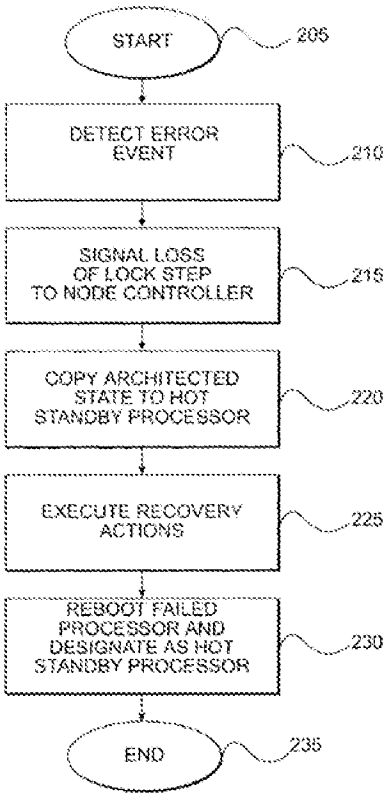


FIG. 4

Id., fig. 4

Safford describes detecting a loss of lockstep by (1) detecting a lockstep mismatch (e.g., a "difference" in the outputs of a lockstep pair of processors) or (2) detecting precursors to loss of lockstep that "will guarantee that [processors] will break lock step at some future time." *See id.* at 3:37-45, 1:53-55 ("[a] difference in processing . . . is by definition a loss of lock step"). Contrary to Applicant's statements made during prosecution, Safford describes recovering from a loss of lockstep in a manner that enables a multiprocessor system to survive the loss of lockstep without crashing. *See id.* at 3:27-30, 3:46-48.

Safford describes "detecting a loss of lock step initiating event in a processor" or "an impending loss of lock step." and responding by "idling the loss of lock step processor." *Id.*, claim 13.

Safford describes maintaining an idle lockstep processor pair (e.g., "designated as a 'hot standby'") that "may be used to speed recovery from [a] loss of lock step" *Id.* at 4:25-30.

Safford describes a system that includes logic for reassigning the role of boot processor in response to a loss of lockstep by the boot processor. For example, Safford's system recovers from

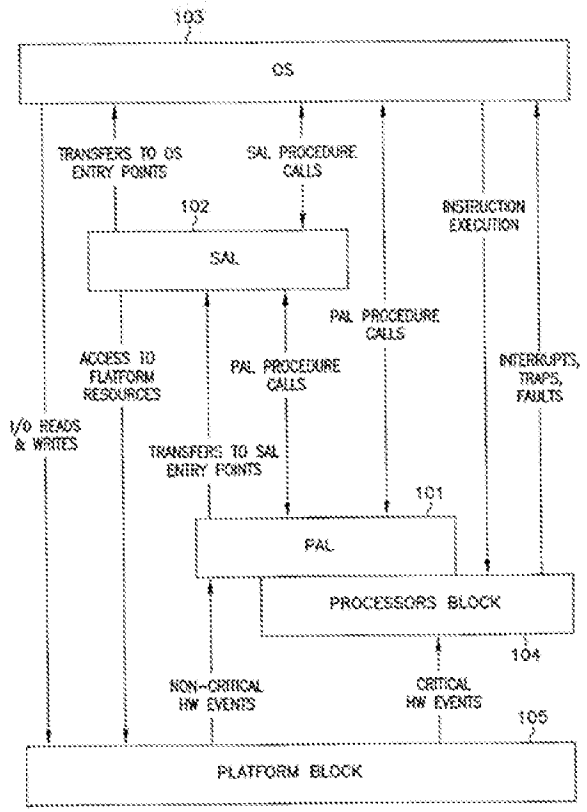
the loss of lockstep by transferring the boot-processor role of the failed processor pair to a hot standby processor pair. For example, Safford's "processor 111 signals the node controller 130 that the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken 'off-line.'" *Id.* at 4:37-40. Then Safford's "node controller 130 copies the architected state of the first processor pair 111/113 to the hot standby processor pair 125/127". *Id.* at 4:40-43. After the architected state is copied to the hot standby processor pair 125/127, "[t]he processor pair 125/127 . . . becomes the logical CPU 0 in the computer system 100." *Id.* at 4:52-54.

Safford is analogous art to the '958 Patent because Safford is both from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) as the '958 Patent and is reasonably pertinent to the problem faced by the inventors of the '958 Patent (e.g., system crashes caused by loss of lockstep). *See Wolfe Decl.* (Ex. 1003) ¶ 66. For example, the '958 Patent, in focusing on "the increased desire for many systems to maintain high availability" notes the problem that "crashing the system each time LOL is detected is not an attractive proposition." *'958 Patent* (Ex. 1001), 3:5-7. Similarly, Safford in focusing on "improv[ing] availability of the processor assets of the computer system," notes that "loss of lock step, if not promptly corrected, may cause the computer system 18 to crash." *Safford* (Ex. 1005), 3:21-22, 1:45-47.

2. Overview of Marisetty

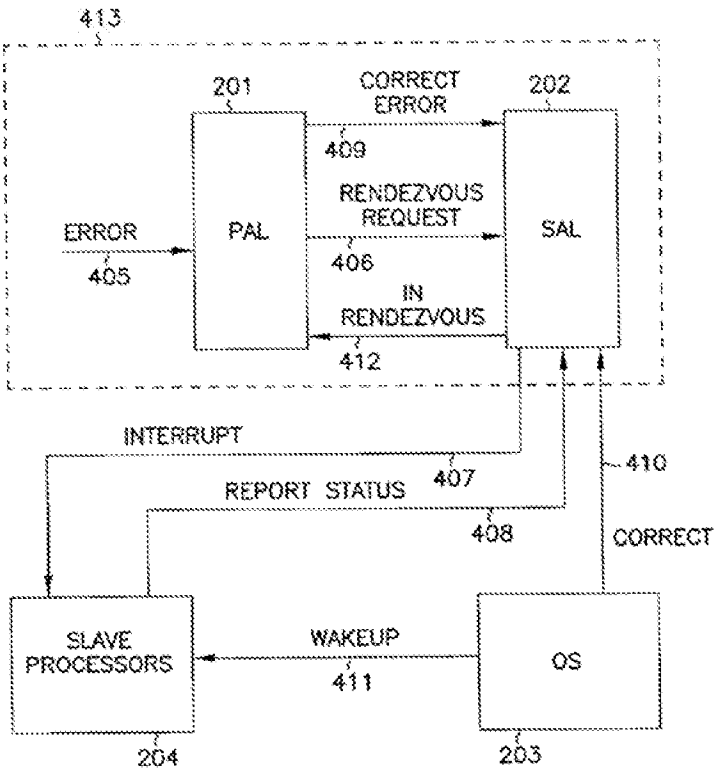
Marisetty generally describes firmware-based "systems and methods for error handling on multiprocessor systems" in the manner disclosed in the '958 Patent, as demonstrated below in §§ II.D and II.E. *Marisetty* (Ex. 1006) ¶ [0014]. These firmware-based systems and methods include an error-handling routine that utilizes a "rendezvous" of processors (e.g., a state in which processors "enter an idle state" while processors errors are corrected). *See id.*, abstract, ¶ [0030].

Marisetty defines "firmware" to include a "system abstraction layer (SAL)" and a "processor abstraction layer (PAL)." *See id.* at ¶¶ [0024] and [0026], *see also* fig. 1.



Id., fig. 1.

Marisetty describes using firmware to communicate with an operating system during recovery from a processor error in a manner that enables the operating system to survive the processor error without crashing. *See id.* fig. 4 and ¶¶ [0042] and [0058]. For example, Marisetty indicates that "**SAL 102 can inform the OS 103 layer of the request for rendezvous state,**" and "[t]he OS 103 can then inform SAL 102 that it is ready to enter rendezvous state and **tells SAL 102 to enter the rendezvous state**." *Id.* at ¶ [0042]. Marisetty further states that, "[o]nce the error has been corrected, the OS is informed that the error has been corrected," "[t]he OS can inform all processors to get out of rendezvous state," and "the system resumes normal operation." *Id.* at ¶ [0042].



Id., fig. 4.

Marisetty describes using firmware to recover from processor errors in a manner that abstracts low-level recovery details from higher-level layers such as operating systems. *See id.* ¶¶ [0004] and [0008].

Marisetty is analogous art to the '958 Patent. *See Wolfe Decl.* (Ex. 1003) ¶ 71. For example, Marisetty is from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) and is reasonably pertinent to the problem faced by the inventors of the '958 Patent. *See* Marisetty (Ex. 1006) ¶¶ [0001], [0002], and [0014], *see also Wolfe Decl.* (Ex. 1003) ¶ 71. As noted above, the '958 Patent is directed to recovering from processor errors in multiprocessor systems in a manner that avoids crashing an operating system or requiring an operating system to have platform-specific recovery knowledge). *'958 Patent* (Ex. 1001) at 3:22-28. Similarly, Marisetty discusses a need for processor-error recovery techniques that do not require multiprocessor systems "to reboot or shutdown for errors because of a lack of proper error handling." *Marisetty* (Ex. 1006) ¶ [0012].

3. Motivation to Combine Safford and Marisetty

While Safford arguably discloses, or at least renders obvious, each limitation of the challenged claims, Safford does not explicitly mention using "firmware" to perform the functions

recited in the Challenged Claims. However, when combined with the teachings of Marisetty, Safford renders obvious a firmware-based process for handling a loss of lockstep for a pair of processors that uses firmware to trigger an operating system to idle the processors, recover lockstep between the pair of processors, and then trigger the operating system to recognize the processors as being available for receiving instructions. This combination of Marisetty and Safford, which are both directed to a similar problem that applicants for the '958 Patent were attempting to solve, would have been obvious for at least the reasons that a POSITA would have arrived at it by simply combining prior art elements (e.g., the lockstep recovery methods of Safford with the IA-64 firmware-based recovery techniques of Marisetty) according to known methods (e.g., the known implementation methods for applying the IA-64 firmware methods of Marisetty, which could be found in the thousands of pages of known specification documents incorporated by reference in the '958 Patent) or by simply applying a known technique (e.g., the firmware-based recovery techniques of Marisetty) to a known device ready for improvement (e.g., the multi-processor systems of Safford). *See* MPEP §§ 2143(A) and 2143(D), *see also Wolfe Decl.* (Ex. 1003) ¶ 157. A POSITA would have had a reasonable expectation that a combination of Marisetty and Safford would succeed for at least the reason that combinations involving the firmware-based recovery techniques of Marisetty were a common and successful practice prior to the priority date of the '958 Patent. *See Wolfe Decl.* (Ex. 1003) ¶ 158.

Prior to the priority date of the '958 Patent, a POSITA would have been motivated to combine the teachings of Safford with those of Marisetty because integration of the error-recovery methods from Safford into a firmware abstraction layer as suggested by Marisetty would lead to more efficient and robust system operation. *See Marisetty* (Ex. 1006) ¶¶ [0008]-[0013], *see also Wolfe Decl.* (Ex. 1003) ¶ 159. Moreover, by localizing and managing errors at lower abstraction levels, demand for higher-level software interventions (such as interventions by operating systems) would be reduced, thus enhancing overall system stability and performance. *See Marisetty* (Ex. 1006) ¶¶ [0008]-[0013], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 157-160, 163.

As explained above, Safford and Marisetty are both directed to a similar problem as the '958 Patent, which is evident in view of the teachings of the '958 Patent and the historical context in which the '958 Patent was filed. In the '076 Application, Applicants claimed that "[t]he traditional response to detected LOL has been to crash [a] system to ensure that [a] detected error is not propagated through the system." *'958 Patent* (Ex. 1001) at 2:66-3:2. According to

Applicants, "prior solutions attempting to resolve" detected LOL "without requiring [a] system to be crashed have been Operating System ('OS') centric," which "requires a lot of processor and platform specific knowledge to be embedded in the OS." *Id.* at 3:17-23. However, Applicants noted at least one prior firmware-based solution that "[recovers] from a LOL without involving the OS." *Id.* at 3:29-30. According to Applicants, the downside of this firmware-based solution was that it did not give an OS "knowledge regarding [processor] unavailability, and if the amount of time required for recovery [was] too long, the system may crash." *Id.* at 3:36-40.

Applicants claim that they invented a novel firmware-based process for handling a "loss of lockstep for a pair of processors" that allowed for recovery from loss of lockstep "without crashing the system" and "without requiring that [an OS] be implemented with specific knowledge for handling such recovery." *Id.* at abstract, 5:56, 5:63-66. Their claimed process included (1) "triggering, by firmware, an operating system to idle the processors," (2) "recovering, by the firmware, lockstep between the pair of processors," and "triggering, by the firmware, the operating system to recognize the processors as being available for receiving instructions." *Id.*, abstract.

However, prior to the priority date of the '958 Patent, Safford described a multi-processor system capable of recovering from a loss of lockstep detected in connection with a certain processor pair, without crashing the system. *Safford* (Ex. 1005), abstract. Safford's system also included logic for transferring the role of a boot processor in response to a loss of lockstep. *Id.* at 3:27-30.

Prior to the priority date of the '958 Patent, Marisetty described a firmware-based error-handling routine for a multiprocessor system that utilizes a rendezvous of processors (e.g., a state in which a single processor handles a processor error while other processors are idled). *See Marisetty* (Ex. 1006), abstract and ¶ [0030]. Marisetty's firmware-based error-handling routine "permits software to be developed for a system without regard to the hardware making up the system, including the number of processors in that system." *Id.* ¶ [0008]. Marisetty's firmware-based error-handling routine handled "four categories [of errors] ranging from least severe to most severe" that "[covered] all errors that might be encountered in a multiprocessor system." *Id.* ¶¶ [0032]-[0035].

As mentioned above, a variety of rationales show why Safford, combined with Marisetty, renders obvious a firmware-based process for handling a loss of lockstep for a pair of processors that uses firmware to trigger an operating system to idle the processors, recover lockstep between

the pair of processors, and then trigger the operating system to recognize the processors as being available for receiving instructions. Marisetty's firmware-based error-handling routine was already well-known, having been published in the '958 IA-64 Manual approximately four years before the priority date of the '958 Patent. *See Wolfe Decl.* (Ex. 1003) ¶ 162. A POSITA reading Safford would have been motivated to incorporate Marisetty's teachings related to placing error handling functionality in firmware because a POSITA reading Safford would have seen the value in Marisetty's process of rendezvousing processors in a multi-processor system during error recovery to avoid operating system crashes as well as Marisetty's ability of abstracting configuration details of hardware from software. *See Wolfe Decl.* (Ex. 1003) ¶ 163. Implementing the lockstep recovery methods of Safford's multi-processor system in firmware would have simply been a matter of applying a known technique (e.g., implementing error handling in a firmware abstraction layer) to a known device ready for improvement (e.g., a multi-processor system) to yield predictable results (e.g., operating systems that avoid crashing without needing processor and platform specific knowledge). *See* MPEP § 2143(D), *see also Wolfe Decl.* (Ex. 1003) ¶ 164. Further, a POSITA would have had a reasonable expectation of success in making the proposed application, at least because both Safford and Marisetty disclose similar systems (multi-processor systems) with similar goals (maintaining high availability) and the background knowledge to apply the teaching of Marisetty to the systems of Safford was already well-known, as demonstrated herein. *See Wolfe Decl.* (Ex. 1003) ¶ 165. Further, the lockstep recovery methods of Safford's multi-processor system fall into the categories of errors expressly handled by Marisetty's solutions, thus implementing the lockstep recovery methods of Safford's multi-processor system in Marisetty's firmware would have yielded predictable results. *Id.*

The combination of Safford and Marisetty renders obvious each and every element of claims 1-8 and 13-22, and therefore presents a substantial new question of patentability. A reasonable examiner would consider the combined teachings of these references important in determining whether claims 1-8 and 13-22 are patentable because neither Marisetty, nor the combination of Safford and Marisetty, was expressly considered or discussed during prosecution. Exhibit B shows in greater detail how the combination of Safford and Marisetty renders obvious each and every element of claims 1-8 and 13-22.

I. Ground 4: Safford as Evidenced by Cepulis and in Combination With Marisetty Presents Substantial New Questions of Patentability for Claims 9, 10, 11, 12, and 23-25

1. Overview of Safford

A discussion of Safford appears above. *Supra* § H.1. That discussion is equally applicable to SNQ3.

Notably, Safford describes a system that includes logic for reassigning the role of boot processor in response to a loss of lockstep by the boot processor. Safford's system recovers from the loss of lockstep by transferring the boot-processor role of the failed processor pair to a hot standby processor pair. For example, Safford describes a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a **logical CPU 0**." *Safford* (Ex. 1005) at 3:4-5. Safford's "processor 111 signals the node controller 130 that the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken 'off-line.'" *Id.* at 4:37-40. Then Safford's "node controller 130 copies the architected state of the first processor pair 111/113 to the hot standby processor pair 125/127". *Id.* at 4:40-43. After the architected state is copied to the hot standby processor pair 125/127, "[t]he processor pair 125/127 . . . becomes the **logical CPU 0** in the computer system 100." *Id.* at 4:52-54.

2. Overview of Marisetty

A discussion of Marisetty appears above. *Supra* § H.2. That discussion is equally applicable to SNQ3 and will not be repeated here for the sake of brevity.

3. Overview of Cepulis

Cepulis generally describes "a multiprocessor computer system [that] handles the failure of one or more of its processors without totally disabling the system." *Cepulis* (Ex. 1007), abstract. Cepulis describes a boot process for a multiprocessor computer system that includes designating one of the multiprocessor computer system's physical processors as a boot processor or "**first logical CPU**." *Id.* Like the boot processor described in the 958 Patent, the boot processor in Cepulis "retains its designation until power is cycled" unless it fails at which time its "designation is transferred to another active CPU." *Id.*

Like Safford, Cepulis uses the "**logical CPU 0**" designator to identify its boot processor as explained in greater detail in § II.E.

Cepulis is analogous art to the '958 Patent. For example, Cepulis is from the same field of endeavor (e.g., providing recovery in a high-availability multi-processor system) and is reasonably

pertinent to the problem faced by the inventors of the '958 Patent. See *Cepulis* (Ex. 1007) at 1:13-17, 1:53-2:26. As noted above, the '958 Patent is directed to recovering from processor errors in multiprocessor systems in a manner that avoids crashing an operating system. *'958 Patent* (Ex. 1001) at 3:22-28. Similarly, Cepulis is directed to "a multiprocessor computer system [that] handles the failure of one or more of its processors without totally disabling the system." *Cepulis* (Ex. 1007), Abstract.

4. Motivation to Combine Safford and Marisetty in light of Cepulis

A discussion of motivations to combine Safford and Marisetty appears above. *Supra* § H.3. That discussion is equally applicable to SNQ3 and will not be repeated here for the sake of brevity. Cepulis is relied on herein for further explanation of the term "**logical CPU 0**" found in Safford and for its detailed disclosure of well-known boot-processor functionality in existence prior to the priority date of the '958 Patent. The combination of Safford and Marisetty in light of Cepulis' explanation of the meaning of the "logical CPU 0" boot-processor designator renders obvious, each and every element of claims 9, 10, 11, 12, and 23-25, and therefore presents a substantial new question of patentability. A POSITA would have considered the combined teachings and explanations of these references important in determining whether claims 9, 10, 11, 12, and 23-25 are patentable because neither Safford, nor the combination of Safford and Marisetty, in light of Cepulis' explanation of the meaning of the "logical CPU 0" boot-processor designator was expressly considered or discussed during prosecution. Exhibit B shows in greater detail how the combination of Safford and Marisetty in light of Cepulis' explanation renders obvious each and every element of claims 9, 10, 11, 12, and 23-25.

II. DETAILED APPLICATION OF THE PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED

A. Proposed Rejections of the Claims

The proposed rejections below are based on anticipation under 35 U.S.C. § 102 or obviousness under 35 U.S.C. § 103(a). Requester notes that, even if there are minor differences between cited disclosures of the prior art and the claimed concepts, the claims are still obvious, as a POSITA would have understood that the claims of the '958 Patent do not offer any new synergy or unexpected results over the disclosures of the prior art and the general knowledge of a POSITA. *See Wolfe Decl.* (Ex. 1003) ¶¶ 75-155. When compared with the prior art described below and in light of the knowledge of a POSITA, any "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 at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. § 103 (pre-AIA). *See Wolfe Decl.* (Ex. 1003) ¶¶ 75-155. The proposed rejections are listed below and are described in further detail in the attached charts.

B. Ground 1: Bigbee under 35 U.S.C. § 102

Claims 1-8 and 13-22 of the '958 Patent are anticipated by Bigbee. For ease of reference, Requester has provided proposed anticipation rejections for claims 1-8 and 13-22 below. Requester has also provided a detailed mapping of Bigbee to claims 1-8 and 13-22 in Appendix A.

Requester notes that the '958 Patent incorporates U.S. Application No. 10/972,888, which was issued as U.S. Patent No. 7,356,733 to Michaelis et al. (the "'958 Idling Reference"), U.S. Application No. 10/973,075, which was issued as U.S. Patent No. 7,818,614 to Michaelis et al. (the "'958 Recognizing Reference"), and Volume 2: IA-64 System Architecture, *Intel IA-64 Architecture Software Developer's Manual* (1.1 rev. 2000) (the "'958 IA-64 Manual"). *'958 Patent* (Ex. 1001) at 6:31-39, 7:13-21, 7:24-29. The '958 Patent further cites the *Advanced Configuration and Power Interface Specification* (2.0 rev. 2000) (the "'958 ACPI Specification"). *Id.* at 6:60-63. These prior art references, which include over 1000 pages in total, are used by the '958 Patent as background references to provide known implementation details about how firmware performs various limitations of the '958 Patent. These references are likewise used below to provide similar evidence of what was already well-known in the art to support the demonstration of how Bigbee anticipates claims 1-8 and 13-22 of the '958 Patent.

Claim 1: [1PRE] ***A method comprising:***

To the extent the preamble is limiting, Bigbee discloses "[a] ***method*** . . . for functional redundancy check mode recovery." *Bigbee* (Ex. 1004) ¶ [0011].

[1A] ***detecting loss of lockstep for a lockstep pair of processors;***

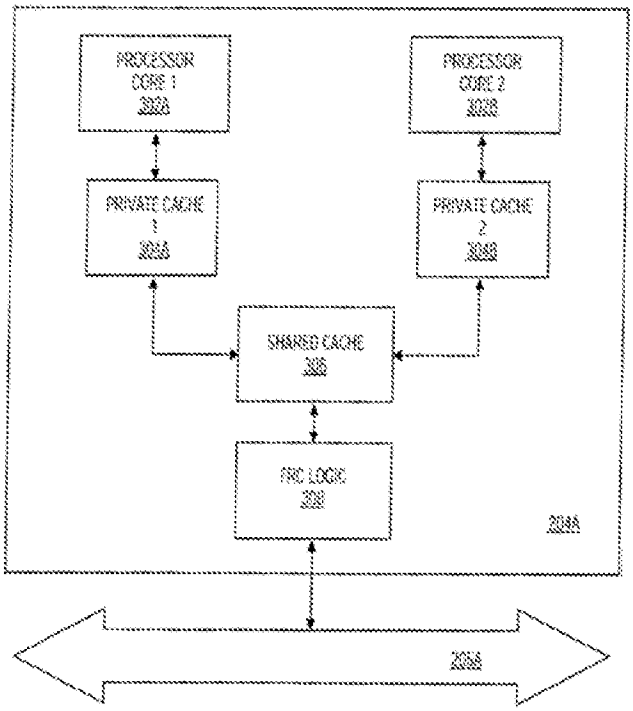
Bigbee discloses this limitation by disclosing (1) "**two or more physical processor cores operating lock step**" and (2) "**[detecting] errors** within individual processor cores . . . **which causes hardware . . . to break lock step**." *Bigbee* (Ex. 1004) ¶ [0019].

Bigbee also discloses *a lockstep pair of processors* by disclosing processors operating in a functional redundancy check (FRC) mode that perform the same operations performed by the lockstep pair of processors disclosed in the '958 Patent. According to the '958 Patent, "in lockstep

processing two processors are paired together, and the two processors perform exactly the same-operations and the results-are compared. (e.g., with an XOR gate). If there is ever a discrepancy between the results of the lockstep processors, an error is signaled." *'958 Patent* (Ex. 1001) at 2:16-20. Similarly, Bigbee discloses "two or more physical processing elements or 'cores'" that

> 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. Specialized FRC logic performs this comparison and generates a signal such as an interrupt or exception once an inconsistent result has been detected.

Bigbee (Ex. 1004) ¶ [0002]. Bigbee additionally discloses *a lockstep pair of processors* by disclosing "two or more processor packages, where each physical processor package includes **two or more physical processor cores operating lock step** in FRC mode." *Bigbee* (Ex. 1004) ¶ [0019], *see also* fig. 3.



Id., fig. 3. Bigbee further discloses *detecting loss of lockstep for a lockstep pair of processors* by disclosing "FRC logic" (i.e., firmware) "[that] **detects errors** within individual processor cores . . . **which causes hardware . . . to break lock step** between at least two processor cores within a processor package, disabling FRC mode operation" and "[that] detects an error within a first

processor core of a plurality of processor cores operating in an FRC mode as a single logical processor." *Bigbee* (Ex. 1004) ¶ [0019], ¶ [0035], *see also* ¶ [0003], ¶ [0033].

For at least the reason that the '958 Patent's description of the operation of "a lockstep pair of processors" is substantially identical to Bigbee's description of two or more processing cores operating lock step in FRC mode, a POSITA would have understood the meaning of *a lockstep pair of processors* to include any two or more processing cores operating in lock step or functional redundancy check (FRC) mode as a single logical processor such as those disclosed in Bigbee. *See Wolfe Decl.* (Ex. 1003) ¶ 83.

Moreover, since the '958 Patent uses "Loss of lockstep" (or "LOL") broadly to "refer to any error in the pair of lockstep processors," a POSITA would have understood the meaning of *detecting a loss of lockstep of a lockstep pair of processors* to include detecting any error in a pair of two or more physical processing elements or cores that are operated in a functional redundancy check (FRC) mode, which would include any or all error events described in Bigbee. '958 *Patent* (Ex. 1001) at 2:27-28), *see also Wolfe Decl.* (Ex. 1003) ¶ 84.

[1B] ***using firmware to***

Bigbee discloses this limitation by disclosing "[f]irmware [that] provides an abstraction layer between . . . [an] operating system . . . and hardware." *Bigbee* (Ex. 1004) ¶ [0014], *see also Wolfe Decl.* (Ex. 1003) ¶ 85.

The '958 Patent defines "firmware" as including "Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI)." *'958 Patent* (Ex. 1001) at 6:31-39. Bigbee similarly states that "**firmware** . . . **includes** an extensible firmware interface (**EFI**), a system abstraction level (**SAL**), and a processor abstraction level (**PAL**)." *Bigbee* (Ex. 1004) ¶ [0018]. The '958 Patent and Bigbee both use "EFI," "SAL," and "PAL" interchangeably with "firmware." For at least these reasons, a POSITA would have understood Bigbee's disclosure of "firmware" to be identical to the '958 Patent and include any and all "EFI," "SAL," and "PAL" functionality. *See Wolfe Decl.* (Ex. 1003) ¶ 86.

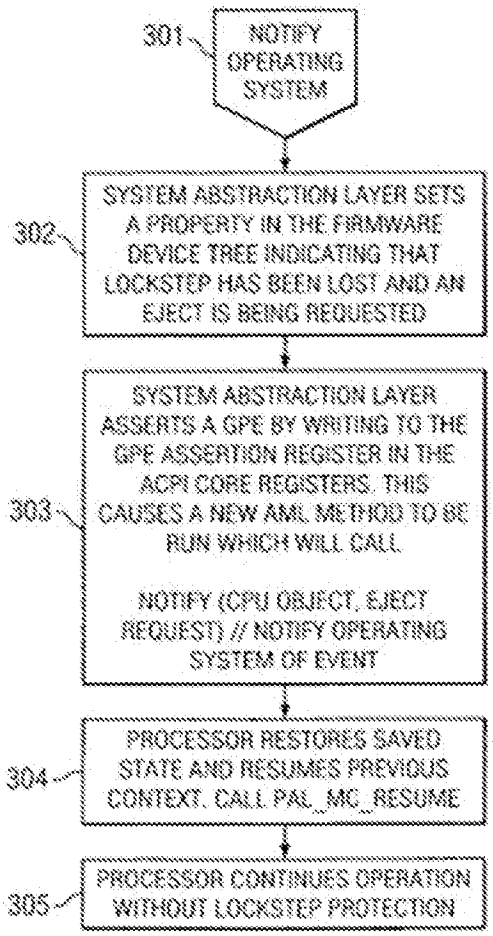
[1Bi] ***trigger an operating system to idle the lockstep pair of processors,***

Bigbee discloses this limitation by disclosing firmware that, in response to a "break" in "lock step," signals "the occurrence of the event to [an] operating system . . . via the generation of a system-visible interrupt (e.g. a system control interrupt or 'SCI')," signals "a request for processor removal or ejection" by "[generating] an SCI to [an] operating system," or "[uses] an

ACPI Notify command" to "[notify] [an] operating System of a request for processor hot removal," which are the same IA-64 methods used by the firmware disclosed in the '958 Patent to *trigger an operating system to idle the lockstep pair of processors* as described in greater detail below. *Bigbee* (Ex. 1004) ¶ [0017], ¶ [0021], ¶ [0022], ¶ [0036], *see also '958 Recognizing Reference* (Ex. 1009) ¶ [0039] ("once LOL is detected for a processor module, system firmware (e.g., SAL) may notify the OS that the processor module is to be **idled (or 'ejected')**"). A POSITA would have recognized that, in disclosing the use of SCI and ACPI commands to respond to loss of lockstep, Bigbee discloses a set of communications between firmware and the OS that results in the OS idling the processor. *See Wolfe Decl.* (Ex. 1003) ¶ 88. In other words, as explained in detail below, the SCI interrupts discussed by Bigbee as signaling a request for processor removal are part of the set of exchanges involved in idling a processor (as discussed in the '958 Idling Reference (Ex. 1008)) through writes to general-purpose registers that are involved in SCIs (as discussed in the '958 ACPI Specification (Ex. 1011)).

The following is a detailed explanation of how idling a processor works in the context of the '958 Patent and why Bigbee discloses idling a processor in the manner claimed. As an initial point, the '958 Patent states that its incorporated '958 Idling Reference (Ex. 1008) describes "[a]n example technique that may be utilized by firmware . . . for triggering [an] OS . . . to idle [a] master processor." *'958 Patent* (Ex. 1001) at 6:31-39. The '958 Idling Reference describes "*firmware . . .* instructing [an] *OS to idle a processor module* for which LOL is detected." *'958 Idling Reference* (Ex. 1008) at 10:49-52, *see also* fig. 3.[6]

[6] Note that step 302 is a step performed by firmware in order to leave a "clue" for itself, while step 303 is the step of actual triggering of the OS to idle a processor. See *'958 Idling Reference* (Ex. 1008) at 10:60-67.



Id., fig. 3. Table 1 demonstrates the identical nature of the methods disclosed in the '958 patent with the disclosure of Bigbee.

Table 1

'958 Patent	**Bigbee**
According to the '958 Idling Reference In operational block 303, **SAL asserts a General Purpose Event Interrupt (GPE)** by writing to the appropriate GPE register in the ACPI register space. **This causes the OS to execute its interrupt handler**, which will run an ACPI Machine Language (AML) Method associated with this particular interrupt. In this example, this AML method that is executed responsive to	Bigbee discloses one of the '958 Patent's methods of using firmware to *trigger an operating system to idle the lockstep pair of processors* by disclosing When an event occurs within data processing system . . ., [firmware] causes **a bit to be set within an event status register, such as a general purpose event (GPE) status register, to indicate the occurrence of the**

the interrupt generated by the GPE includes an AML instruction called "Notify". **This "Notify" operation indicates the CPU object that has lost lockstep (i.e., identifies which processor module lost its lockstep) and that an Eject is being requested as arguments. The AML instruction notifies the OS that action is being requested on this CPU**. The AML method then returns and **the OS can take action asynchronously to satisfy the notify request**. *Id.* at 11:4-16. In addition, the '958 ACPI Specification (Ex. 1011) states that "general-purpose event registers contain the event programming model for generic features" and "**[a]ll generic events generate SCIs.**" *'958 ACPI Specification* (Ex. 1011) at 15.	**event**. If a corresponding bit is set within an event enable register, **the occurrence of the event may be signaled to [an] operating system . . . via the generation of a system-visible interrupt (e.g. a system control interrupt or "SCI")**. The **operating system . . . receives the system-visible interrupt, determines what event caused the interrupt, and then services the event by executing program code (e.g. control methods) corresponding to bits set in the event status register**. *Bigbee* (Ex. 1004) ¶ [0017], *see also* ¶ [0035], ¶ [0036] ("**Firmware** responds to the OS query by **notifying the operating System** of a request for processor hot removal . . . **using an ACPI Notify command**. The **operating system then modifies its internal data Structures to indicate that the processor is offline**.").
The '958 Patent further discloses "**firmware [that] utilizes an ACPI method . . . to 'eject' [a] master processor . . ., thereby triggering [an] OS . . . to idle the master processor . . .** (i.e., stop scheduling tasks for the processor)." *'958 Patent* (Ex. 1001) at 6:9-15, *see also '958 Recognizing Reference* (Ex. 1009) ¶ [0039] ("once LOL is detected for a processor module, **system firmware (e.g., SAL)** may notify the OS that the processor module is to be idled (or 'ejected')"). Additionally, '958	Bigbee discloses *using firmware to trigger an operating system to idle the lockstep pair of processors* by disclosing firmware that **signals "a request for processor removal or ejection" by "[generating] an SCI to [an] operating system"** and an "[o]perating system [that] receives or detects the **firmware-generated SCI and utilizes ACPI control methods to determine that the interrupt was caused by a request for processor removal or ejection** [and]

ACPI Specification states that a "**System Control Interrupt (SCI)" is a "system interrupt used . . . to notify the OS of ACPI events.**" *'958 ACPI Specification* at 18. A POSITA would have understood the "**ACPI method**" described by the '958 Patent to include any method defined by '958 ACPI Specification including those associated with SCIs. *See Wolfe Decl.* (Ex. 1003) ¶ 89.	responsively modifies its internal data structures to reflect that the package's associated processor is offline." *Bigbee* (Ex. 1004) ¶ [0021], ¶ [0022], *see also* ¶ [0020] ("a platform error handler within a SAL . . . initiates a platform-independent device removal sequence for a processor associated with the processor package by generating a system control interrupt (SCI) to [an] operating system"), ¶ [0036] ("[f]irmware" "[uses] an ACPI Notify command" to "[notify] [an] operating System of a request for processor hot removal").

As can be readily seen in this comparison, Bigbee describes the '958 Patent's methods for *[triggering] an operating system to idle the lockstep pair of processors* with the same system-visible interrupts and ACPI notify commands being used for the same purposes (i.e., requesting processor removal or ejection). *See Wolfe Decl.* (Ex. 1003) ¶¶ 87-89. Accordingly, Bigbee discloses limitation [1Bi]. **[1Bii]** ***recover lockstep for the lockstep pair of processors, and***

Bigbee discloses this limitation by disclosing firmware that "resets" or "reinitializes" a processor package having two processor cores and then "re-enables FRC mode lock step operation between [the] two or more of the package's associated processor cores." *Bigbee* (Ex. 1004) ¶ [0024], *see also* ¶ [0037].

The '958 Patent states that "firmware . . . resets [a lockstep] processor pair" as a way of "[attempting] to recover lockstep for the lockstep processor pair." *'958 Patent* (Ex. 1001) at 6:40-42. Thus, at the very least, Bigbee discloses *using firmware to . . . recover lockstep for the lockstep pair of processors* by disclosing

> **Firmware . . . resets the package** to its original state at the time of operating system . . . initial load/boot. In alternative embodiments of the invention, this reset may comprise a "cold" reset or "RESET" operation or a "warm" or "INIT" reset operation. An associated **SAL reinitializes the package and re-enables FRC mode lock step operation between two or more of the package's associated processor cores** in response to detecting the ACPI BIOS-initiated reset operation.

Bigbee (Ex. 1004) ¶ [0024], *see also* ¶ [0037] ("Firmware . . . resets the processor, re-enables FRC operation of at least the first and Second processor cores and generates an SCI to the Operating System.") A POSITA would have understood that Bigbee, by disclosing "re-enabl[ing] FRC mode lock step operation between [the] two or more of the package's associated processor cores," discloses *using firmware to recover lockstep for the pair of lockstep processors*. *See Wolfe Decl.* (Ex. 1003) ¶¶ 90-92.

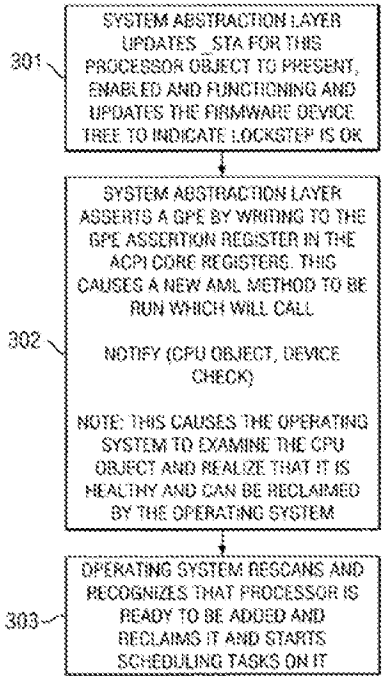
[1Biii] *trigger the operating system to recognize the lockstep pair of processors having recovered lockstep; and*

Bigbee discloses this limitation by disclosing firmware that, after "lock step" is "re-enabled" for a processor package having two processor cores, "initiates a platform-independent device insertion sequence for the processor package's associated processor by generating a system control interrupt (SCI) to [an] operating system" that triggers the "[o]perating system [to] responsively [modify] its internal data structures to reflect that the package's associated processor is being brought online and then wakes and configures the associated processor for use," which are the same IA-64 methods used by the firmware disclosed in the '958 Patent to *trigger the operating system to recognize the lockstep pair of processors having recovered lockstep* as described in greater detail below. *Bigbee* (Ex. 1004) ¶ [0024], ¶ [0025], *see also* ¶ [0037]. In other words, a POSITA would have recognized that Bigbee's discussion of updating internal data structures to bring a processor online discloses *using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep*. *See Wolfe Decl.* (Ex. 1003) ¶ 94. This is particularly clear in view of how the detailed explanation provided by the '958 Patent and its incorporated teachings for recognizing lockstep recovery correspond to Bigbee's discussion of how an OS is alerted that a processor is brought back online. *See Wolfe Decl.* (Ex. 1003) ¶ 94.

A closer look at the recovery processes enabling processor recovery in the '958 Patent and Bigbee reveals that, not only does Bigbee directly read on the claimed triggering step, but that Bigbee also uses the same mechanisms for triggering OS recognition of recovery that the '958 Patent uses. *See Wolfe Decl.* (Ex. 1003) ¶ 95.

For example, the '958 Patent states that its incorporated '958 Recognizing Reference (Ex. 1009) provides an "example technique that may be utilized by firmware . . . for reintroducing a processor to the OS after recovery of lockstep." *'958 Patent* (Ex. 1001) at 7:13-14. The '958 Recognizing Reference includes, as Figure 3, an "operational flow diagram" that "illustrates

operation of one embodiment for reintroducing a processor module to the OS after the processor module's lockstep is reestablished responsive to a detected LOL." *'958 Recognizing Reference* (Ex. 1009) at 10:54-57.



Id., fig. 3. Table 2 demonstrates the identical nature of the methods disclosed in the '958 patent with the disclosure of Bigbee.

Table 2

'958 Patent	**Bigbee**
According to the '958 Recognizing Reference, In operational block 302, **SAL asserts a General Purpose Event (GPE) Interrupt** by writing to the GPE assertion register. This **causes the OS to execute its interrupt handler**, which will run an ACPI Machine Language ("AML") Method associated with this particular interrupt. In this example, this AML method that is executed responsive to the interrupt **generated by the GPE includes an AML instruction called**	Bigbee discloses one of the '958 Patent's methods of using firmware to *trigger an operating system to idle the lockstep pair of processors* by disclosing: When an event occurs within data processing system . . ., firmware . . . causes **a bit to be set within an event status register, such as a general purpose event (GPE) status register, to indicate the occurrence of the event**. If a corresponding bit is set within an event enable register, **the occurrence of the event may be**

"Notify" This "Notify" operation includes an indication of the CPU object that has lost lockstep (i.e., identifies which processor module lost its lockstep) and "Device Check" as arguments. This AML instruction **causes the OS to examine the processor module object (e.g., examine the _STA status for this processor module) and realize that it is healthy and can be reclaimed by the OS** (e.g., because the _STA is set to "Present, Enabled, and Functioning"). . . . **Once the OS receives the Notify event**, in operational block 303, **the OS determines that _STA for the processor module object indicates that the processor module is healthy and ready to be claimed**, and thus the OS claims control of the processor from the system firmware. *Id.* at 10:65-11:19.	**signaled to [an] operating system . . . via the generation of a system-visible interrupt (e.g. a system control interrupt or "SCI")**. The **operating system . . . receives the system-visible interrupt, determines what event caused the interrupt, and then services the event by executing program code (e.g. control methods) corresponding to bits set in the event status register**. *Bigbee* (Ex. 1004) ¶ [0017], *see also* ¶ [0037] ("Once the processor has been inserted and/or activated as described, the **OS modifies its internal data structures to indicate that the processor is online and available**.").
The '958 Patent also states that "[f]irmware . . . utilizes an ACPI method . . .to trigger [an] OS . . . to 'check for' [a] master processor . . ., thereby reintroducing the master processor . . . to [the] OS." *'958 Patent* (Ex. 1001) at 7:7-9. The '958 Patent indicates that the ACPI method is one that triggers the OS to call "_STA."[7] *Id.* at 62-66.	Bigbee discloses one of the '958 Patent's methods of using firmware to *trigger the operating system to recognize the lockstep pair of processors having recovered lockstep* by disclosing "firmware . . . [that] generates an SCI to the Operating System" that triggers "the operating system [to query] firmware to determine the status . . . of the processor" using "one or more control methods (e.g., _STA. . .)" and "[modify] its internal data structures to indicate that the processor is

[7] According to the '958 ACPI Specification, _STA is a "control method that returns a device's status." *'958 ACPI Specification* (Ex. 1011) at 161.

	online and available." *Bigbee* (Ex. 1004) ¶ [0037].

As can be readily seen in this comparison, Bigbee describes the '958 Patent's methods for *[triggering] the operating system to recognize the lockstep pair of processors having recovered lockstep* with the same system-visible interrupts and _STA control methods being used for the same purposes. *See Wolfe Decl.* (Ex. 1003) ¶¶ 93-96. Accordingly, Bigbee discloses limitation [1Biii].

[1C] ***responsive to said detecting loss of lockstep, using said firmware to determine if lockstep is recoverable for the lockstep pair of processors,***

Bigbee discloses this limitation by disclosing firmware that (1) "attempts to correct processor-related errors where possible" and (2) receives a "**recoverable** machine check exception" generated in response to "errors within individual processor cores . . . which causes hardware . . . to break **lock step** between at least two processor cores." *Bigbee* (Ex. 1004) ¶ [0019], ¶ [0021], ¶ [0022].

According to the '958 Patent, "firmware . . . determines. . . whether [a] detected LOL is a recoverable LOL" by determining "whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair . . . without crashing the system." *'958 Patent* (Ex. 1001) at 5:51-56. The '958 Patent further clarifies that "lockstep is recoverable for certain detected LOLs (which may be referred to as 'recoverable LOLs'), while lockstep is not recoverable for other detected LOLs (which may be referred to as 'non-recoverable LOLs'). If the lockstep is not recoverable from the detected LOL, then . . . firmware . . . crashes the system." *'958 Patent* (Ex. 1001) at 5:46-49. Thus, Bigbee discloses *using said firmware to determine if lockstep is recoverable for the lockstep pair of processors* by disclosing firmware that "**attempts to correct processor-related errors when possible,**" for example when the error is of a "**recoverable**" type. *Bigbee* (Ex. 1004) ¶ [0021]. A POSITA would have understood that Bigbee's disclosure of attempting to correct processor-related errors, such as loss of lockstep, "**when possible**," e.g., when they are of a "**recoverable**" type, would necessarily involve Bigbee's firmware determining whether such errors are recoverable. *See Wolfe Decl.* (Ex. 1003) ¶¶ 97-99.

[1D] ***wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.***

Bigbee teaches this limitation by disclosing "firmware," that "monitors processor core store operations or "writes" to shared cache 306 to detect **inconsistencies in the generated results which indicate *an FRC mismatch error.*** *Bigbee* (Ex. 1004) ¶ [0033], ¶ [0034].

The '958 Patent describes "[e]rror detect logic 13C . . . for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *'958 Patent* (Ex. 1001) at 4:46-49. The '958 Patent further defines "lockstep mismatch" as "the output of the paired processors not matching." *Id.* at 2:31-33.

Bigbee similarly discloses *detecting if a lockstep mismatch has occurred between the lockstep pair of processors* by disclosing firmware-based logic that detects a "***mismatch*** error" defined as "inconsistencies in the generated results" of two processor cores that "operate in an FRC mode in which identical instructions may be provided to each core and concurrently executed on identical data." *Bigbee* (Ex. 1004) ¶ [0033], *see also* ¶ [0002]. A POSITA would have understood that Bigbee's disclosure of detecting mismatch errors between two processor cores that operate in an FRC mode discloses the claim limitation of *determining if a lockstep mismatch has occurred between the lockstep pair of processors*. *See Wolfe Decl.* (Ex. 1003) ¶¶ 100-103.

Claim 2: ***The method of claim 1 wherein said using firmware to trigger an operating system to idle the lockstep pair of processors further comprises: using the firmware to trigger the operating system to idle a master processor of the lockstep pair of processors.***

Bigbee discloses this limitation by disclosing "**two or more physical processor cores** operating **lock step**" and a dual-core processor having a "first processor core." *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0019].

The '958 Patent defines a "master processor" as either one processor of "a lockstep processor pair" or a "first core" in a "dual core processor." *'958 Patent* (Ex. 1001) at 4:17-22. Thus, Bigbee discloses a *master processor* by disclosing "*two or more physical processor cores* operating *lock step,*" one being a *master processor* as defined by the '958 Patent, and/or by disclosing a dual-core processor having a "first processor core" and "a second processor core," the first processor core being a master processor as defined by the '958 Patent. *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0019]. Accordingly, Bigbee discloses *using the firmware to trigger the operating system*

to idle a master processor of the lockstep pair of processors by disclosing *using the firmware to trigger the operating system to idle* one processor of a lockstep processor pair or a first core in a dual core processor for at least the reasons discussed in **[1Bi]**. A POSITA would have understood that Bigbee's discussion of triggering processor removal anticipates triggering a master processor to idle. *See Wolfe Decl.* (Ex. 1003) ¶¶ 104-106.

Claim 3: ***The method of claim 2 wherein said using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep further comprises: using the firmware to trigger the operating system to recognize the master of the lockstep pair of processors.***

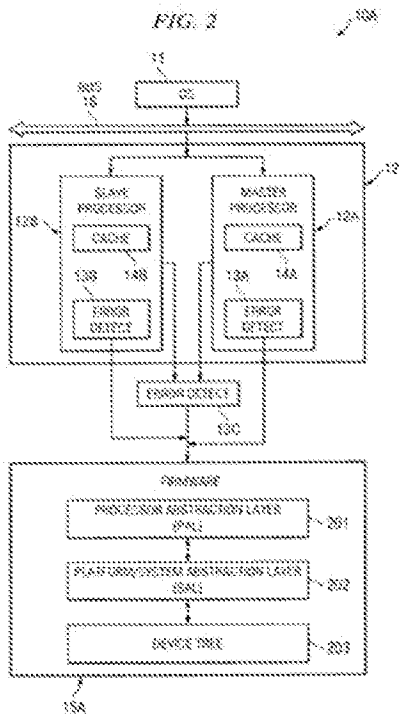
Bigbee discloses this limitation by disclosing "*two or more physical processor cores* operating *lock step*" and a dual-core processor having a "first processor core." *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0019]. Accordingly, Bigbee discloses a master of a *lockstep pair of processors* for the reasons discussed in **[CLAIM 2]** and *triggering the operating system to recognize the master of the lockstep pair of processors* by disclosing *using the firmware to trigger the operating system to recognize* one processor of a lockstep processor pair or a first core in a dual core processor for the reasons discussed in **[1Biii]**. *See Wolfe Decl.* (Ex. 1003) ¶ 107.

Claim 4: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon.***

Bigbee discloses this claim by disclosing "FRC logic" (i.e., firmware) that "**detects errors within individual processor cores**" that are either "single bit data **error correcting code (ECC) errors** occurring in processor **cache[s]**" "or multi-bit **data ECC or parity errors** occurring in processor **cache[s]**." *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019], ¶ [0034].

The '958 Patent describes "[e]rror detect logic 13A and 13B [that] include logic for detecting errors, such as **data cache errors**, present in their respective processors." *'958 Patent* (Ex. 1001) at 4:42-44. The '958 Patent also describes "[e]xamples of error detect logic 13A and 13B [that] include known **parity-based mechanisms and ECC mechanisms**." *Id.* at 4:44-46. The '958 Patent describes separate "[e]rror detect logic 13C . . ., which may include an XOR (exclusive OR) gate, for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *Id.* at 4:46-49. The '958 Patent states that "[o]ne example of LOL is detection of ***data corruption*** **(e.g., data cache error) in one of the processors by a parity-based mechanism**

and/or ECC mechanism." *Id.* at 2:29-31. The '958 Patent's depiction of error detect logic 13A, 13B, and 13C is illustrated in FIG. 2 below.



Id., fig. 2. The '958 Patent states that each of error detect logic 13A, 13B, and 13C are capable of detecting loss of lockstep. *See id.* at 4:56-65. The '958 Patent explains that error detect logic 13A and 13B are different than error detect logic 13C in that error detect logic 13A and 13B detects "**precursors to lockstep mismatch**" and error detect logic 13C detects actual lockstep mismatch. *See id.*

In light of the '958 Patent's descriptions of error detect logic 13A, 13B, and 13C, Bigbee discloses *detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon* by disclosing "FRC logic" that, like error detect logic 13A and 13B in the '958 Patent, "**detects errors within individual processor cores**," rather than errors involving outputs of two or more processor cores, and by disclosing that **the errors are "single bit data error correcting code (ECC) errors occurring in processor cache[s]" "or multi-bit data ECC or parity errors occurring in processor cache[s]**," which are the same error types and locations described in the '958 Patent. *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019]. In other words, as a POSITA would have recognized, by disclosing detecting errors within individual processor cores that are either ECC errors or parity errors, Bigbee discloses the claimed detection of corrupt data in one of said lockstep pair of processors. *See Wolfe Decl.* (Ex. 1003) ¶¶ 108-112.

Claim 5: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting a precursor to lockstep mismatch.***

Bigbee discloses this claim by disclosing "FRC logic" that "**detects errors within individual processor cores**" that are either "single bit data **error correcting code (ECC) errors** occurring in processor **cache[s]**" "or multi-bit **data ECC or parity errors** occurring in processor **cache[s]**." *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019].

In light of the '958 Patent's descriptions of error detect logic 13A, 13B, and 13C discussed in **[CLAIM 4]**, Bigbee discloses *detecting a precursor to lockstep mismatch* by disclosing "FRC logic" that, like error detect logic 13A and 13B in the '958 Patent, "**detects errors within individual processor cores**," rather than errors involving outputs of two or more processor cores, and by disclosing that **the errors are "single bit data error correcting code (ECC) errors occurring in processor cache[s]" "or multi-bit data ECC or parity errors occurring in processor cache[s]**," which are the same error types and locations detected by error detect logic 13A and 13B in the '958 Patent. *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019]. In other words, a POSITA would have recognized that Bigbee's disclosure of detecting errors occurring in processors cache's discloses the claimed detection of precursors to lockstep mismatch. *See Wolfe Decl.* (Ex. 1003) ¶¶ 113-115.

Claim 6: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting lockstep mismatch between the lockstep pair of processors.***

Bigbee discloses this claim by disclosing "FRC logic 308 [that] monitors processor core store operations or "writes" to shared cache 306 to detect **inconsistencies in the generated results which indicate *an FRC mismatch error.***" *Bigbee* (Ex. 1004) ¶ [0033].

As described above, the '958 Patent describes "[e]rror detect logic 13C . . . for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *'958 Patent* (Ex. 1001) at 4:46-49. The '958 Patent further defines "lockstep mismatch" as "the output of the paired processors not matching." *Id.* at 2:31-33.

Bigbee similarly discloses *detecting lockstep mismatch between the lockstep pair of processors* by disclosing similar logic that detects a "***mismatch*** error" defined as "inconsistencies in the generated results" of two processor cores that "operate in an FRC mode in which identical instructions may be provided to each core and concurrently executed on identical data." *Bigbee* (Ex. 1004) ¶ [0033], *see also* ¶ [0002], *Wolfe Decl.* (Ex. 1003) ¶¶ 116-118.

Claim 7: ***The method of claim 1 further comprising: if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable.***

Bigbee discloses this claim by generating a "**recoverable** machine check exception" in response to detecting "errors within individual processor cores . . . which causes hardware . . . to break lock step between at least two processor cores," which is one way in which the '958 Patent discloses determining *that a lockstep mismatch has not occurred between the lockstep pair of processors* as discussed above in connection with **[CLAIM 4]**. *Bigbee* (Ex. 1004) ¶ [0019].

The '958 Patent states that "firmware . . . determines. . . whether [a] detected LOL is a recoverable LOL" by determining "whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair . . . without crashing the system." *'958 Patent* (Ex. 1001) at 5:51-56. The '958 Patent further clarifies that "lockstep is recoverable for certain detected LOLs (which may be referred to as 'recoverable LOLs'), while lockstep is not recoverable for other detected LOLs (which may be referred to as 'non-recoverable LOLs'). If the lockstep is not recoverable from the detected LOL, then . . . firmware . . . crashes the system." *'958 Patent* (Ex. 1001) at 5:46-49.

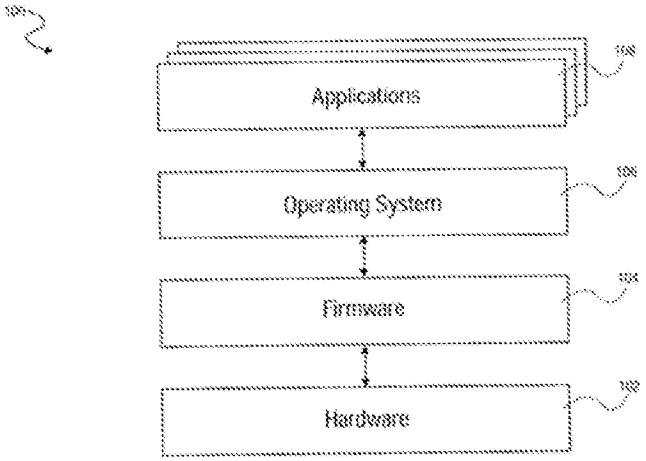
Bigbee similarly discloses *if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable* by disclosing "FRC logic [that] detects errors within individual processor cores . . . which causes hardware . . . to break lock step between at least two processor cores within a processor package" and "[h]ardware [that] disables any of the package's error containing cores. . . and generates a **recoverable** machine check exception." *Bigbee* (Ex. 1004) ¶ [0019]. Bigbee further discloses this limitation by disclosing firmware that "**attempts to correct processor-related errors where possible**" *Bigbee* (Ex. 1004) ¶ [0021]. A POSITA would have recognized that Bigbee's disclosure of attempting to correct processor-related errors "**where possible**" includes *determining that the loss of lockstep is recoverable* after it is *determined that a lockstep mismatch has not occurred between the lockstep pair of processors. See Wolfe Decl.* (Ex. 1003) ¶¶ 119-122.

Claim 8: ***The method of claim 1 further comprising: if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable.***

Bigbee discloses this claim by disclosing firmware that "**attempts to correct processor-related errors where possible**." *Bigbee* (Ex. 1004) ¶ [0021].[8] POSITA would have recognized that Bigbee's discussion of attempting to correct processor-related errors where possible would include identifying situations where recovering lockstep is not possible. *See Wolfe Decl.* (Ex. 1003) ¶¶ 123-124.

Claim 13: [13PRE] ***A system comprising:***

To the extent the preamble is limiting, Bigbee discloses "a data processing **system** 100." *Bigbee* (Ex. 1004) ¶ [0014], *see also* fig. 1.



Id., fig. 1.

[13A] ***an Advanced Configuration and Power Interface (ACPI)-compatible operating system;***

Bigbee discloses this limitation by disclosing "**an advanced configuration and power interface (ACPI)-compliant data processing system**." including an "**operating system**" that utilizes various "**ACPI control methods**" and "**ACPI** machine language (AML) program code." *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0016], ¶ [0025].

[8] **NOTE**: Claim 1 recites recovering from loss of lockstep. The limitation of claim 8 suggests that claim 1 is not limited by at least limitations **[1Bii] and [1Biii]** as it would be impossible to recover from something that is not recoverable. *See* M.P.E.P § 2111.04.

For example, Bigbee states that

> Accordingly, **operating system** 106 includes operating system-directed power management (OSPM) program code, and an *ACPI* device driver and firmware 104 comprises an **ACPI** BIOS including *ACPI* machine language (AML) program code describing what hardware devices are present within data processing system 100, as well as their configuration and interfaces, via **ACPI control methods, objects, registers, and tables**. The disclosed *ACPI* device driver within **operating system** 106 acts as an AML interpreter to interpret and execute AML program code.

Bigbee (Ex. 1004) ¶ [0016].

In another example, Bigbee states that "**Operating system** 106 receives the **firmware-generated** SCI and utilizes **ACPI control methods** . . .." *Bigbee* (Ex. 1004) ¶ [0025].

[13B] ***master processor operable to process instructions scheduled by said ACPI-compatible operating system;***

Bigbee discloses this limitation when given its broadest reasonable interpretation in light of the '958 specification.

As discussed in **[CLAIM 2]**, the '958 Patent defines a "master processor" as either one processor of "a lockstep processor pair" or a "first core" in a "dual core processor." *'958 Patent* (Ex. 1001) at 4:17-22. Thus, Bigbee discloses a *master processor* by disclosing "**two or more physical processor cores operating lock step**," one being a **master processor** as defined by the '958 Patent, and/or by disclosing a dual-core processor having a "first processor core" and "a second processor core," the first processor core being a master processor as defined by the '958 Patent. *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0019]. Accordingly, Bigbee discloses a *master processor operable to process instructions scheduled by said ACPI-compatible operating system* by disclosing "two or more physical processing elements or 'cores'" that execute "identical instructions" on "identical data" of a loaded operating system. *See id.* at ¶ [0015].

[13C] ***slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master processor; and***

Bigbee discloses this limitation by disclosing "**two or more physical processor cores operating lock step**" *Bigbee* (Ex. 1004) ¶ [0019], *see also*¶ [0002].

Bigbee discloses "two or more processor packages, where each physical processor package includes **two or more physical processor cores operating lock step** in FRC mode." *Bigbee* (Ex. 1004) ¶ [0019]. Bigbee discloses that when the "two or more physical processing elements or 'cores'" "are operated in a functional redundancy check (FRC) mode . . . identical instructions are

provided to each core and concurrently executed on identical data." *Bigbee* (Ex. 1004) ¶ [0002]. A POSITA would have understood that Bigbee's discussion of "**two or more physical processor cores operating lock step**" anticipates a *slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master processor*. *See Wolfe Decl.* (Ex. 1003) ¶¶ 132-134.

[13Di] ***firmware operable, responsive to detection of loss of lockstep between the master and slave processors, to use an ACPI method to trigger the operating system to idle the master processor,***

Bigbee discloses this limitation by disclosing firmware that, in response to a "break" in "lock step," signals "the occurrence of the event to [an] operating system . . . via the generation of a system-visible interrupt (e.g. a system control interrupt or 'SCI')," signals "a request for processor removal or ejection" by "[generating] an SCI to [an] operating system," or "[uses] an ACPI Notify command" to "[notify] [an] operating System of a request for processor hot removal" for at least the reasons discussed in **[1Bi]**. *Bigbee* (Ex. 1004) ¶ [0017], ¶ [0021], ¶ [0022], ¶ [0036].

[13Dii] (firmware operable to) attempt to recover lockstep between the master and slave processors, and

Bigbee discloses this limitation by disclosing firmware that "resets" or "reinitializes" a processor package having two processor cores and then "re-enables FRC mode lock step operation between [the] two or more of the package's associated processor cores" for at least the reasons discussed in **[1Bii]**. *Bigbee* (Ex. 1004) ¶ [0024], *see also* ¶ [0037].

[13Diii] (firmware operable) if recovery of lockstep is successful then use an ACPI method to reintroduce the master processor to the operating system.

Bigbee discloses this limitation by disclosing firmware that, after "lock step" is "re-enabled" for a processor package having two processor cores, "initiates a platform-independent device insertion sequence for the processor package's associated processor by generating a system control interrupt (SCI) to [an] operating system" that triggers the "[o]perating system [to] responsively [modify] its internal data structures to reflect that the package's associated processor is being brought online and then wakes and configures the associated processor for use" for at least the reasons discussed in **[1Biii]**. *Bigbee* (Ex. 1004) ¶ [0024], ¶ [0025], *see also*¶ [0037].

Claim 14: ***The system of claim 13 further comprising: error detection logic for detecting loss of lockstep between the master and slave processors.***

Bigbee discloses this limitation by disclosing (1) "**two or more physical processor cores operating lock step**" and (2) "**[detecting] errors** within individual processor cores . . . **which causes hardware . . . to break lock step**" for at least the reasons discussed in **[1A]**. *Bigbee* (Ex. 1004) ¶ [0019].

Claim 15: ***The system of claim 13 further comprising: error detection logic for detecting a precursor to loss of lockstep.***

The limitation of **[CLAIM 15]** is substantially similar to the limitation of **[CLAIM 5]**, with the limitation of **[CLAIM 15]** being the *error detection logic*, which corresponds to the **FRC logic** of Bigbee, that performs the detecting in **[CLAIM 5]**. Thus, Bigbee discloses the limitation of **[CLAIM 15]** for the reasons discussed in connection with **[CLAIM 5]**. *See Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019].

Claim 16: ***The system of claim 15 wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep.***

Bigbee discloses this claim by disclosing "FRC logic" that "**detects errors within individual processor cores**" that are either "single bit data **error correcting code (ECC) errors** occurring in processor **cache[s]**" "or multi-bit **data ECC or parity errors** occurring in processor **cache[s]**," which is the same technique disclosed in the '958 Patent being used to detect precursors to loss of lockstep and thus loss of lockstep as described above in connection with **[CLAIM 5]**. *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019].

As mentioned above, the '958 Patent uses "Loss of lockstep" (or "LOL") broadly to "refer to any error in the pair of lockstep processors." '958 *Patent* (Ex. 1001) at 2:27-28). As such, a POSITA would have understood the meaning of *loss of lockstep* to include detecting any error in a pair of two or more physical processing elements or cores that are operated in a functional redundancy check (FRC) mode, which would include the "ECC or parity errors" described in Bigbee as being *treated as said detection of loss of lockstep*. *See Wolfe Decl.* (Ex. 1003) ¶¶ 140-142.

Claim 17: ***The system of claim 15 wherein said error detection logic comprises a parity-based mechanism.***

Bigbee discloses this claim by disclosing "FRC logic" that "**detects errors within individual processor cores**" that are "**parity errors** occurring in processor **cache[s]**." *Bigbee* (Ex. 1004) ¶ [0018], ¶ [0019], *see also Wolfe Decl.* (Ex. 1003) ¶ 143.

Claim 18: ***The system of claim 13 further comprising error detection logic for detecting lockstep mismatch between the master and slave processors.***

The limitation of **[CLAIM 18]** is substantially similar to the limitation of **[CLAIM 6]**, with the limitation of **[CLAIM 18]** being the *error detection logic* that performs the detecting in **[CLAIM 6]**. Thus, Bigbee discloses the limitation of **[CLAIM 18]** for the reasons discussed in connection with **[CLAIM 6]**. *See Bigbee* (Ex. 1004) ¶ [0033].

Claim 19: [19PRE] ***A system comprising:***

To the extent the preamble is limiting, Bigbee discloses "a data processing **system** 100." *Bigbee* (Ex. 1004) ¶ [0014], *see also* fig. 1.

[19A] ***a pair of lockstep processors and***

Bigbee discloses this limitation by disclosing (1) "**two or more physical processor cores operating lock step**" and (2) "**[detecting] errors** within individual processor cores . . . **which causes hardware . . . to break lock step**" for at least the reasons discussed in **[1A]**. *Bigbee* (Ex. 1004) ¶ [0019].

[19B] ***computer-executable firmware code stored to a computer-readable medium, the computer-executable firmware code comprising:***

Bigbee discloses this limitation by disclosing "executable firmware" that "provides an abstraction layer between . . . [an] operating system . . . and hardware," the firmware being stored in "memory." *Bigbee* (Ex. 1004) ¶ [0014], ¶ [0029] ("memory [storing] one or more firmware interfaces (e.g. SAL or ACPI BIOS program code)"), ¶ [0034].

[19Bi] ***firmware code, responsive to detection of loss of lockstep between the pair of lockstep processors, for determining if the lockstep is recoverable for the pair of lockstep processors, wherein said firmware code for determining if the lockstep is recoverable further comprises firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors; and***

Bigbee discloses this limitation by disclosing firmware that (1) "attempts to correct processor-related errors where possible," (2) receives a "*recoverable* machine check exception"

generated in response to "errors within individual processor cores . . . which causes hardware . . . to break *lock step* between at least two processor cores," and (3) "monitors processor core store operations or "writes" to shared cache 306 to detect **inconsistencies in the generated results which indicate *an FRC mismatch error***" for at least the reasons discussed in **[1C]** and **[1D].** *Bigbee* (Ex. 1004) ¶ [0019], ¶ [0021], ¶ [0022], ¶ [0033], ¶ [0034].

[19Bii] ***firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating system to idle the processors, attempting to recover lockstep between the pair of processors, and if lockstep is successfully recovered between the pair of processors, triggering said operating system to recognize the processors as being available for receiving instructions.***

Bigbee discloses *firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating system to idle the processors* by disclosing firmware that, in response to a "break" in "lock step," signals "the occurrence of the event to [an] operating system . . . via the generation of a system-visible interrupt (e.g. a system control interrupt or 'SCI')," signals "a request for processor removal or ejection" by "[generating] an SCI to [an] operating system," or "[uses] an ACPI Notify command" to "[notify] [an] operating System of a request for processor hot removal" for at least the reasons discussed in **[1Bi]**. *Bigbee* (Ex. 1004) ¶ [0017], ¶ [0021], ¶ [0022], ¶ [0036].

Bigbee further discloses *firmware code, responsive to determining that the lockstep is recoverable, for . . . attempting to recover lockstep between the pair of processors* by disclosing firmware that "resets" or "reinitializes" a processor package having two processor cores and then "re-enables FRC mode lock step operation between [the] two or more of the package's associated processor cores" for at least the reasons discussed in **[1Bii]**. *Bigbee* (Ex. 1004) ¶ [0024], *see also*¶ [0037].

Bigbee discloses *firmware code, responsive to determining that the lockstep is recoverable, for . . . triggering said operating system to recognize the processors as being available for receiving instructions* by disclosing firmware that, after "lock step" is "re-enabled" for a processor package having two processor cores, "initiates a platform-independent device insertion sequence for the processor package's associated processor by generating a system control interrupt (SCI) to [an] operating system" that triggers the "[o]perating system [to] responsively [modify] its internal data structures to reflect that the package's associated processor is being brought online and then

wakes and configures the associated processor for use" for at least the reasons discussed in **[1Biii]**. *Bigbee* (Ex. 1004) ¶ [0024], ¶ [0025], *see also*¶ [0037].

Claim 20: ***The system of claim 19 wherein the code for attempting to recover lockstep further comprises: code for resetting the pair of processors.***

Bigbee discloses this limitation by disclosing firmware that "resets" or "reinitializes" a processor package having two processor cores and then "re-enables FRC mode lock step operation between [the] two or more of the package's associated processor cores." *Bigbee* (Ex. 1004) ¶ [0024], *see also Bigbee* (Ex. 1004) ¶ [0037].

Claim 21: ***The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering the operating system to idle the processors comprises: code for using an ACPI method to idle the processors.***

The limitation of **[CLAIM 21]** is substantially similar to the limitations of **[13A]** and **[1Bi]**. Thus, Bigbee discloses the limitation of **[CLAIM 21]** for at least the reasons discussed in connection with **[13A]** and **[1Bi]**. *See Bigbee* (Ex. 1004) ¶ [0014], ¶ [0016], ¶ [0017], ¶ [0021], ¶ [0022], ¶ [0025], ¶ [0036].

Claim 22: ***The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering said operating system to recognize the processors as being available for receiving instructions comprises: code for using an ACPI method to cause the operating system to check for the processors.***

The limitation of **[CLAIM 22]** is substantially similar to the limitations of **[13A]** and **[1Biii]**. Thus, Bigbee discloses the limitation of **[CLAIM 22]** for at least the reasons discussed in connection with **[13A]** and **[1Biii]**. *See Bigbee* (Ex. 1004) ¶ [0014], ¶ [0016], ¶ [0024], ¶ [0025], ¶ [0037].

C. Ground 2: Bigbee under 35 U.S.C. § 103

As explained above, Bigbee discloses each and every element of claims 1-8 and 13-22. To the extent Patent Owner argues that Bigbee does not expressly disclose each and every element, any differences between the disclosure of Bigbee and the claims of the '958 Patent would have been minor and obvious in view of the knowledge of a POSITA at the time of the earliest priority date of the '958 Patent (e.g., in view of Safford, Marisetty, and Cepulis). *See Wolfe Decl.* (Ex. 1003) ¶¶ 75-155.

D. Ground 3: Safford in view of Marisetty

Claims 1-8 and 13-22 '958 Patent are obvious in view of Safford and Marisetty. *See Wolfe Decl.* (Ex. 1003) ¶¶ 156-283. For ease of reference, Requester has provided some proposed obviousness rejections for claims 1-8 and 13-22 below. Requester has also provided a detailed mapping of Safford and Marisetty to claims 1-8 and 13-22 in Appendix B.

Claim 1: [1PRE] ***A method comprising:***

To the extent the preamble is limiting, Safford discloses a "**method** . . . used to enhance recovery from loss of lock step in a multi-processor computer system." *Safford* (Ex. 1005), abstract.

[1A] ***detecting loss of lockstep for a lockstep pair of processors;***

Safford discloses, or at least renders obvious, this limitation by disclosing (1) **two "processors** [that] may operate . . . in a **lock step mode**" and (2) associated "error detection and signaling logic" that are configured to "**detect an error that will cause a loss of lock step**." *Safford* (Ex. 1005) at 3:1-45, *see also Wolfe Decl.* (Ex. 1003) ¶ 169. Additionally, Safford discloses "[detecting] a loss of lock step. . . [by comparing] outputs from **[two] microprocessor cores**" and detecting "[a] difference in processing" that "is by definition a **loss of lock step**." *Id.* at 1:51-55. Safford further discloses "detecting **a loss of lock step** initiating event in a processor." *Id.*, claim 13.

[1B] ***using firmware to***

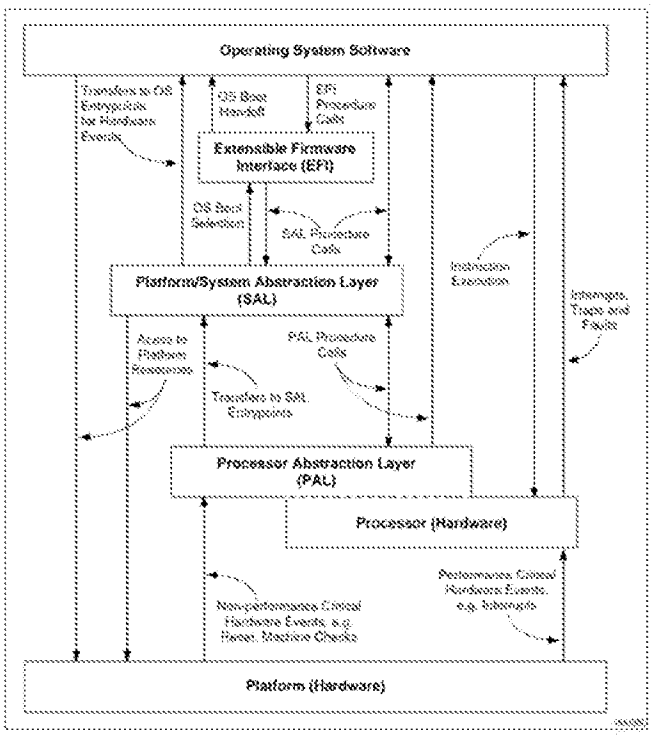
Safford, alone or in combination with Marisetty, discloses, or at least renders obvious, this limitation by disclosing "[a]dvanced computer architectures" and the same IA-64 Firmware utilized in the '958 Patent; that is, a "processor abstraction layer (PAL)" and a "system abstraction layer (SAL) that "can together be known as **firmware**."[9] *Safford* (Ex. 1005) at 1:11-20, *Marisetty* (Ex. 1006) ¶¶ [0024] and [0026], *see also* ¶ [0007] ("An Intel Architecture 64 bit operating system (IA-64 OS) is an operating System written using IA-64 code that runs all IA-64 applications (both IA-64 and IA-32 code)"), fig. 1, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 157-166, 170-175.

[9] Note that the '958 Patent and Safford both use "PAL" and "SAL" interchangeably with "firmware."

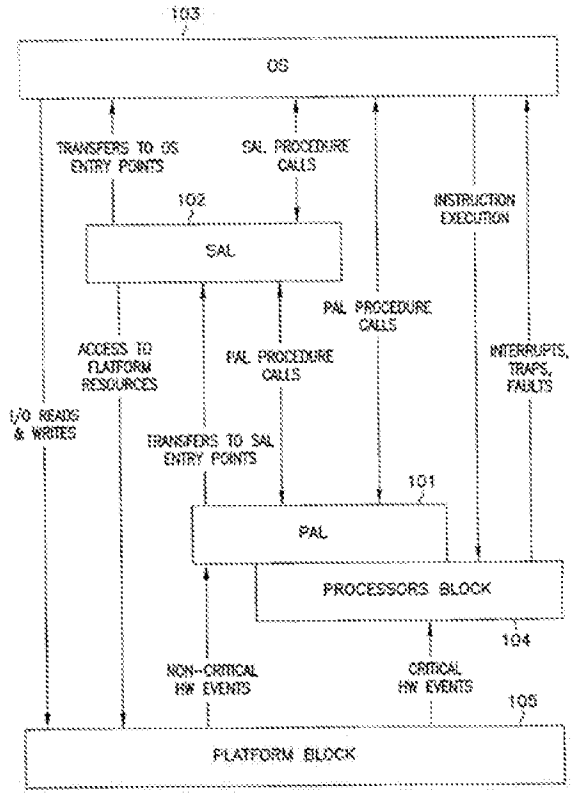
Since Safford does not employ the term "firmware," Safford does not explicitly disclose *using firmware*. However, *using firmware* would have been well within the knowledge, skill level, and common sense of a POSITA at the time Safford was filed and was a well-known characteristic of advanced multi-processor architectures like those disclosed in Safford. *See, e.g., Safford* (Ex. 1005) at 1:11-20, *'958 IA-64 Manual* (Ex. 1010) § 1.7, § 11, *see also Intel EFI Specification* (Ex. 1012) at 1-6, *Intel SAL Specification* (Ex. 1016) § 1.2, *Wolfe Decl.* (Ex. 1003) ¶¶ 170-171. Thus, a POSITA would have understood that Safford uses firmware. *Wolfe Decl.* (Ex. 1003) ¶¶ 170-171.

To the extent Safford does not explicitly disclose firmware, Marisetty discloses the exact IA-64 firmware used in the '958 Patent. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶ 172.

The '958 Patent defines "firmware" as including "Processor Abstraction Layer (PAL) [and] System Abstraction Layer (SAL)." *'958 Patent* (Ex. 1001) at 6:31-39. Moreover, the '958 IA-64 Manual, which is incorporated by reference into the '958 Patent, includes the following illustration of its firmware.



'958 IA-64 Manual § 11. Marisetty, which is assigned to Intel and was filed before the release of the '958 IA-64 Manual, includes the following, nearly identical, illustration of its firmware.



Marisetty (Ex. 1006), fig. 1. For at least these reasons, a POSITA would have understood the "firmware," "PAL," and "SAL" functionality of Marisetty to disclose *using firmware*. *See Wolfe Decl.* (Ex. 1003) ¶¶ 174-175.

[1Bi] ***trigger an operating system to idle the lockstep pair of processors,***

Safford combined with Marisetty discloses, or at least renders obvious, *using firmware to . . . trigger an operating system to idle the lockstep pair of processors*. *See Safford* (Ex. 1005) at 4:36-40, *see also Marisetty* (Ex. 1006) ¶ [0036], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 170-175.

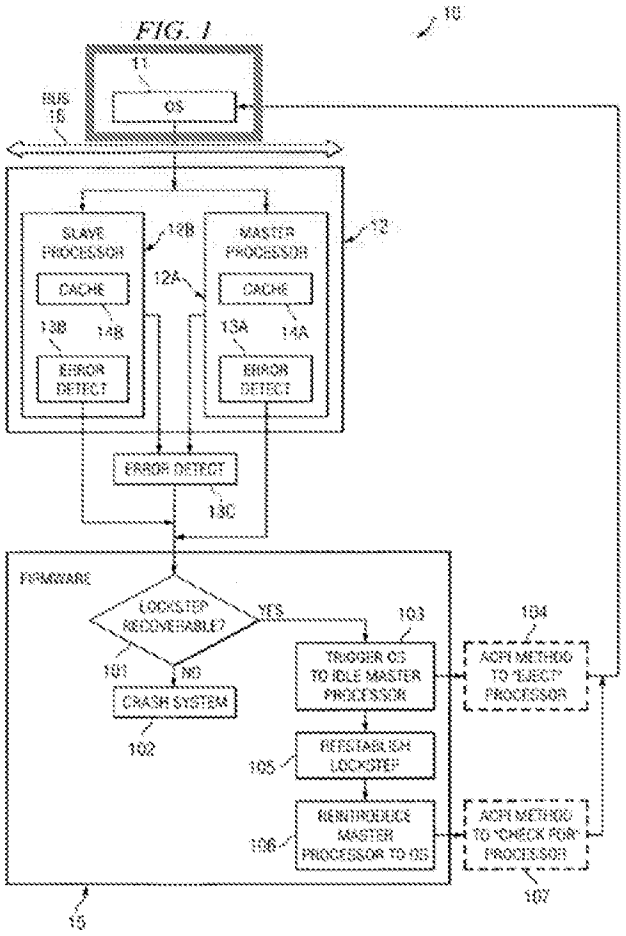
Safford discloses, or at least renders obvious, triggering a node controller to *idle* a *lockstep pair of processors* in response to a loss of lockstep by disclosing a "processor [that] detects an error event that indicates an impending loss of lock step . . and signals [a] node controller . . . that [a] first processor pair . . . has broken lock step and that the first processor pair . . . should be taken 'off-line.'" *Safford* (Ex. 1005) at 4:36-40.

Safford, by itself, does not explicitly disclose triggering an *operating system* as Safford does not use the term "operating system." However, Safford states that "[f]rom the node controller's perspective, each pair of processors appears as a single (logical) processor," which is substantially similar to the perspective of the operating system in the '958 Patent, which "is not

aware of the presence of [a] slave processor . . . but is instead aware of [a] master processor" in a lockstep pair of processors. *Safford* (Ex. 1005) at 1:46-50, *'958 Patent* (Ex. 1001) at 6:18-20, *see also Bigbee* (Ex. 1004) ¶ [0019] ("each physical processor package includes two or more physical processor cores operating lock step in FRC mode as a single logical processor from the point of view of [an] operating system").

Moreover, operating systems would have been well within the knowledge, skill level, and common sense of a POSITA at the time Safford was filed and were a conventional, long-established characteristic of multi-processor systems like those disclosed in Safford. *See, e.g., '958 IA-64 Manual* (Ex. 1010) § 1.7, § 11, *see also Intel EFI Specification* (Ex. 1012) at 1-6, *Intel SAL Specification* (Ex. 1016) § 1.2, *Wolfe Decl.* (Ex. 1003) ¶ 179, *see also* '958 *Patent* (Ex. 1001) ("Prior solutions attempting to resolve at least some detected SDCs without requiring the system to be crashed have been Operating System ("OS") centric."), *Bartlett* (Ex. 1013) at § 4.2. This view was also held by the examiner during prosecution. *See U.S. File History* (Ex. 1002) at 170.

While the Applicant did argue that the "node controller" in Safford is "not an operating system" since "in fig. 3 [of Safford,] the node controller 130 is illustrated as an independent block alongside the CPU hardware blocks, and does not appear to comprise an operating system running on a CPU" (*see id.* at 136-154), the Applicant's argument is nonsensical in light of the '958 Patent's own figures, which illustrate an operating system four times (each time as an independent block alongside a CPU hardware block). *See, e.g.,* '958 *Patent* (Ex. 1001), fig. 1



Id., fig. 1.

> During prosecution the Applicant also argued without basis that
>
> Safford signals to the node controller that a processor pair should be taken offline, not that an operating system should idle the processor. If Safford's node controller merely disables the processor pair by taking it offline, this may lead to the operating system timing out and crashing as described in paragraph 8 of Applicants' specification.

U.S. File History (Ex. 1002) at 154. Applicant's argument is nonsensical, for at least the reason that the entire point of Safford is to enable continued operation of the system during recovery from loss of lockstep. *See, e.g.*, *Safford* (Ex. 1005) at 2:1-9, 3:28-31. Moreover, when reading the '958 Patent's disclosure, a POSITA would have understood *idle* to have its ordinary and customary meaning and to include any offline state of a processor in which the operating system does not schedule code to be executed by the processor.

To the extent Safford does not explicitly disclose *using firmware* to *trigger an operating system to idle the lockstep pair of processors*, Marisetty discloses, or at least renders obvious, *using*

firmware to . . . trigger an operating system to idle all but one processor in a multiprocessor system by disclosing "[a] **SAL error handling routine** [that] causes the system to enter a rendezvous state" by "[selecting] a single processor also known as a monarch processor to handle the error and [by causing] the other processors [to] become **idle** or enter spin loops." *Marisetty* (Ex. 1006) ¶ [0036]. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶¶ 176-184.

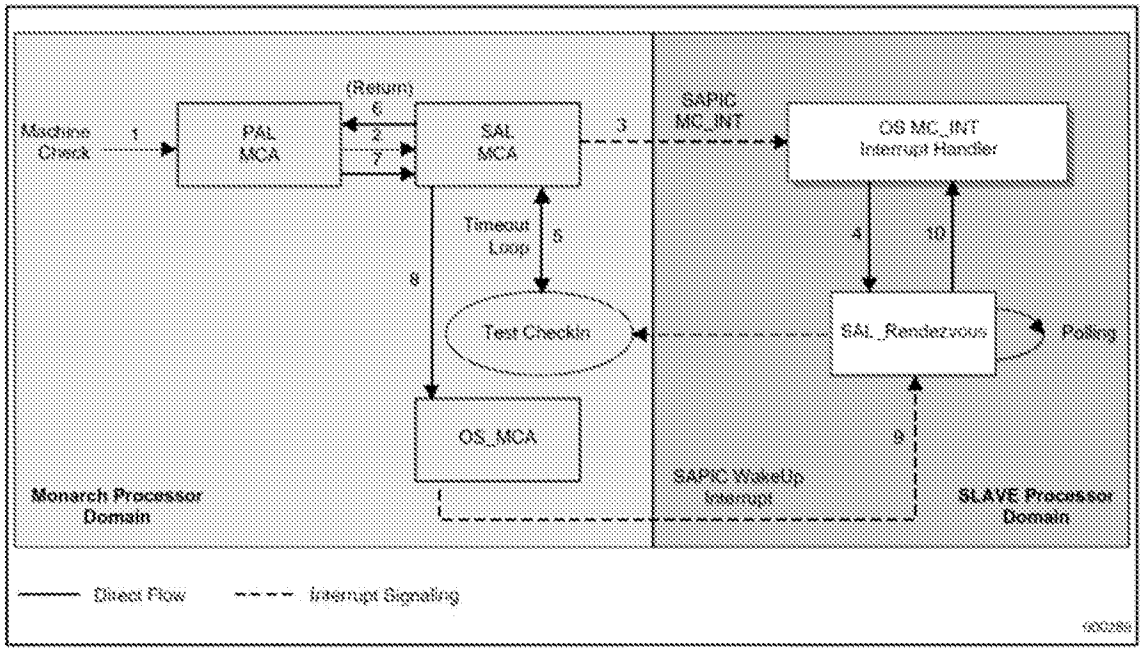
According to Marisetty, an "error handling routine in PAL 101 initiates a rendezvous process by signaling or accessing SAL," "SAL . . . can inform the OS ... layer of the request for rendezvous state[, and the] OS . . . can then inform SAL . . . that it is ready to enter rendezvous state and tells SAL . . . to enter the rendezvous state." *Id.* ¶ [0036]. Marisetty further states that "the layers may inform each other by using interrupts." *Id.*

As mentioned above, Marisetty is an Intel reference filed just prior to the release of the Intel SAL Specification, which is a companion specification of the '958 IA-64 Manual incorporated in the '958 Patent. As such, it should come as no surprise that Marisetty's above-described OS triggering method is mirrored in the Intel SAL Specification in a section discussing machine checks and processor rendezvous in the context of multi-processor configurations. While describing a "Normal SAL Rendezvous Flow," the Intel SAL Specification states that

> If the PAL machine check layer determines that other processors must be rendezvoused for error containment, it passes an indication to SAL_CHECK to perform the rendezvous and supplies a return address within PAL in GR19. Upon return, PALE_CHECK performs the appropriate action and then calls SAL_CHECK again in the normal manner (with no rendezvous indicator).

Intel SAL Specification (Ex. 1016) § 4.5. A figure of this complete process is provided below for ease of understanding.

Figure 4-4. Normal SAL Rendezvous Flow



Intel SAL Specification (Ex. 1016), fig. 4-4.

As an initial point, the '958 Patent states that its incorporated '958 Idling Reference (Ex. 1008) describes "[a]n example technique that may be utilized by firmware . . . for triggering [an] OS . . . to idle [a] master processor." *'958 Patent* (Ex. 1001) at 6:31-39. The '958 Idling Reference describes

> When an LOL is detected, an MCA (Machine Check Abort) is generated. As soon as the MCA is generated, the "state" of the processor is saved. That means all of the control registers, stack and RSE pointers are moved to a storage area (both memory and extra registers in the processor) so that the application that was running on the processor can be resumed later on. The MCA is processed and then all of the stored pointers and registers are reloaded to exactly what their values were when the MCA occurred. The system firmware calls PAL MC RESUME, which triggers PAL to correct the internal state of the processor. That is, PAL MC RESUME then instructs the processor to return to the exact instruction that was executing when the MCA occurred and begin executing again.

'958 Idling Reference (Ex. 1008) at 11:26-33. While this description lacks many of the details of the known rendezvous methods of Marisetty, a POSITA reading the '958 Idling Reference's technique for utilizing firmware to trigger an operation system to idle a processor with knowledge of the known rendezvous methods of Marisetty, which was readily available as described above, would have understood that there was no difference between the '958 Idling Reference's technique and that of Marisetty. *See Wolfe Decl.* (Ex. 1003) ¶¶ 185-189. For at least this reason, Marisetty

discloses, or at least renders obvious, *using firmware to . . . trigger an operating system to idle* all but one processor in a multiprocessor system.

[1Bii] ***recover lockstep for the lockstep pair of processors, and***

Safford combined with Marisetty discloses, or at least renders obvious, *using firmware to . . . recover lockstep for the lockstep pair of processors*. *See Safford* (Ex. 1005) at 3:48-51, 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0031], ¶ [0025], ¶¶ [0032]-[0035], ¶ [0038], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 190-194.

Safford discloses, or at least renders obvious, using a processor to *recover lockstep for the lockstep pair of processors* by disclosing that "recovery from the loss of lock step (and correction of the data cache error) is executed to restore lock step operation of the processors." *Safford* (Ex. 1005) at 3:48-51, *see also* 4:55-59 ("recovery actions are executed on the first processor pair" and "the node controller 130 'reboots' the processors").

While Safford does not explicitly disclose using firmware to recover lockstep for the lockstep pair of processors, Marisetty's teaching that "in the following discussion, we refer to PAL, SAL, and OS with the understanding that they represent PAL, SAL, or OS code executed by a processor" at least suggests that firmware code is being executed by Safford's "processor [that] detects an error event that indicates an impending loss of lock step," especially in light of the fact that the firmware model disclosed in Marisetty and the '958 IA-64 Manual and the systems on which it operated were at least two years old when Safford was filed. *Safford* (Ex. 1005) at 4:36-37, *Marisetty* (Ex. 1006) ¶ [0026], *see also Wolfe Decl.* (Ex. 1003) ¶ 192.

To the extent Safford does not explicitly disclose *using firmware*, Marisetty discloses, or at least renders obvious, *using firmware to . . . recover* an error involving a processor in a multiprocessor system by disclosing firmware-based error handling in a multiprocessor system, which according to Marisetty "cover all errors that might be encountered in a multiprocessor system." *Marisetty* (Ex. 1006) ¶ [0031], *see also* ¶ [0025] ("A variety of hardware . . . errors can occur in multiprocessor Systems" that "can be handled by utilizing . . . the processor abstraction layer (PAL) [and] System abstraction layer (SAL)"), ¶¶ [0032]-[0035], *see also* ¶ [0038] ("The monarch processor executes an error handling routine to **correct the error**" that "may be in firmware, firmware layers such as SAL and PAL. "Once the error has been handled or corrected by the monarch processor, the system **exits the rendezvous state and all processors resume normal operation**"). It would have been obvious to a POSITA to combine the firmware teachings

of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶¶ 193-194.

[1Biii] *trigger the operating system to recognize the lockstep pair of processors having recovered lockstep; and*

Safford combined with Marisetty discloses, or at least renders obvious, *using firmware to . . . trigger the operating system to recognize the lockstep pair of processors having recovered lockstep*. *See Safford* (Ex. 1005) at 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 195-200.

Safford discloses, or at least renders obvious, a method used to *trigger* a *system to recognize the lockstep pair of processors having recovered lockstep* by disclosing that, after "recovery actions are executed on the first processor pair . . . (e.g., all caches are flushed on the processors . . .)," "the node controller 130 'reboots' the processors . . ., and the processors . . . become the new "hot standby processor pair on the system." *Safford* (Ex. 1005) at 4:55-59.

To the extent Safford does not explicitly disclose *using firmware* or an *operating system*, Marisetty discloses, or at least renders obvious, *using firmware to . . . trigger the operating system to recognize* a processor having recovered from a processor error by disclosing that "[o]nce [an] error has been corrected, the OS is informed that the error has been corrected[, and the] OS can inform all processors to get out of rendezvous state" by sending a 'wake up' signal . . . to the slave processors." *Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058], *see also Intel SAL Specification* (Ex. 1016), fig. 4-4. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶¶ 197-198.

As mentioned above, the '958 Idling Reference describes

> When an LOL is detected, an MCA (Machine Check Abort) is generated. As soon as the MCA is generated, the "state" of the processor is saved. That means all of the control registers, stack and RSE pointers are moved to a storage area (both memory and extra registers in the processor) so that the application that was running on the processor can be resumed later on. The MCA is processed and then all of the stored pointers and registers are reloaded to exactly what their values were when the MCA occurred. The system firmware calls PAL MC RESUME, which triggers PAL to correct the internal state of the processor. That is, PAL MC RESUME then instructs the processor to return to the exact instruction that was executing when the MCA occurred and begin executing again.

'958 Idling Reference (Ex. 1008) at 11:26-33. While this description lacks many of the details of the known rendezvous methods of Marisetty, a POSITA reading the '958 Idling Reference's

technique for utilizing firmware to trigger an operating system to recognize a processor with knowledge of the known rendezvous methods of Marisetty, which was readily available as described above, would have been unable to decern a difference between the '958 Idling Reference's technique and that of Marisetty. *See Wolfe Decl.* (Ex. 1003) ¶ 200. For at least this reason, Marisetty discloses, or at least renders obvious, *using firmware to . . . trigger the operating system to recognize* a processor having recovered from a processor error.

[1C] *responsive to said detecting loss of lockstep, using said firmware to determine if lockstep is recoverable for the lockstep pair of processors,*

Safford combined with Marisetty discloses, or at least renders obvious, *using said firmware to determine if lockstep is recoverable for the lockstep pair of processors*. *See Safford* (Ex. 1005) at 3:10-14, 3:25-28, *see also Marisetty* (Ex. 1006) ¶¶ [0032]-[0035], ¶ [0049], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 201-205.

According to the '958 Patent, "firmware . . . determines. . . whether [a] detected LOL is a recoverable LOL" by determining "whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair . . . without crashing the system." *'958 Patent* (Ex. 1001) at 5:51-56. The '958 Patent further clarifies that "lockstep is recoverable for certain detected LOLs (which may be referred to as 'recoverable LOLs'), while lockstep is not recoverable for other detected LOLs (which may be referred to as 'non-recoverable LOLs'). If the lockstep is not recoverable from the detected LOL, then . . . firmware . . . crashes the system." *'958 Patent* (Ex. 1001) at 5:46-49.

Safford discloses, or at least renders obvious, *[determining] if lockstep is recoverable for the lockstep pair of processors* by disclosing at least two types of error detection and signaling logic, each type detecting different types of errors. *See Safford* (Ex. 1005) at 3:10-14 ("External logic circuit . . . monitors outputs of the processors . . . and may be used to detect any differences in the outputs"), 3:25-28 ("error detection and signaling logic . . . may be included in the processors . . . , respectively, . . . to signal an impending loss of lock step"). As explained in greater detail below, error types detected in Safford are the same type of errors detected in the '958 Patent.

To the extent Safford does not explicitly disclose *using firmware*, Marisetty discloses *using said firmware to determine if* a processor error *is recoverable* by disclosing using a "PAL sublayer" to "[determine] the severity of [an] error" and, "depending on the severity," "request the system enter a rendezvous state." *Marisetty* (Ex. 1006) ¶ [0049]. Marisetty further discloses, or renders

obvious, this limitation by explicitly naming "four categories [of errors] ranging from least severe to most severe" that "cover all errors that might be encountered in a multiprocessor system" and by explicitly indicating if each is "correctable" or "not correctable and require a reboot." *Marisetty* (Ex. 1006) ¶¶ [0032]-[0035]. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶ 205.

[1D] ***wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.***

Safford combined with Marisetty discloses, or at least renders obvious, this claim by disclosing *using said firmware to . . . determining if a lockstep mismatch has occurred between the lockstep pair of processors. See Safford* (Ex. 1005) at 1:51-55, *see also Marisetty* (Ex. 1006) ¶ [0033], *see also Wolfe Decl.* (Ex. 1003) ¶¶ 206-210.

Safford discloses, or at least renders obvious, *determining if a lockstep mismatch has occurred between the lockstep pair of processors* by disclosing detecting "[a] difference in processing" that "is by definition a **loss of lock step**." *Safford* (Ex. 1005) at 1:51-55.

As described above, the '958 Patent describes "[e]rror detect logic 13C . . . for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *'958 Patent* (Ex. 1001) at 4:46-49. The '958 Patent further defines "lockstep mismatch" as "the output of the paired processors not matching." *Id.* at 2:31-33. Thus, Safford similarly discloses, or at least renders obvious, *detecting lockstep mismatch between the lockstep pair of processors* by disclosing an "[e]xternal logic circuit [that] monitors outputs of **[a pair of] processors** . . . to detect any **differences** in the outputs," where "**any difference in the outputs causes a loss of lock step**." *Safford* (Ex. 1005) at 3:10-20, *see also* fig. 2.

To the extent Safford does not explicitly disclose *using firmware*, Marisetty discloses *using said firmware to determine if* a processor error *has occurred* by disclosing a "category of errors [that include] errors correctable using routines in PAL [or] SAL," one example being "a parity error in the processor instruction cache." *Marisetty* (Ex. 1006) ¶ [0033]. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶¶ 209-210.

Claim 2: ***The method of claim 1 wherein said using firmware to trigger an operating system to idle the lockstep pair of processors further comprises: using the firmware to trigger the operating system to idle a master processor of the lockstep pair of processors.***

Safford combined with Marisetty discloses, or at least renders obvious, this limitation by disclosing "lock stepped processor cores that operate in a **master**/checker pair" for at least the reasons discussed in **[1Bi]**. *Safford* (Ex. 1005) at 2:56-57, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 211-212.

Additionally, the '958 Patent defines a "master processor" as either one processor of "a lockstep processor pair" or a "first core" in a "dual core processor." *'958 Patent* (Ex. 1001) at 4:17-22. Thus, Safford additionally discloses, or at least renders obvious, a *master processor* by disclosing (1) **two "processors** [that] may operate . . . in a **lock step mode**" and (2) "a computer system [that] employs **lock stepped** processor **cores**." *Id.* at 2:55-56, 3:1-3, *see also* 1:32-2:9 ("To enhance reliability, the dual core processor, or other multiple microprocessor architected computer systems, may employ lock step features").

Claim 3: ***The method of claim 2 wherein said using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep further comprises: using the firmware to trigger the operating system to recognize the master of the lockstep pair of processors.***

Safford combined with Marisetty discloses, or at least renders obvious, this limitation by disclosing "lock stepped processor cores that operate in a **master**/checker pair" for at least the reasons discussed in **[CLAIM 2]** and **[1Biii]**. *Safford* (Ex. 1005) at 2:56-57.

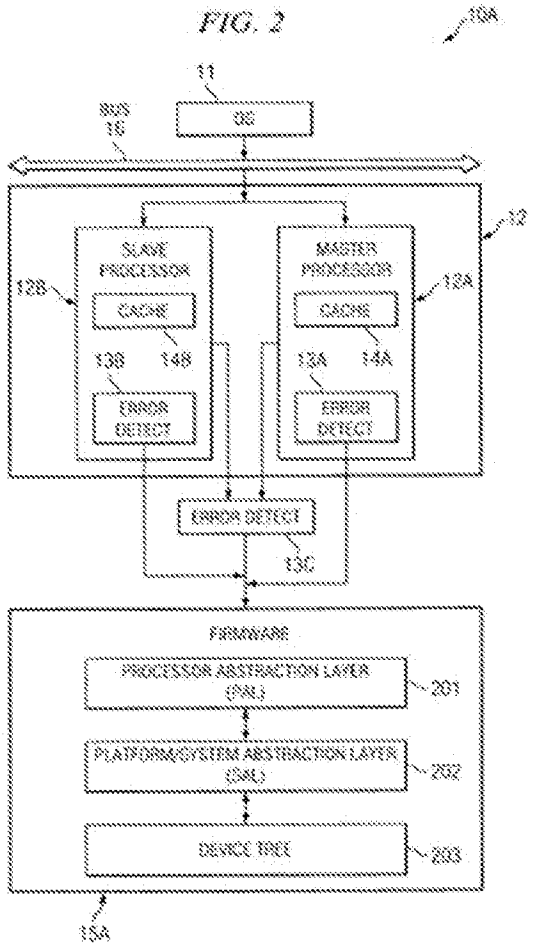
Claim 4: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon.***

Safford discloses, or at least renders obvious, this claim by disclosing "error detection and signaling logic" that "may be included in" a processor "to signal an impending loss of lock step" based on, for example, "a data cache error" involving the processor. *Safford* (Ex. 1005) at 3:25-41, *see also* claim 13 ("detecting **a loss of lock step** initiating event in a processor"), *see also Wolfe Decl.* (Ex. 1003) ¶¶ 214-219.

The '958 Patent describes "[e]rror detect logic 13A and 13B [that] include logic for detecting errors, such as **data cache errors**, present in their respective processors." *'958 Patent* (Ex. 1001) at 4:42-44. The '958 Patent describes separate "[e]rror detect logic 13C . . ., which may

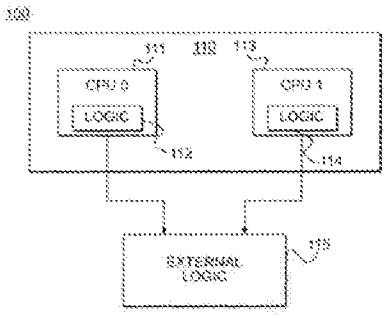
include an XOR (exclusive OR) gate, for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *Id.* at 4:46-49. The '958 Patent states that "[o]ne example of LOL is detection of ***data corruption*** **(e.g., data cache error) in one of the processors**." *Id.* at 2:29-31. The '958 Patent's depiction of error detect logic 13A, 13B, and 13C is provided below.

FIG. 2



Id., fig. 2. The '958 Patent states that each of error detect logic 13A, 13B, and 13C are capable of detecting loss of lockstep. *See id.* at 4:56-65. The '958 Patent explains that error detect logic 13A and 13B are different than error detect logic 13C in that error detect logic 13A and 13B detects "**precursors to lockstep mismatch**" and error detect logic 13C detects actual lockstep mismatch. *See id.*

Figure 2 of Safford illustrates similar logic in similar locations.



Safford (Ex. 1005), fig 2. Like error detect logic 13A and 13B in the '958 Patent, "error detection and signaling logic 112 and 114" in Safford "may be included in" their respective processors "to signal an impending loss of lock step" based on, for example, "a data cache error" of one of the processors. *Id.* at 3:25-41. For example, Safford discloses using the error detection and signaling logic to detect "a data cache error for a cache associated with the processor" that "can be completely corrected (i.e., the processor . . . does not need to be replaced), but will guarantee that [a pair of] processors . . . will break lock step at some future time because the data cache error causes timing differences between the processors." *Id.* at 3:35-41.

In light of the '958 Patent's descriptions of error detect logic 13A and 13B, Safford discloses, or at least renders obvious, *detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon* by disclosing error detection and signaling logic" that "may. . . signal an impending loss of lock step" based on "a data cache error," which is the same error type and location described in the '958 Patent used to *detecting corrupt data . . .that would lead to a lockstep mismatch if acted upon*. *Safford* (Ex. 1005) at 3:25-41, *see also Wolfe Decl.* (Ex. 1003) ¶ 219.

Claim 5: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting a precursor to lockstep mismatch.***

Safford discloses, or at least renders obvious, this claim by disclosing the detection of "a data cache error for a cache associated with [a lockstep] processor" that "can be completely corrected (i.e., the processor . . . does not need to be replaced), but will guarantee that [a pair of] processors . . . will break lock step at some future time because the data cache error causes timing differences between the processors" for at least the reasons discussed in **[CLAIM 4]**. *Safford* (Ex. 1005) at 3: 3:35-41, *see also* claim 13 ("detecting **a loss of lock step initiating event** in a processor"), *see also Wolfe Decl.* (Ex. 1003) ¶¶ 214-219, 220.

Claim 6: ***The method of claim 1 wherein said detecting loss of lockstep comprises: detecting lockstep mismatch between the lockstep pair of processors.***

Safford discloses, or at least renders obvious, this claim by disclosing detecting "[a] difference in processing" that "is by definition a **loss of lock step**." *Safford* (Ex. 1005) at 1:51-55, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 221-222.

As described above, the '958 Patent describes "[e]rror detect logic 13C . . . for detecting a lockstep mismatch between master processor 12A and slave processor 12B. *'958 Patent* (Ex. 1001) at 4:46-49. The '958 Patent further defines "lockstep mismatch" as "the output of the paired processors not matching." *Id.* at 2:31-33. Thus, Safford similarly discloses, or at least renders obvious, *detecting lockstep mismatch between the lockstep pair of processors* by disclosing an "[e]xternal logic circuit [that] monitors outputs of **[a pair of] processors** . . . to detect any **differences** in the outputs," where "**any difference in the outputs causes a loss of lock step**." *Safford* (Ex. 1005) at 3:10-20, *see also* fig. 2.

Claim 7: ***The method of claim 1 further comprising: if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) detecting "a data cache error for a cache associated with [a lockstep] processor" that "can be completely corrected (i.e., the processor . . . does not need to be replaced)," (2) "[using] the detection of this data cache error to signal . . . that the processor . . . is experiencing an error that will cause a loss of lock step," and continuing "processing . . . using the 'good processor'" until "recovery from the loss of lock step (and correction of the data cache error) is executed to restore lock step operation of the processors." *Safford* (Ex. 1005) at 3: 3:35-51, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 223-225.

The '958 Patent "determines. . . whether [a] detected LOL is a recoverable LOL" by determining "whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair . . . without crashing the system." *'958 Patent* (Ex. 1001) at 5:51-56. The '958 Patent further clarifies that "lockstep is recoverable for certain detected LOLs (which may be referred to as 'recoverable LOLs'), while lockstep is not recoverable for other detected LOLs (which may be referred to as 'non-recoverable LOLs'). If the lockstep is not recoverable from the detected LOL, then . . . firmware . . . crashes the system." *'958 Patent* (Ex. 1001) at 5:46-49. As explained above in connection with **[CLAIM 4]** and **[CLAIM 5]**, Safford discloses the detection and recovery from the same types of "recoverable LOLs" found in the '958 Patent. Thus,

by disclosing detection of recoverable types of loss of lockstep, Safford discloses, or at least renders obvious, *if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable*. *Safford* (Ex. 1005) at 3: 3:35-51, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 214-219, 220.

Claim 8: ***The method of claim 1 further comprising: if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable.***

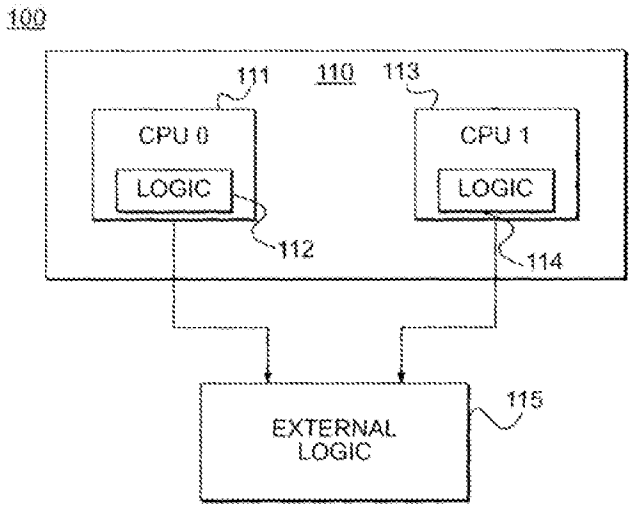
Safford discloses, or at least renders obvious, this claim[10] by disclosing an "[e]xternal logic circuit [that] monitors outputs of **[a pair of] processors** . . . to detect any **differences** in the outputs," where "**any difference in the outputs causes a loss of lock step, and a halt to processing**." *Safford* (Ex. 1005) at 3:10-20, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 226-227.

The '958 Patent "determines. . . whether [a] detected LOL is a recoverable LOL" by determining "whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair . . . without crashing the system." *'958 Patent* (Ex. 1001) at 5:51-56. The '958 Patent further clarifies that "lockstep is recoverable for certain detected LOLs (which may be referred to as 'recoverable LOLs'), while lockstep is not recoverable for other detected LOLs (which may be referred to as 'non-recoverable LOLs'). If the lockstep is not recoverable from the detected LOL, then . . . firmware . . . crashes the system." *'958 Patent* (Ex. 1001) at 5:46-49. In this light, a POSITA would have understood that the '958 Patent's *determining that the loss of lockstep is not recoverable* is equivalent to *determining that the loss of lockstep is not* of a *recoverable* type. *See Wolfe Decl.* (Ex. 1003) ¶ 227. Thus, Safford similarly discloses, or at least renders obvious, *if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable* by disclosing a "difference in the outputs" (i.e., a type of LOL described in '958 Patent as being a "non-recoverable LOL") of a lockstep pair of processors that causes "a halt to processing," which is the same outcome described in the '958 Patent for "non-recoverable LOLs." *Safford* (Ex. 1005) at 3:19-20, *see also* 1:46-47 ("A loss of lock step, if not promptly corrected, may cause the computer system 18 to 'crash'"), 2:7-9.

[10] **NOTE**: Claim 1 recites recovering from loss of lockstep. The limitation of claim 8 suggests that claim 1 is not limited by at least limitations **[1Bii] and [1Biii]** as it would be impossible to recover from something that is not recoverable. *See* M.P.E.P § 2111.04.

Claim 13: [13PRE] ***A system comprising:***

To the extent the preamble is limiting, Safford discloses "a computer **system** . . . that employs processors . . . (central processor unit (CPU) 0) and 113 (CPU 1), which" "may operate in . . . a lock step mode." *Safford* (Ex. 1005) at 2:63-3:3, fig. 2.



Id., fig 2.

[13A] ***an Advanced Configuration and Power Interface (ACPI)-compatible operating system;***

Safford discloses, or at least renders obvious, this limitation by disclosing "[an] apparatus, operating on an advanced multi-core processor architecture, and a corresponding method, [that] are used to enhance recovery from loss of lock step in a computer system." *Safford* (Ex. 1005) at 2:13-16, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 237-240. A POSITA would have understood the meaning of "an advanced multi-core processor architecture" to include *an Advanced Configuration and Power Interface (ACPI)-compatible operating system* for at least the reason that, as of the effective filing date, the Advanced Configuration and Power Interface (ACPI) Specification was considered "the current state-of-the-art" and "fully [defined] the methodology that allows the OS to discover and configure all platform resources." *Intel EFI Specification* (Ex. 1012) at References-5, *see also Wolfe Decl.* (Ex. 1003) ¶ 238. In fact, the '958 Patent admits that, "[i]n an ACPI-compatible system, OS 11 makes ACPI calls to parse the ACPI tables to discover the other processors of a multi-processor system in a manner as is well-known in the art." '958 *Patent* (Ex. 1001) at 8:8-11.

To the extent Safford does not explicitly disclose *an Advanced Configuration and Power Interface (ACPI)-compatible operating system*, Marisetty discloses, or at least renders obvious,

this limitation. For example, Marisetty discloses "an Intel Architecture 64 bit operating system (IA-64 OS)" and the '958 IA-64 Manual's rendezvous method. *Marisetty* (Ex. 1006) ¶ [0007], *see also Wolfe Decl.* (Ex. 1003) ¶ 239. As of the effective filing date of the '958 Patent, *an Advanced Configuration and Power Interface (ACPI)-compatible operating system* was a common component of IA-64 systems. *See, e.g., Intel EFI Specification* (Ex. 1012) at References-5, Glossary-1, *see also Wolfe Decl.* (Ex. 1003) ¶ 239. It would have been obvious to a POSITA to combine the firmware teachings of Marisetty with the teachings of Safford with a reasonable expectation of success for at least the reasons provide in § I.H.3. *See also Wolfe Decl.* (Ex. 1003) ¶ 240.

[13B] ***master processor operable to process instructions scheduled by said ACPI-compatible operating system;***

Safford discloses, or at least renders obvious, this limitation by disclosing "lock stepped processor cores that operate in a **master**/checker pair." *Safford* (Ex. 1005) at 2:56-57, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 241-242.

As discussed in **[CLAIM 2]**, the '958 Patent defines a "master processor" as either one processor of "a lockstep processor pair" or a "first core" in a "dual core processor." *'958 Patent* (Ex. 1001) at 4:17-22. Thus, Safford additionally discloses, or at least renders obvious, a *master processor* by disclosing (1) **two "processors** [that] may operate . . . in a **lock step mode**" and (2) "a computer system [that] employs **lock stepped** processor **cores**." Id. at 2:55-56, 3:1-3, *see also* 1:32-2:9 ("To enhance reliability, the dual core processor, or other multiple microprocessor architected computer systems, may employ lock step features"), *see also Wolfe Decl.* (Ex. 1003) ¶ 242.

[13C] ***slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master processor; and***

Safford discloses, or at least renders obvious, this limitation when given its broadest reasonable interpretation in light of the '958 specification. *See Safford* (Ex. 1005) at 2:56-57, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 243-244.

The '958 Patent defines a "slave processor" as either one processor of "a lockstep processor pair" or a "second core" in a "dual core processor." *'958 Patent* (Ex. 1001) at 4:17-22. Thus, Safford discloses, or at least renders obvious, a *slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master*

processor by disclosing (1) **two "processors** [that] may operate . . . in a **lock step mode**" and (2) "a computer system [that] employs **lock stepped** processor **cores**." *Id.* at 2:55-56, 3:1-3, *see also* 1:32-2:9 ("To enhance reliability, the dual core processor, or other multiple microprocessor architected computer systems, may employ lock step features"), *see also Wolfe Decl.* (Ex. 1003) ¶ 244.

[13Di] ***firmware operable, responsive to detection of loss of lockstep between the master and slave processors, to use an ACPI method to trigger the operating system to idle the master processor,***

The limitation of **[13Di]** is substantially similar to the limitation of **[1Bi]**, with the limitation of **[13Di]** being the *firmware* used in **[1Bi]**. Thus, Safford combined with Marisetty discloses, or at least renders obvious, this limitation for at least the reasons discussed in **[1Bi]**. *See Safford* (Ex. 1005) at 4:36-40, *see also Marisetty* (Ex. 1006) ¶ [0036].

[13Dii] (firmware operable to) attempt to recover lockstep between the master and slave processors, and

The limitation of **[13Dii]** is substantially similar to the limitation of **[1Bii]**, with the limitation of **[13Dii]** being the *firmware* used in **[1Bii]**. Thus, Safford combined with Marisetty discloses, or at least renders obvious, this limitation for at least the reasons discussed in **[1Bii]**. *See Safford* (Ex. 1005) at 3:48-51, 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0031], ¶ [0025], ¶¶ [0032]-[0035], ¶ [0038].

[13Diii] (firmware operable) if recovery of lockstep is successful then use an ACPI method to reintroduce the master processor to the operating system.

The limitation of **[13Diii]** is substantially similar to the limitation of **[1Biii]**, with the limitation of **[13Diii]** being the *firmware* used in **[1Biii]**. Thus, Safford combined with Marisetty discloses, or at least renders obvious, this limitation for at least the reasons discussed in **[1Biii]**. *See Safford* (Ex. 1005) at 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058].

Claim 14: ***The system of claim 13 further comprising: error detection logic for detecting loss of lockstep between the master and slave processors.***

Safford discloses, or at least renders obvious, this limitation by disclosing "**error detection** and signaling **logic**" that are configured to "**detect an error that will cause a loss of lock step**" for at least the reasons discussed in **[1A]**. *Safford* (Ex. 1005) at 3:1-45.

Claim 15: ***The system of claim 13 further comprising: error detection logic for detecting a precursor to loss of lockstep.***

The limitation of **[CLAIM 15]** is substantially similar to the limitation of **[CLAIM 5]**, with the limitation of **[CLAIM 15]** being the *error detection logic* that performs the detecting in **[CLAIM 5]**. Thus, Safford discloses, or at least renders obvious, the limitation of **[CLAIM 15]** for the reasons discussed in connection with **[CLAIM 5]**. *See Safford* (Ex. 1005) at 3: 3:35-41, *see also* claim 13 ("detecting **a loss of lock step initiating event** in a processor").

Claim 16: ***The system of claim 15 wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep.***

Safford discloses, or at least renders obvious, this claim by disclosing "[signaling] an impending loss of lock step" based on, for example, "a data cache error" of one of the processors for at least the reasons discussed in **[CLAIM 4]**. *Safford* (Ex. 1005) at 3:25-41. Safford further discloses *wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep* by disclosing "[using] the detection of [a] data cache error to signal the logic circuit . . . that the processor . . . is experiencing an error that will cause a loss of lock step." *Id.* at 3:42-45.

Claim 17: ***The system of claim 15 wherein said error detection logic comprises a parity-based mechanism.***

Safford, alone or in combination with Marisetty, discloses, or at least renders obvious, this claim by disclosing "[signaling] an impending loss of lock step" based on, for example, "a data cache error" and "[using] the detection of [a] data cache error to signal the logic circuit . . . that the processor . . . is experiencing an error that will cause a loss of lock step." *Safford* (Ex. 1005) at 3:25-41, 3:42-45, *see also Marisetty* (Ex. 1006) ¶ [0033] (describing a "category of errors [that] are errors correctable using routines in PAL [or] SAL," one example being "a parity error in the processor instruction cache"), *see also '958 Patent* at 4:42-46 ("Error detect logic [include] logic for detecting errors, such as data cache errors, present in their respective processors," and "Examples of error detect logic . . . include **known** parity-based mechanisms"), *see also Wolfe Decl.* (Ex. 1003) ¶ 251.

Claim 18: ***The system of claim 13 further comprising error detection logic for detecting lockstep mismatch between the master and slave processors.***

The limitation of **[CLAIM 18]** is substantially similar to the limitation of **[CLAIM 6]**, with the limitation of **[CLAIM 18]** being the *error detection logic* that performs the detecting in **[CLAIM 6]**. Thus, Safford discloses, or at least renders obvious, the limitation of **[CLAIM 18]**

for the reasons discussed in connection with **[CLAIM 6]**. *See Safford* (Ex. 1005) at 1:51-55, 3:10-20, fig. 2.

Claim 19: [19PRE] ***A system comprising:***

To the extent the preamble is limiting, Safford discloses "a computer **system** . . . that employs processors . . . (central processor unit (CPU) 0) and 113 (CPU 1), which" "may operate in . . . a lock step mode." *Safford* (Ex. 1005) at 2:63-3:3, fig. 2.

[19A] ***a pair of lockstep processors and***

Safford discloses, or at least renders obvious, this limitation by disclosing "a computer **system** . . . that employs processors . . . **(central processor unit (CPU) 0) and 113 (CPU 1)**, which" "may operate in . . . a **lock step mode**." *Safford* (Ex. 1005) at 2:63-3:3, fig. 2, *see also Wolfe Decl.* (Ex. 1003) ¶ 256.

[19B] ***computer-executable firmware code stored to a computer-readable medium, the computer-executable firmware code comprising:***

The limitation of **[19B]** is substantially similar to the limitation of **[1B]**, with the limitation of **[19B]** being the *firmware code used in* **[1B]**. Thus, Safford in combination with Marisetty discloses, or at least renders obvious, the limitation of **[19B]** for the reasons discussed in connection with **[1B]**. *See Marisetty* (Ex. 1006) ¶¶ [0024] and [0026], ¶ [0007], fig. 1.

[19Bi] ***firmware code, responsive to detection of loss of lockstep between the pair of lockstep processors, for determining if the lockstep is recoverable for the pair of lockstep processors, wherein said firmware code for determining if the lockstep is recoverable further comprises firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors; and***

The limitation of **[19Bi]** is substantially similar to the limitations of **[1C]** and **[1D]**, with the limitation of **[19Bi]** being the *firmware code* used in **[1C]** and **[1D]**. Thus, Safford in combination with Marisetty discloses, or at least renders obvious, the limitation of **[19Bi]** for the reasons discussed in connection with **[1C]** and **[1D]**. *See Safford* (Ex. 1005) at 1:51-55, 3:10-14, 3:25-28, *see also Marisetty* (Ex. 1006) ¶¶ [0032]-[0035], ¶ [0049].

[19Bii] ***firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating system to idle the processors, attempting to recover lockstep between the pair of processors, and if lockstep is successfully recovered between the pair of processors, triggering said operating system to recognize the processors as being available for receiving instructions.***

The limitation of **[19Bii]** is substantially similar to the limitations of **[1Bi]**, **[1Bii]**, and **[1Biii]**, with the limitation of **[19Bii]** being the *firmware code* used in **[1Bi]**, **[1Bii]**, and **[1Biii]**. Thus, Safford in combination with Marisetty discloses, or at least renders obvious, the limitation of **[19Bii]** for the reasons discussed in connection with **[1Bi]**, **[1Bii]**, and **[1Biii]**. *See Safford* (Ex. 1005) at 4:36-40, *see also Marisetty* (Ex. 1006) ¶ [0036], *see also Safford* (Ex. 1005) at 3:48-51, 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0031], ¶ [0025], ¶¶ [0032]-[0035], ¶ [0038], *see also Safford* (Ex. 1005) at 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058].

Claim 20: ***The system of claim 19 wherein the code for attempting to recover lockstep further comprises: code for resetting the pair of processors.***

Safford discloses, or at least renders obvious, this limitation by disclosing that "both microprocessors" of a "dual core processor [suffering] a loss of lock step" are "reset and reinitialized." *Safford* (Ex. 1005) at 1:64-2:2, *see also Wolfe Decl.* (Ex. 1003) ¶ 259.

Claim 21: ***The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering the operating system to idle the processors comprises: code for using an ACPI method to idle the processors.***

The limitation of **[CLAIM 21]** is substantially similar to the limitations of **[13A]** and **[1Bi]**. Thus, Safford, alone or in combination with Marisetty, discloses, or at least renders obvious, the limitation of **[CLAIM 21]** for at least the reasons discussed in connection with **[13A]** and **[1Bi]**. *See Safford* (Ex. 1005) at 2:13-16, 4:36-40, *see also Marisetty* (Ex. 1006) ¶ [0036].

Claim 22: ***The system of claim 19 wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system, said code for triggering said operating system to recognize the processors as being available for receiving instructions comprises: code for using an ACPI method to cause the operating system to check for the processors.***

The limitation of **[CLAIM 22]** is substantially similar to the limitations of **[13A]** and **[1Biii]**. Thus, Safford, alone or in combination with Marisetty, discloses, or at least renders obvious, the limitation of **[CLAIM 22]** for at least the reasons discussed in connection with **[13A]**

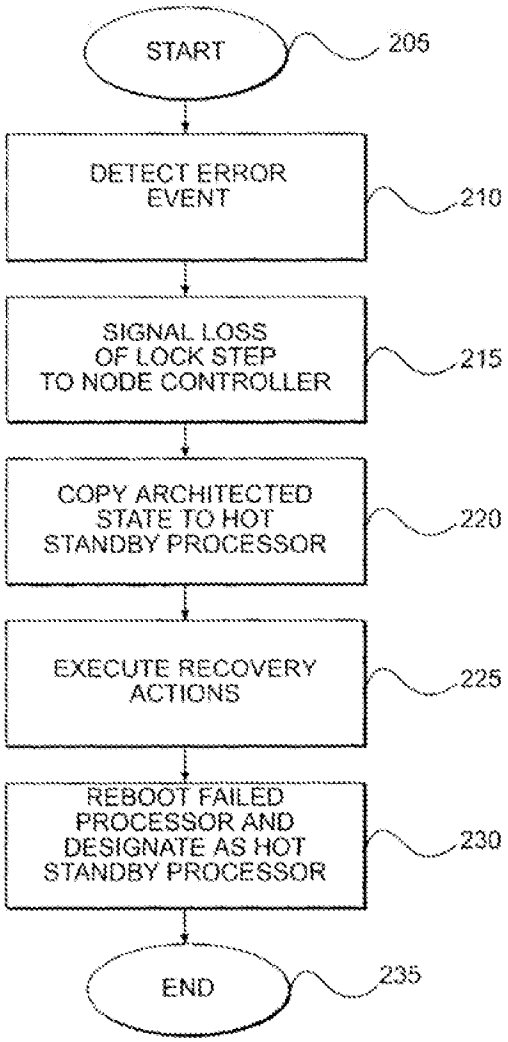
and **[1Biii]**. *See Safford* (Ex. 1005) at 2:13-16, 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058].

E. Ground 4: Safford as Evidenced by Cepulis in Combination with Marisetty

Claims 9-12 and 23-25 of the '958 Patent are obvious in view of Safford and Marisetty when Safford is viewed in light of Cepulis. *See Wolfe Decl.* (Ex. 1003) ¶¶ 157-166, 228-234, 264-282. For ease of reference, Requester has provided some proposed obviousness rejections for claims 9-12 and 23-25 below. Requester has also provided a detailed mapping of Safford and Marisetty in light of Cepulis to claims 9-12 and 23-25 in Appendix B.

Claim 23: [23PRE] ***A method comprising:***

To the extent the preamble is limiting, Safford discloses a "**method** . . . used to enhance recovery from loss of lock step in a multi-processor computer system." *Safford* (Ex. 1005), abstract, *see also*, fig. 4.

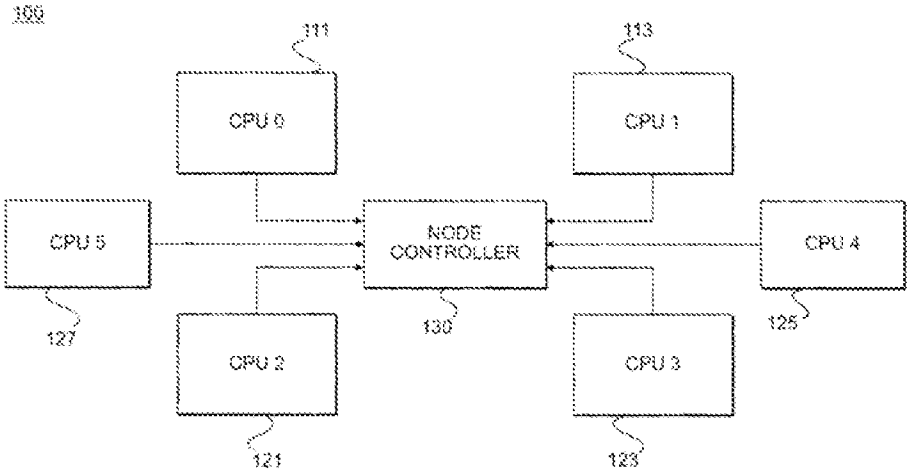


Id., fig. 4.

[23A] ***establishing, in a multi-processor system, a hot spare processor for a system boot processor;***

Safford discloses, or at least renders obvious, this limitation by disclosing (1) "a **multi-processor computer system** . . . [that] includes multiple processor units . . ., each of the processor units having at least two processor units operating in lock step, and at least one **idle processor unit** operating in lock step" that "may be designated as a '**hot standby**' . . . sitting idle" or referred to as a "**spare processor unit** operating idle in lock step." *Safford* (Ex. 1005), abstract, 4:25-26, claim 7, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 266-273.

More specifically, Safford discloses a "computer system 100" having processors 111, 113, 121, 123, 125, and 127. *See Safford* (Ex. 1005) at 3:52-56, fig. 3.



Id., fig. 3. At the beginning of the process illustrated in figure 4, Stafford states that

> The six processors 111, 113, 121, 123, 125, and 127 operate in lock step (i.e., are processing code sequences). For example, the processor 111 operates in lock step with the processor 113, and the processor 121 operates in lock step with the processor 123 and the processor 125 operates in lock step with the processor 127.

Id. at 4:19-24. Stafford further states that

> The processor 125 may be designated as a "hot standby," and is sitting idle in lock step mode with the processor 127. 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. at 4:19-24.

Safford further states that "computer system 100 [initially] employs processors 111 (central processor unit **(CPU) 0**) and 113 (**CPU 1**)" as "**logical CPU 0**" "[w]hen [they are] operating in a lock step mode." *Safford* (Ex. 1005) at 2:63-65, 3:3-5, fig. 3.

A POSITA would have recognized "logical CPU 0" as being the designation used by computing system 100 to identify its boot processor for at least the reason that, in multi-processor system architectures like that described in Safford, "logical CPU 0" is commonly used to indicate that a processor has been selected as a boot processor. *See Cepulis* (Ex. 1007) at 2:52-64, *see also Wolfe Decl.* (Ex. 1003) ¶ 271. For example, during boot-processor selection,

> If CPU P0 is not functioning properly and cannot perform even the most basic functions, it is designated as inoperative. The hardware 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. 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. The process repeats until an operational microprocessor is found to perform the CPU L0 functions.[11]

Cepulis (Ex. 1007) at 2:52-64.

A POSITA would have also recognized "CPU 0" and "CPU 1" as being physical designations used by computing system 100 to identify processors 111 and 113. The use of both logical and physical designators by a computing system is generally necessary when the role of boot processor can be assigned and reassigned to a processor regardless of its physical position within a system. *See Wolfe Decl.* (Ex. 1003) ¶ 272, *see also Safford* (Ex. 1005) at 4:52-53 ("The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100").

For at least these reasons, Safford, discloses, or at least renders obvious, *establishing, in a multi-processor system, a hot spare processor for a system boot processor* by disclosing a designation, in computer system 100, of processor 125 as a hot standby for processors 111 and/or 113, which initially act as the boot processor of computer system 100. *Id.* at 3:3-5, 4:19-24.

[23B] ***detecting loss of lockstep for a lockstep pair of processors in said multi-processor system;***

Safford discloses, or at least renders obvious, this limitation by disclosing (1) **two "processors** [that] may operate . . . in a **lock step mode**" and (2) associated "error detection and signaling logic" that are configure to "**detect an error that will cause a loss of lock step**." *Safford* (Ex. 1005) at 3:1-45, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 274-275. Additionally, Safford discloses "[detecting] a loss of lock step. . . [by comparing] outputs from **[two] microprocessor cores**" and detecting "[a] difference in processing" that "is by definition a **loss of lock step**." *Id.* at 1:51-55. Safford further discloses "detecting **a loss of lock step** initiating event in a processor." *Id.*, claim 13.

More specifically, Safford discloses that "the processor 111 detects an error event that indicates an impending loss of lock step. In block 215, the processor 111 signals the node controller 130 that the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken 'off-line.'" *Safford* (Ex. 1005) at 4:35-40.

---

[11] Cepulis is provided herein to explain the meaning of the term "boot processor" used in Safford, the primary reference. *See* MPEP § 2131.01.

[23C] ***determining if said lockstep pair of processors for which the loss of lockstep is detected is the system boot processor;***

Safford discloses, or at least renders obvious, this limitation by disclosing a pair of processors "operating in a lock step mode" that initially "appear to [a] computer system . . . to be . . . a logical CPU 0" but, later, have the logical CPU 0 designation reassigned to another processor. *See Safford* (Ex. 1005) at 3:4-5, 4:52-53 ("The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100"), *see also Wolfe Decl.* (Ex. 1003) ¶¶ 276-277.

As explained above, a POSITA would have understood "logical CPU 0" to be a boot-processor designation used by an operating system to determine which of its processors is a boot processor and/or to distinguish its boot processor from its non-boot processors. *See Wolfe Decl.* (Ex. 1003) ¶ 277. As such, *determining if* a processor is a *system boot processor* is performed by the operating system through recognition of the boot processor designation. *See Wolfe Decl.* (Ex. 1003) ¶ 277. Thus, Safford discloses, or at least renders obvious, *determining if* a processor is a *system boot processor* by disclosing a pair of processors "operating in a lock step mode" that initially "appear to [a] computer system . . . to be . . . a logical CPU 0" but, later, have the logical CPU 0 designation reassigned to another processor. *See Safford* (Ex. 1005) at 3:4-5, 4:52-53 ("The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100"), *see also Wolfe Decl.* (Ex. 1003) ¶ 277.

[23D] ***if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor, copying a state of the system boot processor to the hot spare processor; and***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0" (logical CPU 0 being a system-required boot-processor designation), (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair," and (5) the "hot standby" processor pair becoming "logical CPU 0" (logical CPU 0 being the system-required boot-processor designation). *Safford* (Ex. 1005) at 3:4-5, 3:25-53, 4:40-43 ("the node controller 130 copies the architected state of the first processor pair 111/113 to the hot standby processor pair 125/127"), *see also Intel Multiprocessor Specification* (Ex. 1017) § B.1 ("The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the BSP operating as the last running processor during

system shutdown. The BSP is not necessarily the first processor, especially in fault-tolerant MP systems in which any available processor can be designated as the BSP."), *see also Wolfe Decl.* (Ex. 1003) ¶ 278.

[23E] ***if determined that said lockstep pair of processors for which the loss of lockstep is not the system boot processor, then triggering an operating system to a) idle the lockstep pair of processors, b) attempt to recover lockstep between the lockstep pair of processors, and c) if lockstep is successfully recovered between the lockstep pair of processors, trigger said operating system to recognize the processors as being available for receiving instructions.***

The limitation of **[23E]** is substantially similar to the limitations of **[1Bi]**, **[1Bii]**, and **[1Biii]** discussed above in § II.D. Thus, Safford in combination with Marisetty discloses, or at least renders obvious, the limitation of **[23E]** for the reasons discussed in connection with **[1Bi]**, **[1Bii]**, and **[1Biii]** above in § II.D. *See Safford* (Ex. 1005) at 4:36-40, *see also Marisetty* (Ex. 1006) ¶ [0036], *see also Safford* (Ex. 1005) at 3:48-51, 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0031], ¶ [0025], ¶¶ [0032]-[0035], ¶ [0038], *see also Safford* (Ex. 1005) at 4:55-59, *see also Marisetty* (Ex. 1006) ¶ [0042], ¶ [0058].

Claim 24: ***The method of claim 23 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising: attempting to recover lockstep between the lockstep pair of processors after copying the state of the system boot processor to the hot spare processor.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0" (logical CPU 0 being the boot-processor designation as explained above), (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair, (5) the "hot standby" processor pair becoming "logical CPU 0" (logical CPU 0 being the system-required boot-processor designation as explained above), (6) "recovery actions" being "executed on the . . . processor pair . . . (e.g., all caches are flushed on the processors), and (7) a rebooting that enables the processor pair to "become [a] new 'hot standby' processor pair." *Safford* (Ex. 1005) at 3:4-5, 3:25-63, *see also*, fig. 4, *see also Wolfe Decl.* (Ex. 1003) ¶ 280.

Claim 25: ***The method of claim 24 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising: if lockstep is successfully recovered between the lockstep pair of processors, establishing the lockstep pair of processors for which lockstep was recovered as a hot spare processor.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0" (logical CPU 0 being the boot-processor designation as explained above), (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair," (5) the "hot standby" processor pair becoming "logical CPU 0" (logical CPU 0 being the system-required boot-processor designation as explained above), (6) "recovery actions" being "executed on the . . . processor pair . . . (e.g., all caches are flushed on the processors), and (7) a rebooting that enables the processor pair to "become [a] new 'hot standby' processor pair." *Safford* (Ex. 1005) at 3:4-5, 3:25-63 ("the operation . . . ends. . . with the processors . . . **idle** and in hot standby"), *see also*, claim 3 ("rebooting the loss of lock step processor unit, whereby the rebooted processor unit becomes a new spare processor unit"), fig. 4, *see also Wolfe Decl.* (Ex. 1003) ¶ 281.

Safford discloses *recovering the lockstep for the lockstep pair of processors* as part of step 225. *See id.* at 4:54-57. Safford discloses *holding the recovered pair of processors in idle as a spare for the processor to which the role of system boot processor was swapped* as part of step 230. *See id.* at 4:57-63 ("the operation . . . ends . . . with the processors . . . **idle** and in hot standby").

Claim 9: ***The method of claim 1 further comprising: determining if said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor.***

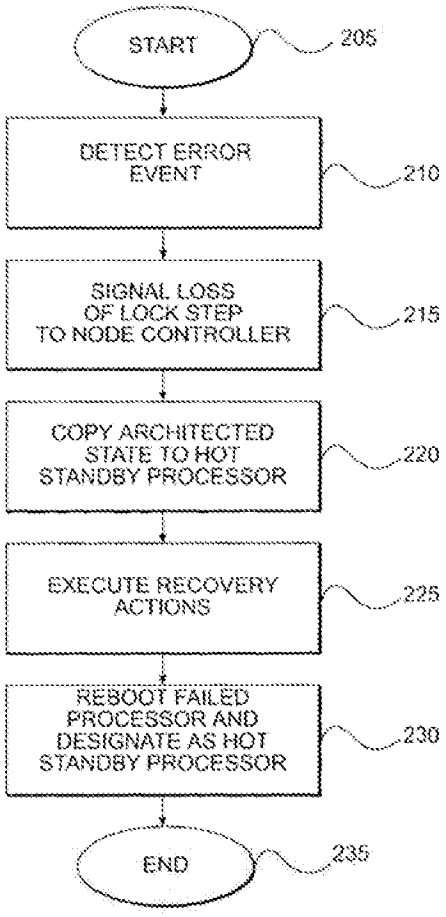
Claim 9 depends from Claim 1, which is obvious in view of Safford and Marisetty. *Supra* § D. Safford discloses, or at least renders obvious, the limitations of Claim 9 by disclosing a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0." *Safford* (Ex. 1005) at 3:4-5. A POSITA would have understood "logical CPU 0" to be a boot-processor designation used by an operating system to determine which of its processors is a boot processor and/or to distinguish its boot processor from its non-boot processors as described in greater detail above in connection with **[23A]**. *See Wolfe Decl.* (Ex. 1003) ¶ 228.

Claim 10: ***The method of claim 9 wherein if determined that said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor, further comprising: swapping the role of system boot processor to another processor in the system.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0," (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair," and (5) the "hot standby" processor pair becoming "logical CPU 0." *Safford* (Ex. 1005) at 3:4-5, 3:25-53. A POSITA would have understood "logical CPU 0" to be a boot-processor designation used by an operating system to determine which of its processors is a boot processor and/or to distinguish its boot processor from its non-boot processors as described in greater detail above in connection with **[23A]**. *See Wolfe Decl.* (Ex. 1003) ¶¶ 229-230.

Claim 11: ***The method of claim 10 further comprising: after said swapping, recovering the lockstep for the lockstep pair of processors.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0," (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair," (5) the "hot standby" processor pair becoming "logical CPU 0," (5) "recovery actions" being "executed on the . . . processor pair . . . (e.g., all caches are flushed on the processors . . .), and (6) a rebooting that enables the processor pair to "become [a] new 'hot standby' processor pair." *Safford* (Ex. 1005) at 3:4-5, 3:25-63, *see also*, fig. 4, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 231-232.



Id., fig. 4. Safford discloses *swapping the role of system boot processor to another processor in the system* as part of step 220. *See id.* at 4:40-54. Safford discloses *recovering the lockstep for the lockstep pair of processors after said swapping* as part of step 225. *See id.* at 4:54-57.

Claim 12: ***The method of claim 11 further comprising: after recovering lockstep for the lockstep pair of processors, holding the recovered pair of processors in idle as a spare for the processor to which the role of system boot processor was swapped.***

Safford discloses, or at least renders obvious, this claim by disclosing (1) a pair of processors "operating in a lock step mode" that "appear to [a] computer system . . . to be . . . a logical CPU 0," (2) "an error condition in the . . . processor pair . . . that indicates an impending loss of lock step," (3) the "processor pair" being "taken 'off-line,'" (4) "the state of the . . . processor pair" being copied to a "hot standby" processor pair," (5) the "hot standby" processor pair becoming "logical CPU 0," (5) "recovery actions" being "executed on the . . . processor pair . . . (e.g., all caches are flushed on the processors . . .), and (6) a rebooting that enables the processor pair to "become [a] new 'hot standby' processor pair." *Safford* (Ex. 1005) at 3:4-5, 3:25-63 ("the

operation . . . ends. . . with the processors . . . **idle** and in hot standby"), *see also*, claim 3 ("rebooting the loss of lock step processor unit, whereby the rebooted processor unit becomes a new spare processor unit"), fig. 4, *see also Wolfe Decl.* (Ex. 1003) ¶¶ 233-234.

Safford discloses *recovering the lockstep for the lockstep pair of processors* as part of step 225. *See id.* at 4:54-57. Safford discloses *holding the recovered pair of processors in idle as a spare for the processor to which the role of system boot processor was swapped* as part of step 230. *See id.* at 4:57-63 ("the operation . . . ends. . . with the processors . . . **idle** and in hot standby").

F. Secondary Considerations

This Request demonstrates that the Challenged Claims of the '958 Patent are unpatentable as obvious in view of the prior art references. The Applicant did not identify any evidence of secondary considerations during prosecution. Further, the clear teachings in the prior art outweighs any supposed "secondary considerations." *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 36 (1966).

III. DISCLOSURE OF CONCURRENT LITIGATION, REEXAMINATION, AND RELATED PROCEEDINGS

Based on information available to Requester, the '958 Patent is the subject of five District Court litigations, as listed below. Requester is unaware of any prior reexamination or other post-grant proceeding in which the '958 Patent is or has been involved.

- *Foras Technologies, Ltd. vs Toyota Motor, Corp.*, Case No. 2:23-cv-00150 (E.D. Tex., Apr. 5, 2023).
- *Foras Technologies, Ltd. vs Kia Corporation*, Case No. 2:23-cv-00219 (E.D. Tex., May 18, 2023).
- *Foras Technologies, Ltd. vs BMW*, Case No. 6:23-cv-00386 (W.D. Tex., May 19, 2023).
- *Foras Technologies, Ltd. vs Nissan Motor Co., Ltd.*, Case No. 1:23-cv-00640 (W.D. Tex., June 6, 2023).
- *Foras Technologies, Ltd. vs Volkswagen, AG*, Case No. 2:23-cv-00314 (E.D. Tex., June 28, 2023)

IV. CONCLUSION

Reexamination and cancellation of claims 1-25 of the '958 Patent is respectfully requested. The Commissioner is hereby authorized to charge Deposit Account 50-6990 under Docket No. 185945.012500 the Ex Parte Reexamination fee of $12,600 under 37 C.F.R. § 1.20(c)(1). Requester believes no other fee is due with this submission, however, the Commissioner is hereby authorized to charge any fee deficiency or credit any over-payment to Deposit Account 50-6990.

Please direct all correspondence in this matter to the undersigned.

Dated: September 1, 2023

Respectfully Submitted,

By: /Jared Lee/
Reg. No. 72,786

GREENBERG TRAURIG, LLP
222 South Main Street, Suite 1730
Salt Lake City, UT 84101
Tel: (801) 478-6900

Counsel for Requester Unified Patents, LLC