UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ZF FRIEDRICHSHAFEN AG, ZF ACTIVE SAFETY AND ELECTRONICS US LLC, AND NISSAN MOTOR COMPANY, LTD.,
Petitioners

v.

FORAS TECHNOLOGIES LTD.,
Patent Owner

Patent No. 7,502,958
Filing Date: October 25, 2004
Issue Date: March 10, 2009
Title: SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION

Inter Partes Review No.: IPR2024-00969

PETITION FOR *INTER PARTES* REVIEW UNDER 35 U.S.C. §§ 311-319 AND 37 C.F.R. § 42.100 *et seq.*

TABLE OF CONTENTS

EXHIBITS

Ex. 1001:	U.S. Patent No. 7,502,958 to Michaelis et al. (“the ’958 Patent”)
Ex. 1002:	Declaration of Dr. Jacob Baker
Ex. 1003:	Curriculum Vitae of Dr. Jacob Baker
Ex. 1004:	Prosecution History of the ’958 Patent
Ex. 1005:	U.S. Patent Application Publication No. US 2003/0126498A1 to Bigbee et al., filed January 2, 2002, published July 3, 2003 (“Bigbee”)
Ex. 1006:	U.S. Patent Application Publication No. US 2004/0123201 Al to Nguyen et al., filed December 19, 2002, published June 24, 2004 (“Nguyen”)
Ex. 1007:	U.S. Patent Application Publication No. US 2004/0006722 Al to Safford, filed July 3, 2002, published Jan. 8, 2004 (“Safford”)
Ex. 1008:	U.S. Patent No. 7,085,959 B2 to Safford, filed July 3, 2002, published August 1, 2006 (“Safford-959”)
Ex. 1009:	U.S. Patent No. 7,296,181 B2 to Safford, filed April 6, 2004, published November 13, 2007 (“Safford-181”)
Ex. 1010:	U.S. Patent Application Publication No. US 2002/0144177 Al to Kondo et al., filed January 31, 2002, published October 3, 2002 (“Kondo”)
Ex. 1011:	U.S. Patent No. 6,516,429 B1 to Bossen et al., filed November 4, 1999, published February 4, 2003 (“Bossen”)
Ex. 1012:	U.S. Patent No. 6,158,015 to Klein, filed March 30, 1998, published December 5, 2000 (“Klein”)
Ex. 1013:	U.S. Patent Application Publication No. US 2005/017216 A1 to Fox et al., filed January 21, 2004, published August 4, 2005 (“Fox”)
Ex. 1014:	U.S. Patent No. 7,404,105 B2 to Arai, filed August 16, 2004, published July 22, 2008 (“Arai”)

Ex. 1015:	U.S. Patent No. 6,675,324 B2 to Marisetty et al., filed September 27, 1999, published January 6, 2004 ("Marisetty")
Ex. 1016:	U.S. Patent Application Publication No. US 2003/0074601 A1 to Schultz et al., filed September 28, 2001, published April 17, 2003 ("Schultz")
Ex. 1017:	U.S. Patent Application Publication No. 2003/0126142 A1 to Tu et al., filed December 31, 2001, published July 3, 2003 ("Tu")
Ex. 1018:	U.S. Patent No. 5,915,082 to Marshall et al., filed June 7, 1996, published June 22, 1999 ("Marshall")
Ex. 1019:	Reserved
Ex. 1020:	PCT Patent Application WO 01/80007 A2 to Suffin, filed April 12, 2001, published Oct. 25, 2001 ("Suffin")
Ex. 1021:	U.S. Patent No. 5,627,962 to Goodrum et al., filed Dec. 30, 1994, published May 6, 1997 ("Goodrum")
Ex. 1022:	Advanced Configuration and Power Interface Specification, Revision 2.0, July 27, 2000 ("ACPI 2.0")
Ex. 1023:	Reserved
Ex. 1024:	Reserved
Ex. 1025:	Prosecution History of U.S. Patent No. 7,356,733
Ex. 1026:	Request for Ex Parte Reexamination of the '958 Patent, Control No. 90/019,245 (Sept. 1, 2023)
Ex. 1027:	Order granting stay in *Foras Technologies Ltd. v. Nissan Motor Company, Ltd. and ZF Friedrichshafen AG*, Case No. 1-23-cv-00640 (W.D. Tex)
Ex. 1028:	Infringement contention cover pleading for ZF in *Foras Technologies Ltd. v. Nissan Motor Company, Ltd. and ZF Friedrichshafen AG*, Case No. 1-23-cv-00640 (W.D. Tex)

Ex. 1029:	Infringement contention cover pleading for Nissan in *Foras Technologies Ltd. v. Nissan Motor Company, Ltd. and ZF Friedrichshafen AG*, Case No. 1-23-cv-00640 (W.D. Tex)
Ex. 1030:	U.S. Patent No. 5,790,850 to Natu, filed September 30, 1996, published August 4, 1998 (“Natu”)

MANDATORY NOTICES

Real Parties in Interest

The real parties-in-interest for this petition are ZF Friedrichshafen AG, ZF North America, Inc., ZF Active Safety and Electronics US LLC, Nissan North America, Inc., and Nissan Motor Company, Ltd.

No unnamed entity is funding, controlling, or directing this Petition for *inter partes* review of the ’958 Patent, or otherwise has an opportunity to control or direct this Petition or Petitioners’ participation in any resulting IPR.

Related Matters

To Petitioners’ knowledge, the ’958 Patent is involved in the following:

Name	Number	Court	Filed
ZF Friedrichshafen AG et al. v. Foras Technologies Ltd.	IPR2024-00727	USPTO	Apr. 19, 2024
HL Klemove Corporation v. Foras Technologies Ltd.	IPR2024-00813	USPTO	Apr. 19, 2024
Valeo SE et al. v. Foras Technologies Ltd.	IPR2024-00823	USPTO	Apr. 18, 2024
Bayerische Motoren Werke AG et al. v. Foras Technologies Ltd.	1-24-cv-00363	E.D. Va.	Mar. 6, 2024
BMW of North America, LLC et al. v. Foras Technologies Ltd.	IPR2023-01373	USPTO	Sept. 8, 2023

Name	**Number**	**Court**	**Filed**
Foras Technologies Limited v. Volkswagen AG et al.	2-23-cv-00314	E.D. Tex.	Jun. 28, 2023
Foras Technologies Limited v. Nissan Motor Company, Ltd. and ZF Friedrichshafen AG	1-23-cv-00640	W.D. Tex.	Jun. 6, 2023
Foras Technologies Ltd. v. Bayerische Motoren Werke AG et al.	6-23-cv-00386	W.D. Tex.	May 19, 2023
Ex Parte Reexamination filed by Unified Patents	Control No. 90/019,245	USPTO	Sept. 1, 2023

This petition is the first *inter partes* review petition to challenge claims 9-12 and 23-25 of the '958 Patent. In IPR2023-01373, unrelated third parties BMW and Bosch challenged claims 1-8 and 13-22 of the '958 Patent. That petition was instituted on March 20, 2024, and Petitioners herein (ZF Friedrichshafen AG, ZF Active Safety and Electronics US LLC, and Nissan Motor Company, Ltd.) filed a copycat petition as IPR2024-00727 and moved to join IPR2023-01373 within the one-month window for doing so. Claims 9-12 depend from claim 1, and this Petition relies on the same prior art (Bigbee-Nguyen) as was used against claim 1 in IPR2023-01373, but uses additional art, such as Goodrum and Suffin, for additional claim elements found in claims 9-12, and for certain claim elements found in claims 23-25.

Lead & Back-Up Counsel

Petitioner designates counsel listed below and consents to electronic service. A power of attorney for counsel is being filed with this Petition.

Lead Counsel	Back-Up Counsel
John R. Hutchins (Reg. No. 43,686) jhutchins@bannerwitcoff.com BANNER & WITCOFF, LTD. 1100 13th Street, NW, Suite 1200 Washington, DC 20005 Tel: (202) 824-3000 Fax: (202) 824-3001	Paul T. Qualey (Reg. No. 45,027) pqualey@bannerwitcoff.com Wesley W. Jones (Reg. No. 56,552) wjones@bannerwitcoff.com BANNER & WITCOFF, LTD. 1100 13th Street, NW, Suite 1200 Washington, DC 20005 Tel: (202) 824-3000 Fax: (202) 824-3001 Binal J. Patel (Reg. No. 42,065) bpatel@bannerwitcoff.com BANNER & WITCOFF, LTD. 71 South Wacker Drive, Suite 3600 Chicago, IL 60606 Tel: (312) 463-5000 Fax: (312) 463-5001

Service Information

Please address all correspondence to counsel at the addresses shown above. Petitioner further consents to electronic service by email at the following address if sent to each of the emails indicated above, as well to: ZFIPRService@bannerwitcoff.com.

I. INTRODUCTION AND RELIEF REQUESTED

ZF Friedrichshafen AG, ZF Active Safety and Electronics US LLC, and Nissan Motor Company, Ltd. ("Petitioners") hereby petition for *inter partes* review under 35 U.S.C. §§ 311-319 and 37 C.F.R. § 42.100 *et seq*. and cancellation of claims 9-12 and 23-25 of U.S. Patent No. 7,502,958 ("the '958 Patent"). This petition is supported by the Declaration of Dr. Jacob Baker, Exhibit 1002.

Claims 9-12 and 23-25 of the '958 Patent are directed to obvious combinations of simple concepts, long known in the art: detecting and correcting loss of lockstep in processors, and identifying and using a spare processor to boot the system in the event a problem (such as loss of lockstep) occurs with respect to the boot processor. More particularly, references such as Bigbee, Nguyen, and Safford teach how to detect and correct a loss of lockstep, as claimed in the '958 Patent. References such as Goodrum and Suffin teach that problems with a boot processor can be ameliorated by designating a spare processor to be the boot processor. In combination they render claims 9-12 and 23-25 obvious.

II. OVERVIEW

A. Technical Background

1. Lockstep/FRC Processors Were Known

Before the '958 Patent, it was typical to improve the fault tolerance of a computer system using lockstep processing. For example, two processors 22 and 24 (or two processor units or two processor cores) are paired together to execute the same instructions on identical data in parallel and their outputs are compared by, e.g., lockstep logic 26. Safford, [0003]; *see also* Bigbee, [0033]; Nguyen, [0006]; Kondo, [0004]; Ex. 1001, 2:13-19. A discrepancy between the outputs is a loss of lockstep. Safford, [0004]; Ex. 1002, ¶58.

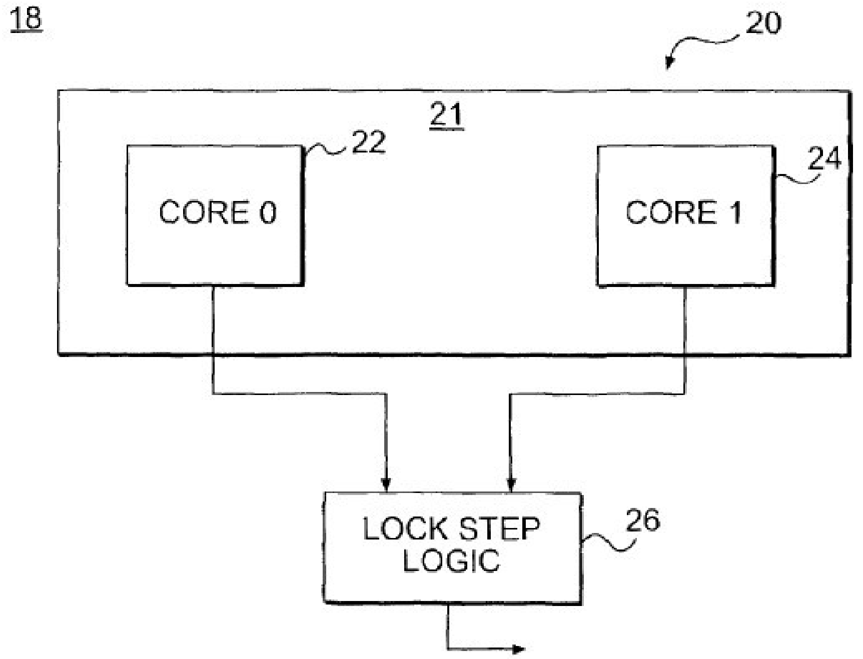


FIG. 1B
PRIOR ART

Safford, Fig. 1B.

Lockstep processors are also said to operate in "functional redundancy check" (FRC) mode, because they execute redundant instructions in parallel. Safford-181, 1:67-2:3; Ex. 1002, ¶59.

If the same outputs are produced, no error is detected. If the comparison indicates a mismatch, it is assumed that at least one processor has an error, such as a data cache error or a soft error.[1] Safford, [0003], [0016]; Nguyen, [0008]. A loss of lockstep, if not promptly corrected, may cause the computer system to crash. That is, a failure of one of the pair of processors may halt the entire computer system, even if the other processor does not encounter an error. Safford [0003]; Ex. 1002, ¶60.

It was known that an error detected in an individual processor of a pair of lockstep processors signals an impending mismatch error. Safford, [0015]-[0017] (error detected in an single processor core of a lockstep processor signals an *impending* loss of lockstep); Kondo, [0101] (detecting an error in the processor provides *early warning* of a divergence), [0103], [0106]; Nguyen, [0008] (soft errors in FRC-enabled processors are likely to be manifested as FRC errors since they take the execution cores out of lockstep), [0022]; Ex. 1002, ¶¶61-62.

[1] A "soft error" results when, for example, a high-energy particle switches the state of a memory element causing a parity or other error. Nguyen, [0004]; Kondo, [0006]. Soft errors are generally correctable by rewriting the affected memory location. So-called "hard" errors, in contrast, indicate failure of a hardware element.

2. Determining Whether Lockstep Is Recoverable Was Known

It was known that lockstep is recoverable from certain errors, such as soft errors, but not others, such as some lockstep mismatch errors. For example, lockstep is recoverable from an error detected in an individual processor before a mismatch error is detected and signaled. As shown below, a pair of lockstep processors 111 and 113 can include error detection and signaling logic 112 and 114, respectively, to signal an impending loss of lockstep. Safford, [0017]; Ex. 1002, ¶¶63-64.

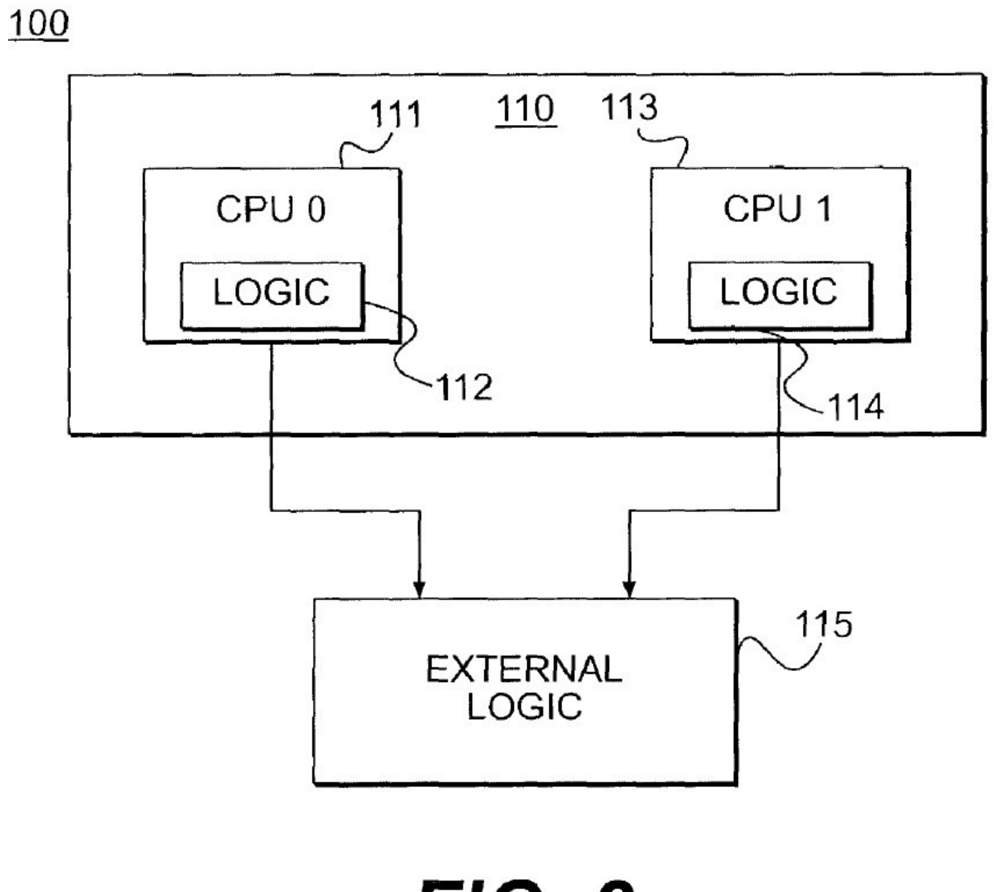


FIG. 2

Safford, Fig. 2.

It was known that detecting and handling such errors before signaling a mismatch error improves processor or system availability. Safford, [0017]; Nguyen, [0046] (detecting an error in the processor core before triggering an FRC reset allows recovery from the error); Kondo, [0100]; Ex. 1002, ¶65.

It was also known that lockstep is non-recoverable from a mismatch error in various situations, for example, a hard error that caused the mismatch (e.g., hardware failure). Bigbee, [0033]; Bossen, 1:35-40 (no recovery is possible for hard errors). Also, lockstep is not recoverable if the processor or FRC logic does not know which processor core was responsible for the mismatch error. Nguyen, [0023]; Safford-181, 6:53-56. As another example, lockstep is not recoverable if the mismatch error is triggered before detection of the error in the individual processor. Nguyen, [0022] (triggering a non-recoverable FRC error before the underlying parity error in the processors is detected). And lockstep is not recoverable if the mismatch error is detected in the absence of detecting an error in the individual processor. Marshall, 2:17-34. Such mismatch error typically would indicate a high risk of corrupt data or incorrect results about to propagate through the system. Kondo, [0089], [0102]; Ex. 1002, ¶¶66-67.

3. Handling Recoverable or Non-Recoverable Errors Differently Was Known

It was known that once an error was detected in FRC processors, the appropriate responsive action depends on whether the error was non-recoverable or recoverable, in order to reduce the impact of errors on the system. Ex. 1002, ¶68.

When a non-recoverable error occurs, there is risk of corrupt data or results propagating through the system. An appropriate responsive action would have been immediately resetting or rebooting the system to return the system to a safe state or shut down the processor. Nguyen, [0007] (FRC errors are detectable but not recoverable; the FRC error handling routine typically resets the system, which is time consuming and reduces system availability); Marshall, 2:3-6 (the only recovery action responsive to an output mismatch is to reset the system); Kondo, [0039], [0190]; Schultz, [0030]; Marisetty, 5:21-28; Ex. 1002, ¶69.

Also, a mismatch error can be non-recoverable because it indicates a non-recoverable hardware failure of the processor (Bigbee, [0033]), for which an appropriate action would have been replacing it with another processor. Klein, 1:19-66 (hot swapping a failed CPU while the system is running); Fox, Abstract, [0005]-[0014]; Arai, 1:39-45, 2:23-41; Ex. 1002, ¶70.

With respect to recoverable errors, in contrast, various recovery routines have been used to reduce the impact of such errors on the system, such as those disclosed in Bigbee, Nguyen, Kondo, and Safford. Kondo, [0191]; Ex. 1002, ¶71.

Treating an otherwise recoverable error as non-recoverable would reduce system availability. Recovery of lockstep from a recoverable error can be implemented with a lower latency as compared to the reset or repair/replacement processes for handling a non-recoverable error. Nguyen, [0022]-[0023], [0040]; Ex. 1002, ¶72.

Conversely, treating a non-recoverable error as a recoverable error would reduce system reliability. For example, it would create risks of having corrupt data or results propagating through the system. Kondo, [0101] (the longer time that passes from error or divergence detection without remedial action, the more serious the impact); Marisetty, 2:38-52; Ex. 1002, ¶73.

4. Using Firmware to Detect and Handle Processor Errors Was Known

It was known that firmware can be used for detecting lockstep errors. Bigbee, [0034] (FRC logic 308 may be implemented as executable firmware); Safford-181, 15:60-64; Ex. 1002, ¶74.

It was also common to use firmware for handling processor errors. Nguyen, [0045] (using PAL or SAL layers of BIOS—firmware—for implementing error recovery mechanisms); Marisetty, 5:48-8:64, Figs. 1-2; Schultz, [0017]-[0018], Figs. 1-3; Bigbee, [0017]-[0025]; Ex. 1002, ¶75.

5. Designating Spare Processors to Boot the System in the Event the Current Boot Processor Suffers a Problem Was Known

It was known to designate a particular processor as the boot processor in a multi-processor system. Natu, 1:5-9, 2:20-36; Goodrum, 2:15-31, 2:40-43, 6:1-3, 6:49-51, 8:60-61; Suffin, 12:3-18, 13:16-20. The designated boot processor is responsible for turning on and testing the other processors or otherwise initializing the system. Goodrum, 2:15-31; Natu, 2:20-36. If a designated boot processor does not function properly, then the overall system may become incapacitated. Goodrum, 2:15-31; Natu, 2:20-50; Ex. 1002, ¶76.

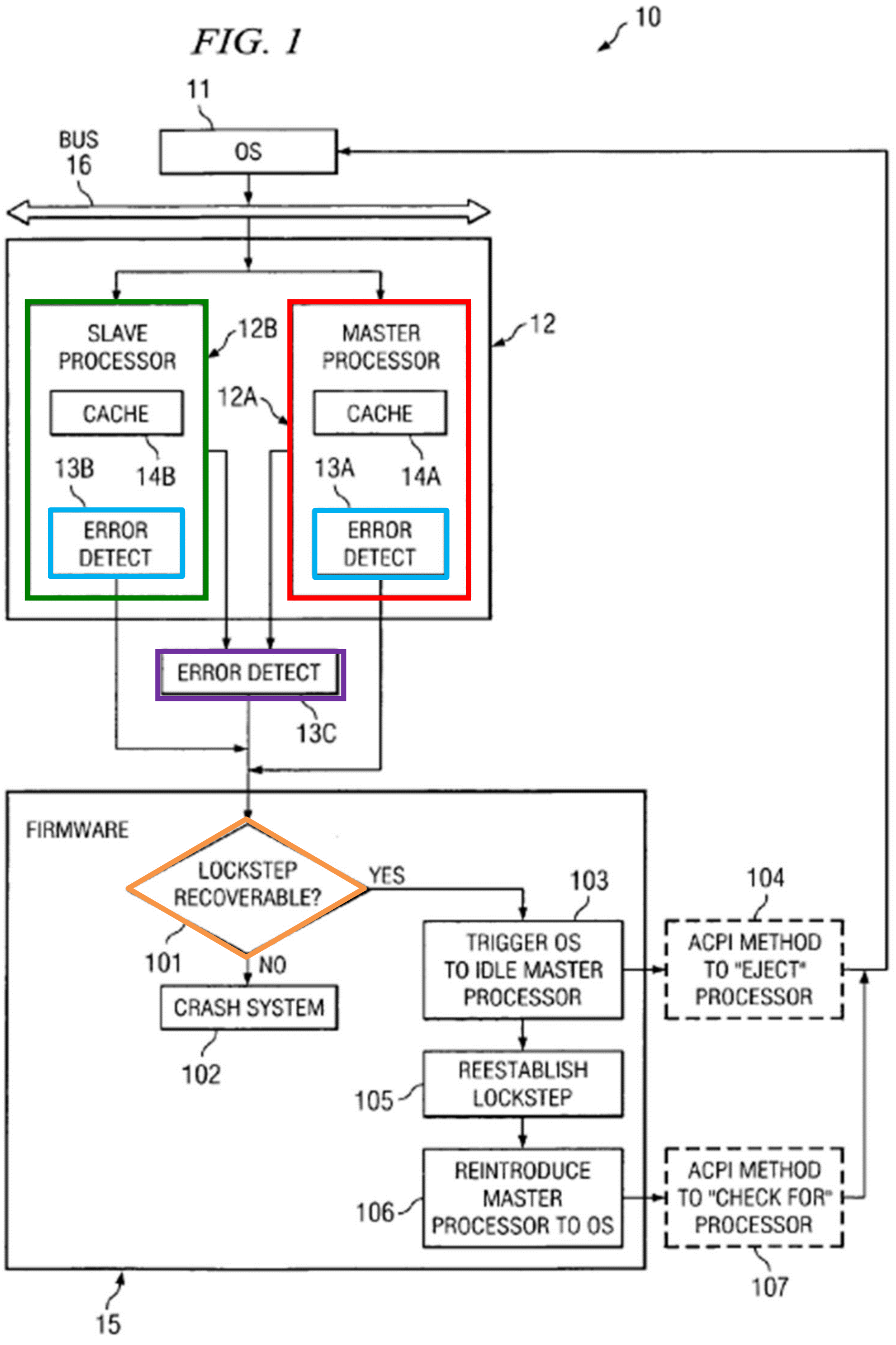
It was therefore known to designate a backup processor to take over boot processor functions should the designated boot processor fail in order to improve the reliability of the multi-processor system. Goodrum, 1:17-20, 2:15-31, 3:7-31, 8:60-64, 9:39-50; Suffin, 2:18-3:11, 13:20-14:2, Fig. 4A; Natu, 1:5-9, 2:20-50, 2:57-3:13. It also beneficially enables the system to diagnose errors with the failed boot processor. Absent a backup boot processor, a reset of the system is typically necessary, which clears the operational status of the system, including operational status information associated with the failed boot processor, hindering diagnosis. Suffin, 1:1-5, 1:12-3:3, 16:9-17:11; Ex. 1002, ¶77.

Use of a backup boot processor also enables an operational state of a functional processor to be copied over to the failed boot processor for recovery. Suffin, 1:1-5, 1:12-3:3, 16:9-17:11. It likewise was known that the operational state of a failed processor may be copied over to a backup processor to facilitate recovery. Safford, [0006], [0018], [0021], Fig. 4; Ex. 1002, ¶78.

B. Brief Description of the Alleged Invention

The '958 Patent claims priority to October 25, 2004. Ex. 1001, p. 1. It discloses a system and method for detecting loss of lockstep (LOL) between pairs of processors and LOL recovery. Ex. 1001, 3:51-59, Figs. 1-6. The system includes a lockstep processor pair, including a master processor and a slave processor, each with error detect logic to detect errors present in the respective processor. *Id.*, 4:14-46, Fig. 1 (reproduced below). The system also includes error detect logic to detect a lockstep mismatch between the master and slave processors. *Id.*, 4:46-55. The specification describes detecting LOL as including at least one of (1) detecting mismatch of the output of the paired processors ("lockstep mismatch error"); and (2) detecting an error in either of the processors that may eventually propagate to a lockstep mismatch error. *Id.*, 2:26-59, 4:36-67. The specification also refers to detecting an error in either of the processors before detecting a lockstep mismatch as a detection of a "precursor to lockstep mismatch." *Id.*, 4:60-67; Ex. 1002, ¶79.

FIG. 1



Ex. 1001, Fig. 1 (annotated).

Once the system detects LOL, it uses firmware to determine whether lockstep is recoverable (or whether the detected LOL is a recoverable LOL); if recoverable, the system's firmware cooperates with the operating system ("OS") via standard OS methods to recover the lockstep. *Id.*, 5:49-6:1, Fig. 1 (step 101). If the OS is Advanced Configuration and Power Interface (ACPI) compatible, its firmware uses ACPI methods to cooperate with the OS to idle the processors, recover lockstep of the processors, and reintroduce the processors to the OS. *Id.*, 6:9-7:21, Fig. 1 (steps 103-107); Ex. 1002, ¶80.

In the event the loss of lockstep is in the boot processor, the system will transfer the role of boot processor to a "hot spare" processor. Ex. 1001, 9:15-29, 10:7-12, 10:26-31, Fig. 3. Loss of lockstep is then recovered in the processor that was the boot processor, and that processor becomes the new hot spare processor. *Id.*; Ex. 1002, ¶80.

C. The Scope and Content of the Prior Art

1. Bigbee (Ex. 1005)

Bigbee discloses a data processing system including hardware 102, firmware 104, and an operating system 106. Bigbee, [0014]-[0025], Fig. 1.

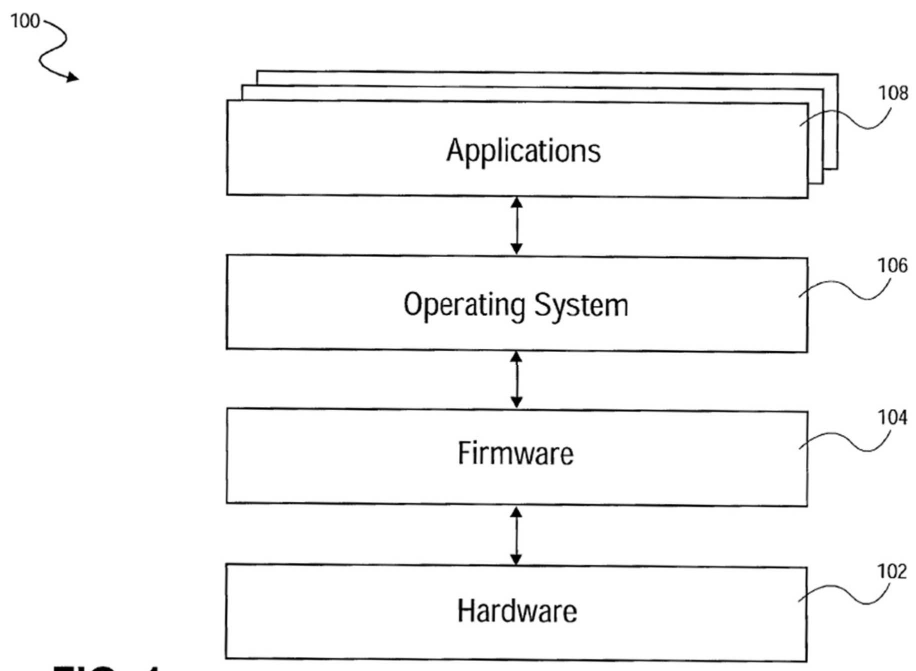


FIG. 1

Bigbee, Fig. 1.

Hardware 102 includes two or more processor packages, each processor package including two processor cores operating in lockstep functional redundancy check (FRC) mode as a single logical processor from the point of view of operating system 106.[2] *Id.*, [0019], [0032]-[0033], Fig. 3 (reproduced below). The two processor cores concurrently execute identical instructions on identical data and each processor generates results. *Id.*, [0002], [0033]. An FRC logic 308 monitors and detects inconsistencies in the results of the two processor cores that indicate an FRC

[2] FRC mode is synonymous with operating in lockstep.

mismatch error, which may be the result of an alpha particle impacting on one or both processor cores. *Id.*, [0033].

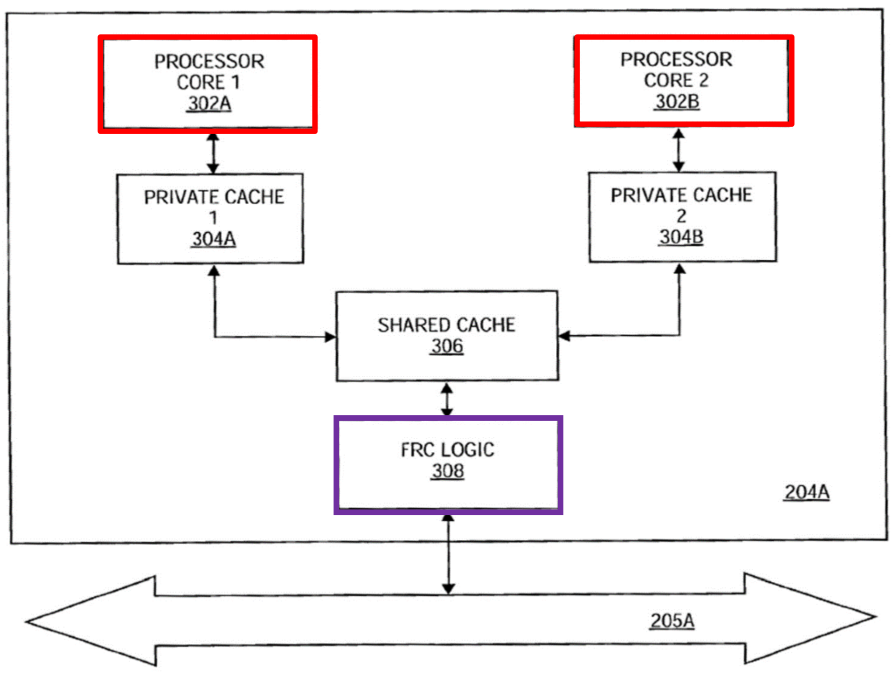


FIG. 3

Bigbee, Fig. 3 (annotated).

The FRC logic also detects an error within each of the processor cores and generates a system machine check to cause the firmware and the operating system to perform an error recovery routine that recovers the FRC operation ("lockstep") of the cores. *Id.*, [0035]-[0037], Figs. 4A-4B; Ex. 1002, ¶¶83-85.

2. Nguyen (Ex. 1006)

Nguyen likewise discloses a processor including two cores 120A and 120B that operate in an FRC mode ("lockstep"), an FRC checker 130 that compares results from the two cores, and an error detector 140 that detects errors in the respective cores. Nguyen, Abstract, [0006], [0020]-[0022], [0039], [0045], Fig. 1. One of the two cores "may be designated as the master core and the other as the slave core." *Id.*, [0047]; Ex. 1002, ¶¶86-90.

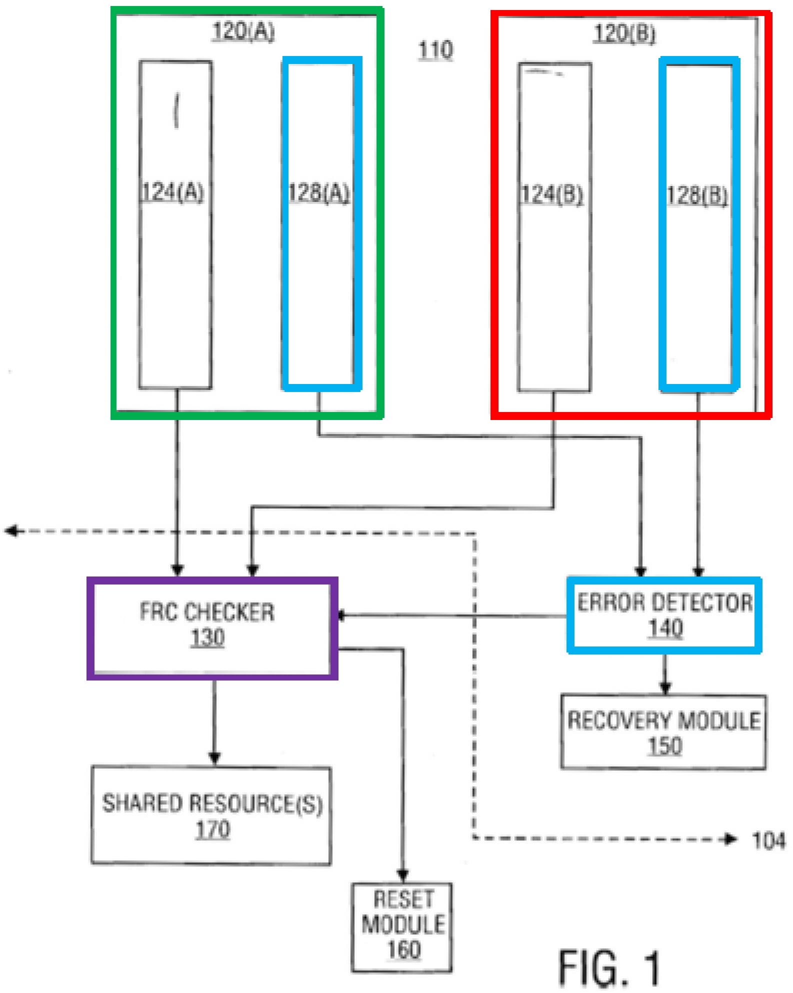


FIG. 1

Nguyen, Fig. 1 (annotated).

Nguyen discloses determining if lockstep is recoverable from a detected error that takes the two cores out of lockstep, such as a soft error or a mismatch error, and mechanisms for handling the detected error based on the determination. *Id.*, [0005], [0007]-[0008], [0022]-[0025], [0040]-[0041], [0056]-[0058].

First, Nguyen discloses lockstep is not recoverable from an FRC error detected by FRC checker 130. *Id.*, [0007], [0022], [0040]. If FRC checker 130 signals an FRC error after detecting a mismatch error before error detector 140 detects the underlying soft error, this FRC error is non-recoverable mismatch error and a reset module 160 resets the system. *Id.*, [0022].

Second, Nguyen discloses lockstep is recoverable from a soft error detected before detecting a mismatch error. *Id.*, [0005], [0022]-[0024], [0046]. If error detector 140 detects a soft error in either processor core (e.g., result from alpha particle collision) before FRC checker 130 detects an FRC error indicating mismatch, error detector 140 disables FRC checker 130 and recovery module 150 implements a recovery routine to restore the cores and FRC mode ("lockstep"). *Id.*

Third, Nguyen discloses lockstep is recoverable from a mismatch error detected before an FRC error is triggered. *Id.*, [0056]-[0058]. As shown below, Nguyen's FRC checker includes a compare unit 734 and a timer unit 738. *Id.* Compare unit 734 compares the data from both cores and sets a status flag to indicate if the comparison yields a match. *Id.* If the data does not match, compare unit 734 sets the status flag to indicate the mismatch and triggers timer unit 738 to begin a countdown interval. *Id.* If error detector 140 receives an error flag before the timeout interval expires, it disables FRC checker 730 and triggers recovery module 150. *Id.* But if the timeout interval does expire, the FRC checker signals a non-recoverable

FRC error and triggers reset module 160 to reset the system. *Id.*, [0022], [0056]-[0058].

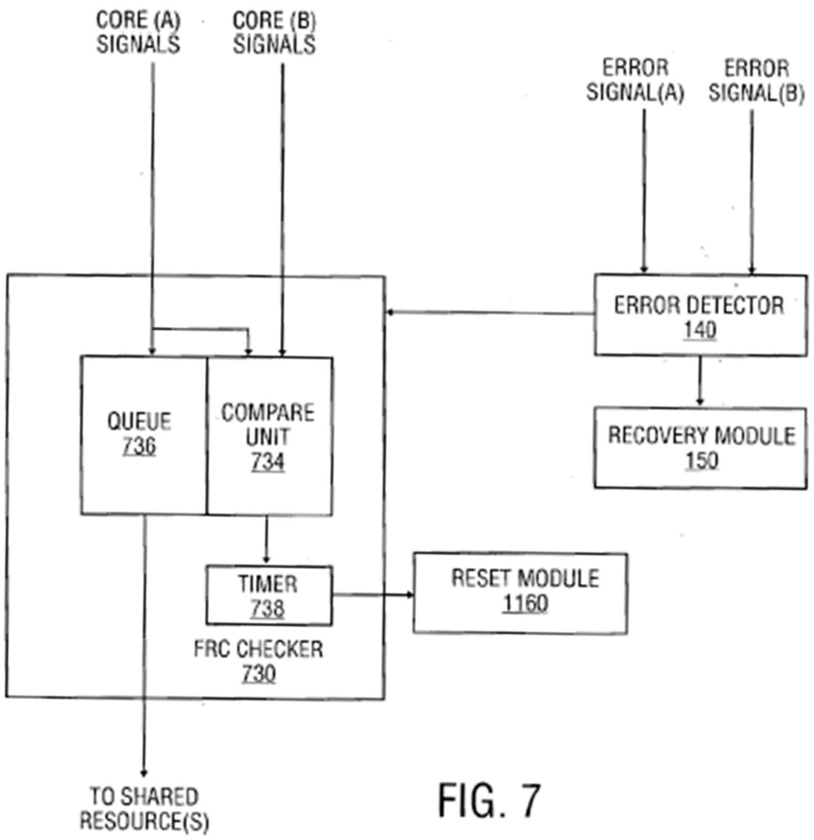


FIG. 7

Nguyen, Fig. 7.

Nguyen's process for determining if lockstep is recoverable from an error in a lockstep processor pair was known in the prior art. *See e.g.*, Tu (Ex. 1017), [0041]-[0043], Fig. 5.

3. Suffin (Ex. 1020)

Suffin, like Bigbee and Nguyen, discloses a computer system having multiple redundant CPUs and associated controllers, operating in lockstep. Suffin, 6:1-7:14,

Fig. 2. When a failure occurs in a processor, such as from a loss of lockstep (*Id.*, 7:12-14), Suffin discusses using an ordered list of processors (a "boot list") to determine which processor to use as a boot processor. *Id.*, 2:5-4:5, 11:3-9, 11:23-12:18, 13:1-7, 13:16-14:10, Fig. 4A. If a boot using a first processor is unsuccessful, the system selects the next processor on the list to be the boot processor. *Id.* When recovering from a system failure, if a processor is suspended or not operating, that processor likewise is skipped on the boot list, so that the next processor on the list is used for rebooting. *Id.*, 16:20-17:11, 13:1-7. Further, the state of the processor that failed may be copied for analysis. *Id.*; Ex. 1002, ¶¶94-96.

The swapping of the role of boot processor from one processor to another is shown, for instance, in Figure 4A, below.

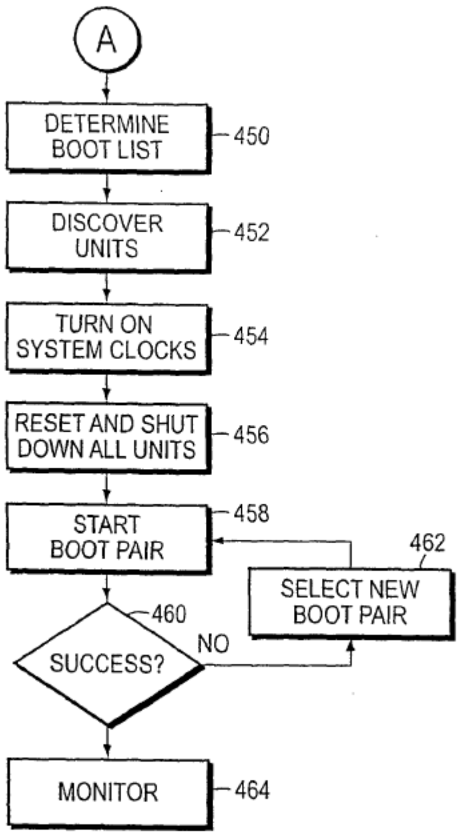


FIG. 4A

Suffin, Fig. 4A.

The system determines the boot list in step 450, then determines which processors on the boot list are available (skipping those that are not) in step 452, then uses the first available processor on the list to boot in step 458, and if that is not successful selects another processor as the boot processor in step 462. *Id.*, 11:23-12:18, 13:1-14:6, 14:15-15:3, 16:9-17:11, claims 21, 30, Fig. 4A.

4. Goodrum (Ex. 1021)

Goodrum, like Suffin, discloses a multiple processor system and methods for switching the role of boot processor to another processor in the event of a failure of the boot processor. For example, "a hot spare boot circuit … automatically reassigns the power up responsibilities to an operational second processor should the primary processor fail." Goodrum, 3:10-13; 8:60-64; 9:39-50; 18:34-49, claims 1, 4, 7, 10, Fig. 2; Ex. 1002, ¶97.

5. Safford (Ex. 1007)

Safford, like Bigbee and Suffin, relates to a multi-processor system with processors operating in lockstep. Safford, Abstract, [0001], [0006]-[0007], [0015]-[0016], [0018]-[0021], claim 1, Figs. 2-4. In order to "enhance recovery from loss of lockstep," Safford copies the state of one or more of the processors in a processor pair for which a loss of lockstep has been detected to a spare processing unit, which becomes an active processor unit in the computer system. *Id.*, [0006]-[0007], [0021]. This process is shown, for instance, in Figure 4, step 220:

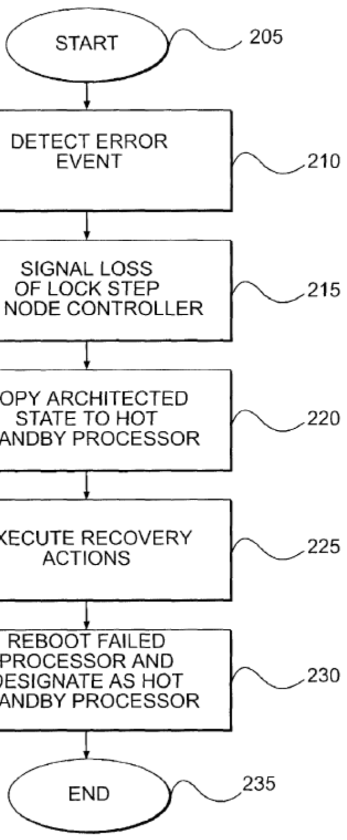

START
205
DETECT ERROR EVENT
210
SIGNAL LOSS OF LOCK STEP TO NODE CONTROLLER
215
COPY ARCHITECTED STATE TO HOT STANDBY PROCESSOR
220
EXECUTE RECOVERY ACTIONS
225
REBOOT FAILED PROCESSOR AND DESIGNATE AS HOT STANDBY PROCESSOR
230
END
235

FIG. 4

Safford, Fig. 4, [0021].

Once the processor with the loss of lockstep is idled and lockstep is recovered, then that processor (whose state was copied to the standby processor) becomes the new hot standby processor, as seen in step 230. *Id.*; Ex. 1002, ¶¶98-99.

III. IDENTIFICATION OF CHALLENGE PURSUANT TO 37 C.F.R. § 42.104(B) AND STATEMENT OF THE RELIEF REQUESTED

Petitioner requests *inter partes* review of claims 9-12 and 23-25 on these grounds:

Ground	Prior Art[3]	Claims Challenged
A	Obvious under 35 U.S.C. § 103 over Bigbee, Nguyen, and Goodrum	9-12
B	Obvious under 35 U.S.C. § 103 over Bigbee, Nguyen, and Suffin	9-12
C	Obvious under 35 U.S.C. § 103 over Bigbee, Nguyen, Suffin and Safford	23-25, 12
D	Obvious under 35 U.S.C. § 103 over Bigbee, Nguyen, Goodrum and Safford	23-25, 12

This petition does not present grounds or substantially the same arguments presented during prosecution.

A. Level of Ordinary Skill in the Art

A person of ordinary skill in the art ("POSITA") would have at least a bachelor's degree in Computer Science, Computer Engineering, or equivalent, and at least two years of prior work experience with computer architecture design, including fault tolerant computer architecture design, as of the earliest priority date of the '958 Patent—October 25, 2004. Ex. 1002, ¶¶101-104. Additional education could substitute for professional experience and vice versa. *Id.*

[3] All references to 35 U.S.C. § 102 and § 103 refer to pre-AIA statutes.

B. How the Challenged Claims Are to Be Construed

The Board construes claims per *Phillips v. AWH Corp.*, 415 F.3d 1303 (Fed. Cir. 2005) (en banc). Under *Phillips*, claims terms are given their ordinary and customary meaning as understood by a POSITA in the context of the specification. *Id.*, 1312-13.

Aside from the terms addressed below, all terms are given this ordinary and customary meaning. Ex. 1002, ¶¶105-113.

1. "Detection of Loss of Lockstep" and "Detecting Loss of Lockstep"

For this IPR, the terms "detection of loss of lockstep" and "detecting loss of lockstep" should be construed to include both mismatch errors and errors that may eventually result in mismatch errors constituting a "precursor to lockstep mismatch"—including at least one of (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error. Ex. 1001, 4:56-67.

The specification provides detecting these two types of errors as examples of detection of LOL. *See e.g.*, Ex. 1001, 2:26-59, 4:37-67. It states that "[l]ockstep mismatch is one way of detecting a LOL." Ex. 1001, 4:56-57. It also states that "[b]ecause the detection of LOL … may occur before an actual lockstep mismatch occurs, the detection of LOL … may be referred to as a detection of a 'precursor to lockstep mismatch.'" Ex. 1001, 4:60-64. Both lockstep mismatch and precursors to lockstep mismatch are detection of loss of lockstep in the specification of the ’958 Patent because, as the specification explains, 1) mismatch errors indicate that lockstep is already lost and 2) precursors to lockstep mismatch indicate that the “error may eventually propagate to a lockstep mismatch error.” Ex. 1001, 4:56-67.

This is consistent with the dependent claims, which expressly define detecting loss of lockstep as either of the two options. Ex. 1001, claims 1, 4-8. Claim 4, for example, further defines “detecting loss of lockstep” as comprising “detecting corrupt data … ***that would lead to a lockstep mismatch***.” Similarly, claim 5 defines “detecting loss of lockstep” as comprising “***detecting a precursor to lockstep mismatch***.” Both claims 4 and 5 expressly require that a precursor to lockstep mismatch is included in the definition of “detecting loss of lockstep.” Claim 6 further defines “detecting loss of lockstep” as “detecting lockstep mismatch between the lockstep pair of processors,” expressly requiring that lockstep mismatch is included.

The term must be interpreted as broad enough to encompass both dependent claim definitions—including at least one of (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error.

2. “Lockstep Is Recoverable” and “Loss of Lockstep Is Recoverable”

The terms “lockstep is recoverable” and “loss of lockstep is recoverable” should be construed as “lockstep is recoverable from the detected loss of lockstep” because the specification only describes lockstep recovery from the detected LOL—“whether the detected LOL is a recoverable LOL.” *See e.g.*, Ex. 1001, 5:56-6:1.

3. Optional Conditional Limitations in Method Claims Are Not Limiting

Limitations [23D], [23E], and claims 10-12, 24, and 25 are optional conditional limitations that need not be performed in order to perform the claimed method and, thus, are not limiting. *See Ex Parte Schulhauser*, Appeal No. 2013-007847 (P.T.A.B. Apr. 28, 2016), pp. 8-10; *Marvell Semiconductor, Inc. v. Uniloc 2017 LLC,* IPR2019-01350, Paper 24, pp. 36-39 (P.T.A.B. Feb. 1, 2021).

IV. GROUND A: CLAIMS 9-12 ARE OBVIOUS OVER BIGBEE-NGUYEN-GOODRUM

Claims 9-12 depend from claim 1 (directly or through other claims). As explained below, the elements of claim 1 are taught by the combination of Bigbee-Nguyen. The additional elements of claims 9-12 are taught by Goodrum and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Goodrum. Below is an overview of the Bigbee-Nguyen combination that teaches the elements of claim 1. Ex. 1002, ¶¶114-115.

A. Overview of the Bigbee-Nguyen Combination that Teaches Claim 1

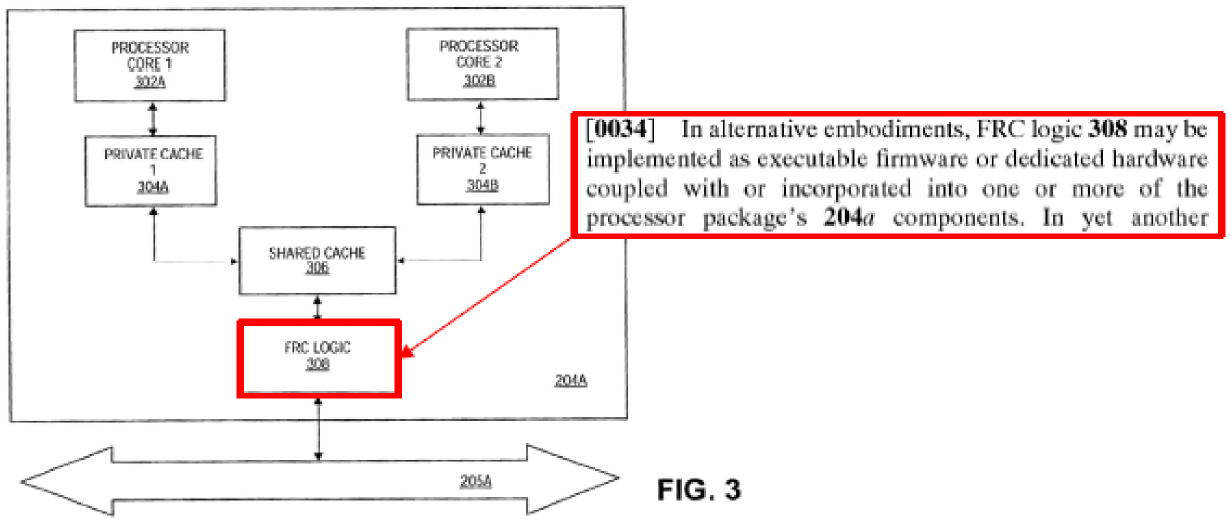
Bigbee discloses all limitations of claim 1 except “determin[ing] if lockstep is recoverable.” Nguyen discloses a method of determining if the lockstep of the processor pair is recoverable or non-recoverable. Bigbee’s firmware performing

Nguyen's determining if the lockstep is recoverable or non-recoverable renders claim 1 obvious. Ex. 1002, ¶¶116-117.

Applying the combination of Bigbee and Nguyen to the core components of claim 1, each limitation is expressly disclosed in the combination Ex. 1002, ¶118:

Core Claim Requirement	**Prior Art Reference**
1) Detecting loss of lockstep between a pair of lockstep processors	Bigbee
2) With firmware, determining if lockstep is recoverable	Bigbee (with firmware) Nguyen (determining if lockstep is recoverable)
3) With firmware, determining if lockstep mismatch has occurred	Bigbee
4) Triggering an operating system to idle the processors	Bigbee
5) Attempting to recover lockstep	Bigbee
6) If recovered, trigger the operating system to recognize the processors to be available	Bigbee

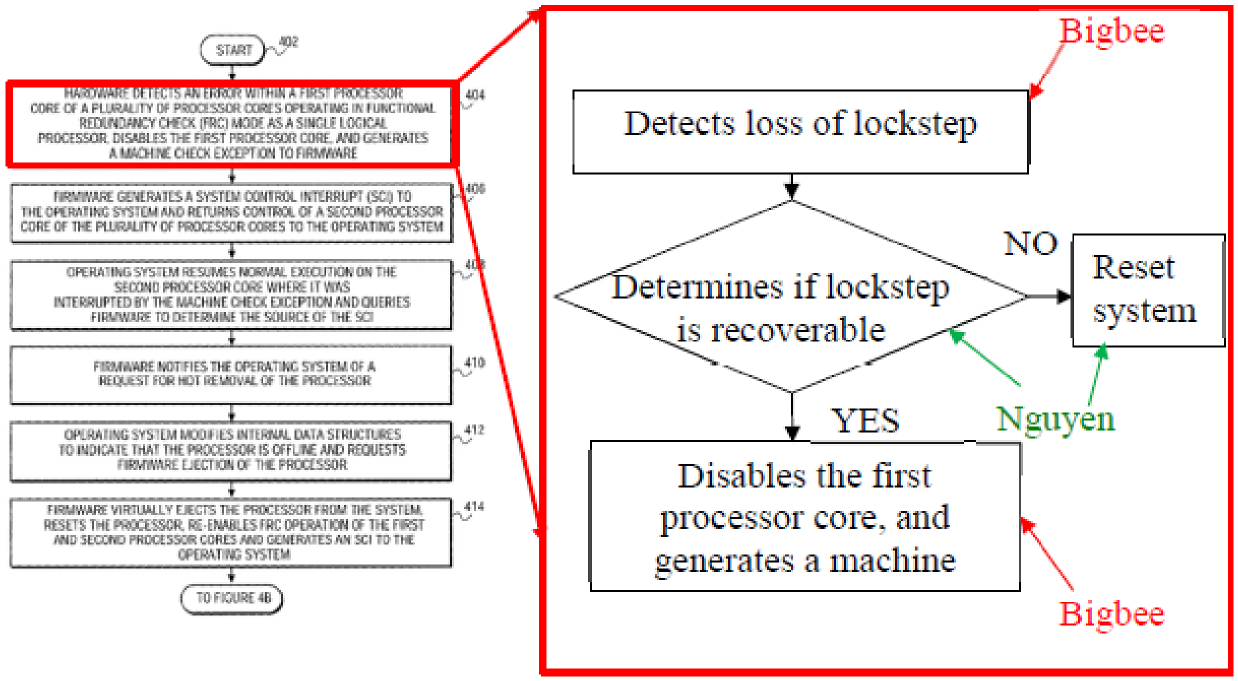
The combined system, when implementing Nguyen's express disclosure of determining whether loss of lockstep is recoverable, in Bigbee's firmware (FRC Logic) responsive to Bigbee's detection of loss of lockstep would be as follows:



[0034] In alternative embodiments, FRC logic **308** may be implemented as executable firmware or dedicated hardware coupled with or incorporated into one or more of the processor package's **204***a* components. In yet another

FIG. 3

Ex. 1002, ¶119 (annotating Bigbee, Fig. 3).


START
402
HARDWARE DETECTS AN ERROR WITHIN A FIRST PROCESSOR CORE OF A PLURALITY OF PROCESSOR CORES OPERATING IN FUNCTIONAL REDUNDANCY CHECK (FRC) MODE AS A SINGLE LOGICAL PROCESSOR, DISABLES THE FIRST PROCESSOR CORE, AND GENERATES A MACHINE CHECK EXCEPTION TO FIRMWARE
404
FIRMWARE GENERATES A SYSTEM CONTROL INTERRUPT (SCI) TO THE OPERATING SYSTEM AND RETURNS CONTROL OF A SECOND PROCESSOR CORE OF THE PLURALITY OF PROCESSOR CORES TO THE OPERATING SYSTEM
406
OPERATING SYSTEM RESUMES NORMAL EXECUTION ON THE SECOND PROCESSOR CORE WHERE IT WAS INTERRUPTED BY THE MACHINE CHECK EXCEPTION AND QUERIES FIRMWARE TO DETERMINE THE SOURCE OF THE SCI
408
FIRMWARE NOTIFIES THE OPERATING SYSTEM OF A REQUEST FOR HOT REMOVAL OF THE PROCESSOR
410
OPERATING SYSTEM MODIFIES INTERNAL DATA STRUCTURES TO INDICATE THAT THE PROCESSOR IS OFFLINE AND REQUESTS FIRMWARE EJECTION OF THE PROCESSOR
412
FIRMWARE VIRTUALLY EJECTS THE PROCESSOR FROM THE SYSTEM, RESETS THE PROCESSOR, RE-ENABLES FRC OPERATION OF THE FIRST AND SECOND PROCESSOR CORES AND GENERATES AN SCI TO THE OPERATING SYSTEM
414
TO FIGURE 4B
Bigbee
Detects loss of lockstep
Determines if lockstep is recoverable
NO
Reset system
YES
Nguyen
Disables the first processor core, and generates a machine
Bigbee

Ex. 1002, ¶119 (annotating and modifying Bigbee, Fig. 4A).

B. A POSITA Would Have Found It Obvious to Combine Bigbee and Nguyen

A POSITA would have found it obvious to implement Nguyen's determination of whether the detected error causing the loss of lockstep is recoverable by Bigbee's firmware FRC logic to create a system that can handle both recoverable and non-recoverable lockstep errors. Ex. 1002, ¶¶120-126.

1. A POSITA would have been motivated to implement Nguyen's determining whether lockstep is recoverable from the detected error in Bigbee

A POSITA would have been motivated to implement Nguyen's determination of whether the detected error causing the loss of lockstep is recoverable by Bigbee's firmware FRC logic to appropriately handle the detected error. Because different responsive actions should be taken to handle each of non-recoverable and recoverable errors, determining whether the error is recoverable or non-recoverable is critical to reduce the impact of errors on the system. *See* Ex. 1002, ¶¶127, 63-73; Marisetty, 4:43-5:63, 7:24-27 (after detection of an error, the processor determines the severity of the error and takes action depending on the severity); Kondo, [0100] (determining whether an error is a) fatal to the processor or b) a correctable error that will produce a divergence but will not compromise data integrity); Schultz, [0027], [0154] (firmware SAL reports the appropriate error severity as being fatal or recoverable); [0021] (errors detected are classified according to the error type and severity).

Dr. Baker explained (Ex. 1002, ¶¶68-73, 128) that non-recoverable errors cause a risk of data corruption or propagating an incorrect result if the system is not immediately rebooted or reset to return to a safe state. Dr. Baker further explained (Ex. 1002, ¶¶68-73, 129) that recoverable errors can be recovered using known recovery routines that reduce the impact on the system. A POSITA would have understood the importance of handling recoverable errors differently from non-recoverable errors. Ex. 1002, ¶¶128-129.

Handling recoverable errors differently from non-recoverable errors also prevents loss of system availability. Ex. 1002, ¶129 (citing Nguyen, [0007]-[0008], [0022]-[0023], [0040]). Nguyen teaches that treating a recoverable error as a non-recoverable error would reduce system availability, i.e., increase system downtime. Nguyen, [0007]-[0008], [0022]-[0023], [0040]. Recovery of lockstep from a recoverable error can be implemented with lower latency compared to the reset or repair/replacement processes for handling a non- recoverable error. *Id.*

Handling non-recoverable errors differently from recoverable errors also prevents reduction in system reliability. Ex. 1002, ¶130. For example, handling non-recoverable errors as recoverable creates risks of having corrupt data or incorrect results propagating through the system. *Id.* (citing Kondo, [0101], [0103] (a nonrecoverable error involves the risk of data corruption); Marisetty, 2:38-52; Safford, [0003].

Reducing system availability or reliability by handling errors incorrectly—either restarting the entire system when an error can be recovered or attempting to recover a non-recoverable error and causing error propagation—would have been undesirable for fault-tolerant multiprocessor systems, particularly for systems that need to have high reliability and availability, such as server systems (Bigbee, [0030]-[0031]), resources in a network (Kondo, [0191]), electronic fund transfer systems or airline traffic control systems (Klein, 1:19-24). Ex. 1002, ¶131.

Accordingly, a POSITA would have understood that fault tolerant systems would distinguish between recoverable and non-recoverable errors by first determining whether an error is recoverable and then responding accordingly based on that determination. Ex. 1002, ¶132.

A POSITA, who would have had knowledge of handling recoverable and non-recoverable errors detected in FRC processors, would have been motivated to use Bigbee's FRC logic to determine whether lockstep is recoverable and take the appropriate action responsive to the recoverability of the error, thereby reducing the impact of the error and improving system reliability and availability. Ex. 1002, ¶133.

2. Reasonable Expectation of Success

A POSITA would have understood how to implement Nguyen's determining whether a detected error is recoverable or non-recoverable using Bigbee's firmware FRC logic with a reasonable expectation of success because implementing logical steps in firmware was known in the art. Ex. 1002, ¶134 (citing Bigbee, [0034], Safford-181, 15:60-64 (lockstep logic may be implemented in firmware)).

A POSITA would have expected success in implementing Nguyen's determining step in Bigbee's firmware because Bigbee's system is compliant with well-known industry standards—its operating system is ACPI-compliant, and its firmware includes ACPI BIOS code and a processor abstraction level (PAL) that provides an abstraction of implementation-specific processor features. Ex. 1002, ¶135; Bigbee, [0016]-[0018].

A POSITA would have known how to use ACPI Source Language (ASL) to program Bigbee's firmware code to implement Nguyen's determining step using ordinary computer programming skill because using such firmware, such as the PAL or SAL layer of ACPI BIOS, for implementing error diagnosis and recovery routines was well known, involving only ordinary skill and common knowledge. Ex. 1002, ¶136 (citing Schultz, [0027], [0154] (firmware SAL reports the appropriate error severity as being fatal or recoverable); Nguyen, [0045] (using PAL or SAL layers of BIOS for implementing error recovery mechanisms); Marisetty, 5:48-8:64 (teaching using firmware including PAL and SAL to execute error handling routines), Figs. 1-2; Schultz, Figs. 1-3, [0017]-[0018] (teaching using firmware including PAL and SAL for correcting an error that is not correctable directly by hardware)).

Implementing Nguyen's determining step in firmware was also known to be one of only three available alternatives—firmware, hardware, or combination—all of which would have been obvious to try, and any of which would be a suitable option. Ex. 1002, ¶137; *Intel Corp. v. PACT XPP Schweiz AG*, 61 F.4th 1373, 1380-81 (Fed. Cir. 2023).

The benefits of implementing Nguyen's determination in firmware would have been understood by a POSITA, because firmware is flexible and so can be modified after production, updated, and supported relatively quickly and inexpensively. Hardware, on the other hand, is inflexible. As noted by Dr. Baker, once manufactured, hardware cannot be easily modified, and any modification or revision is often expensive because it requires remanufacture of new components. Ex. 1002, ¶138.

Therefore, implementing Nguyen's determining step in Bigbee's firmware would have been both desirable and readily achievable by a POSITA. Ex. 1002, ¶¶134-138, 74-75.

C. Independent Claim 1

1. Limitation [1A]: A method comprising: detecting loss of lockstep for a lockstep pair of processors [4]

Bigbee teaches element [1A]. Bigbee's FRC logic detects loss of lockstep between the pair of lockstep processor cores and further discloses implementing the FRC logic by executable firmware. Ex. 1002, ¶¶139-140; Bigbee, [0019], [0034].

As shown below, Bigbee discloses a processor package including two processor cores, each with a private cache, which are coupled to a shared cache. Bigbee, Fig. 3, [0032]. The two processor cores function as a single logical processor and operate in an FRC mode ("lockstep pair of processors"), executing identical instructions on identical data to generate results for storage in associated private cache and/or shared cache. Bigbee, [0033]. Bigbee discloses "detecting loss of lockstep for a lockstep pair of processors" because FRC logic detects both inconsistencies in the generated results that indicate an FRC mismatch error and errors within the processor cores of a processor package that cause a break in the lockstep between the cores, disabling FRC mode operation. Bigbee, [0019], [0033]; Ex. 1002, ¶141.

[4] These limitations appear and are labelled in the Claims Listing Appendix.

Bigbee further discloses implementing its FRC logic by executable firmware, as was known in the art. Bigbee, [0034]; Ex. 1002, ¶141 (citing Safford-181, 15:60-64).

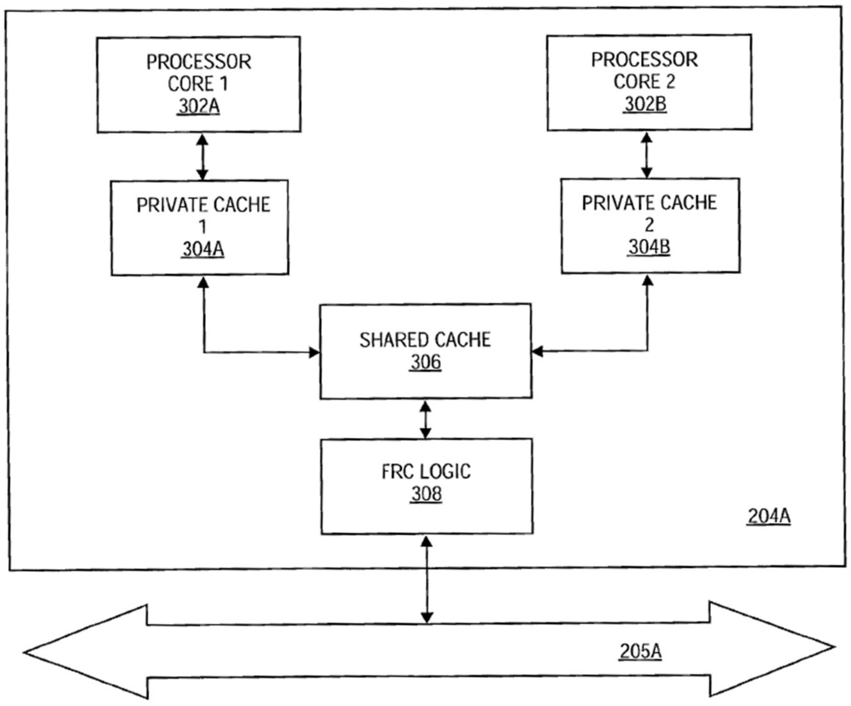


FIG. 3

Bigbee, Fig. 3.

2. Limitation [1B]: 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

Bigbee in view of Nguyen renders obvious [1B]. Ex. 1002, ¶¶142-147.

As explained above, a POSITA would have incorporated Nguyen's determining if the lockstep is recoverable into Bigbee to take an appropriate action responsive to the severity of the error detected in the lockstep processor pair.

Sections IV.A-B, *supra*. Bigbee in view of Nguyen further discloses that, responsive to determining if lockstep is recoverable from the detected error, the firmware triggers the operating system to idle the processors and recover lockstep between the pair of processors. Ex. 1002, ¶143.

First, Bigbee discloses "using firmware to trigger an operating system to idle the lockstep pair of processors." Ex. 1002, ¶144. As shown in Figure 4A below, Bigbee discloses that firmware (e.g., SAL program code) generates a system control interrupt (SCI) to the operating system. Bigbee, [0035], Fig. 4A (block 406). In response to the SCI, the operating system queries firmware (ACPI BIOS) to determine the source of the SCI; firmware responds to the query by notifying the operating system of a request for processor hot removal (e.g., by a ACPI Notify command).[5] Bigbee, [0036], Fig. 4A (blocks 408-410) ("trigger an operating system"). The operating system then modifies its internal data structures to indicate that the processor is offline and requests firmware ejection of the processor (e.g., by executing an _EJ$_X$ control method ("to idle the lockstep pair of processors"). Bigbee, [0036]-[0037], Fig. 4A (block 412); Ex. 1022, p. 162.

[5] The term "processor" used in Bigbee's error recovering routine refers to the single logical processor that includes the pair of processor cores operating in lockstep. Ex. 1002, ¶144, n.4 (citing Bigbee, [0032], [0035]-[0038]).

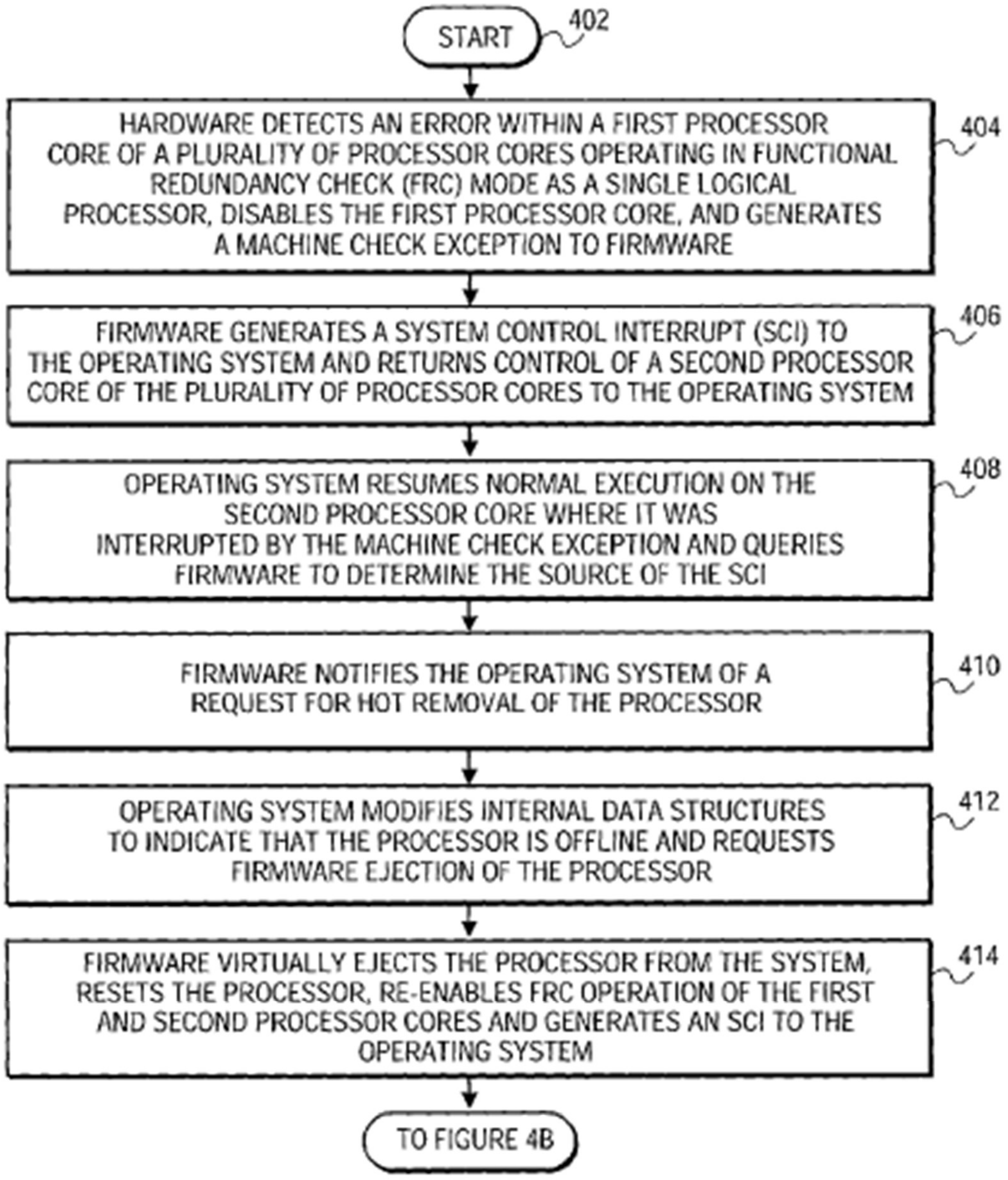

START
402
HARDWARE DETECTS AN ERROR WITHIN A FIRST PROCESSOR CORE OF A PLURALITY OF PROCESSOR CORES OPERATING IN FUNCTIONAL REDUNDANCY CHECK (FRC) MODE AS A SINGLE LOGICAL PROCESSOR, DISABLES THE FIRST PROCESSOR CORE, AND GENERATES A MACHINE CHECK EXCEPTION TO FIRMWARE
404
FIRMWARE GENERATES A SYSTEM CONTROL INTERRUPT (SCI) TO THE OPERATING SYSTEM AND RETURNS CONTROL OF A SECOND PROCESSOR CORE OF THE PLURALITY OF PROCESSOR CORES TO THE OPERATING SYSTEM
406
OPERATING SYSTEM RESUMES NORMAL EXECUTION ON THE SECOND PROCESSOR CORE WHERE IT WAS INTERRUPTED BY THE MACHINE CHECK EXCEPTION AND QUERIES FIRMWARE TO DETERMINE THE SOURCE OF THE SCI
408
FIRMWARE NOTIFIES THE OPERATING SYSTEM OF A REQUEST FOR HOT REMOVAL OF THE PROCESSOR
410
OPERATING SYSTEM MODIFIES INTERNAL DATA STRUCTURES TO INDICATE THAT THE PROCESSOR IS OFFLINE AND REQUESTS FIRMWARE EJECTION OF THE PROCESSOR
412
FIRMWARE VIRTUALLY EJECTS THE PROCESSOR FROM THE SYSTEM, RESETS THE PROCESSOR, RE-ENABLES FRC OPERATION OF THE FIRST AND SECOND PROCESSOR CORES AND GENERATES AN SCI TO THE OPERATING SYSTEM
414
TO FIGURE 4B

FIG. 4A

Bigbee, Fig. 4A.

Second, Bigbee discloses "using firmware to … recover lockstep for the lockstep pair of processors." Ex. 1002, ¶145. In response to the operating system's ejection request, Bigbee's firmware virtually ejects the processor from the system, resets the processor, re-enables FRC operation of the processor cores ("recover lockstep for the lockstep pair of processors") and generates an SCI to the operating system. *Id.*; Bigbee, [0037], Fig. 4A (block 414).

The '958 Patent similarly discloses idling the processors, by having the firmware use an ACPI method to "eject" the processors, thereby triggering the operating system to idle the master processor, resets the processor pair, and recover lockstep. Ex. 1001, 6:9-17, 6:40-42, 6:55-57; Ex. 1002, ¶146.

Bigbee discloses that its firmware generates an SCI to the operating system after recovering lockstep between the pair of processors. Bigbee, [0037], Fig. 4A (block 414). The operating system queries firmware (e.g., ACPI BIOS) to determine the source of the SCI, and the firmware responds to the query by notifying the operating system of a request for processor hot insertion ("trigger the operating system"). Bigbee, [0037], Fig. 4B (block 416-418). The firmware also provides the operating system with status and resource data of the processor to be inserted via one or more ACPI methods (e.g., _STA, _MAT, and _CRS). Bigbee, [0037], Fig. 4B (block 422). The operating system then requests insertion of the processor by calling an ACPI method (e.g., _PS0), which the firmware virtually executes to insert the processor into the system. Bigbee, [0037], Fig. 4B (block 424-426). Once the processor has been inserted, the operating system modifies its internal data structures to indicate that the processor is online and available ("recognize the lockstep pair of processors having recovered lockstep”). Bigbee, [0037], Fig. 4B (e.g., block 428); Ex. 1002, ¶147.

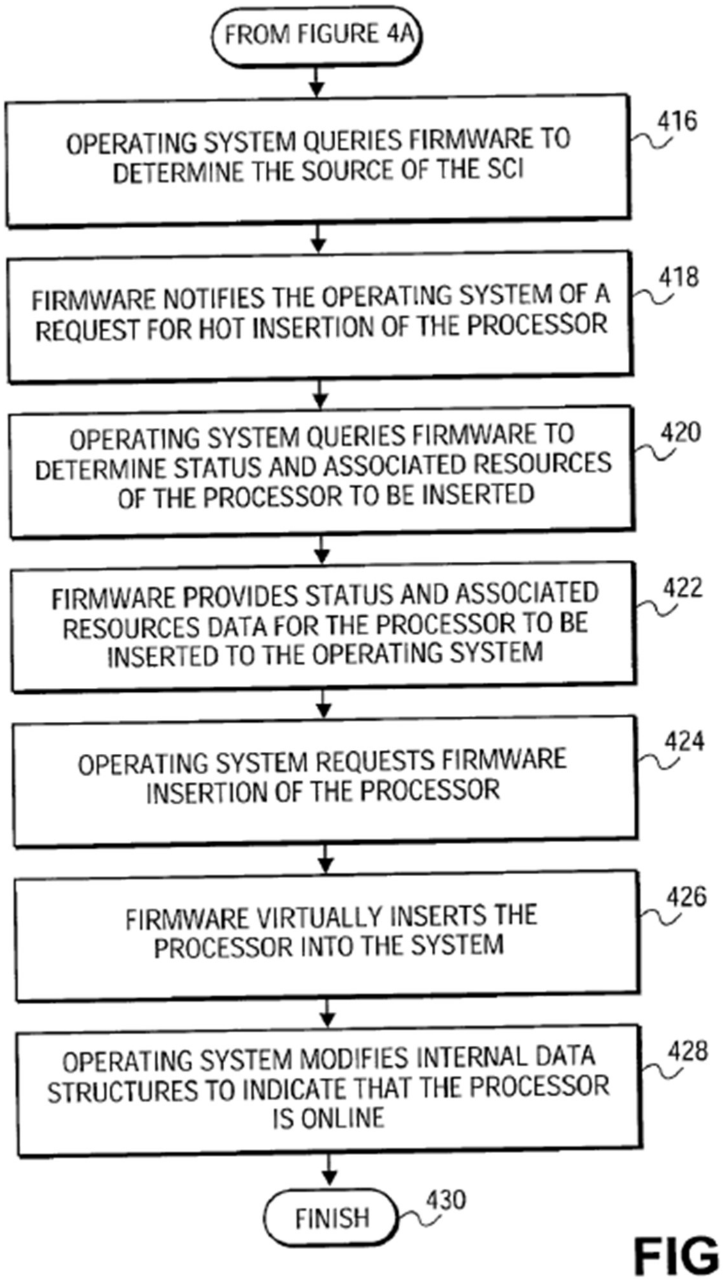

FROM FIGURE 4A
OPERATING SYSTEM QUERIES FIRMWARE TO DETERMINE THE SOURCE OF THE SCI
416
FIRMWARE NOTIFIES THE OPERATING SYSTEM OF A REQUEST FOR HOT INSERTION OF THE PROCESSOR
418
OPERATING SYSTEM QUERIES FIRMWARE TO DETERMINE STATUS AND ASSOCIATED RESOURCES OF THE PROCESSOR TO BE INSERTED
420
FIRMWARE PROVIDES STATUS AND ASSOCIATED RESOURCES DATA FOR THE PROCESSOR TO BE INSERTED TO THE OPERATING SYSTEM
422
OPERATING SYSTEM REQUESTS FIRMWARE INSERTION OF THE PROCESSOR
424
FIRMWARE VIRTUALLY INSERTS THE PROCESSOR INTO THE SYSTEM
426
OPERATING SYSTEM MODIFIES INTERNAL DATA STRUCTURES TO INDICATE THAT THE PROCESSOR IS ONLINE
428
FINISH
430

FIG. 4B

Bigbee, Fig. 4B.

The ’958 Patent similarly discloses that the firmware provides the operating system with status of the processors using an ACPI method (e.g., _STA), and uses an ACPI method to trigger the operating system to recognize the processors as available and as having recovered lockstep. Ex. 1001, 6:55-7:12; Ex. 1002, ¶148.

3. Limitation [1C]: 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.

Bigbee-Nguyen renders obvious element [1C]. As explained for element [1A], Bigbee detects a loss of lockstep using firmware. Section IV.C.1. Nguyen discloses determining if lockstep is recoverable from lockstep errors, as was also known in the art. Ex. 1002, ¶¶150-151. Nguyen discloses responsive to the detection of loss of lockstep, the FRC checker and error detector operating together determine whether lockstep is recoverable from the detected error. Nguyen, [0022]-[0025], [0030], [0037], [0039]-[0041], [0046], [0056]-[0057]; Ex. 1002, ¶¶149-156.

Nguyen's FRC checker and error detector determine lockstep is recoverable from the detected error in two instances and non-recoverable from the detected error in one instance. Ex. 1002, ¶¶152. ***First***, Nguyen's FRC checker and error detector determine a mismatch error that cannot be traced to a processor containing an underlying soft error is not recoverable. Nguyen, [0007], [0023]. ***Second***, Nguyen's FRC checker and error detector determine that lockstep is recoverable from a soft error detected by the error detector before detecting a mismatch error. Nguyen, [0005], [0022]-[0024], [0046]. ***Third***, Nguyen's FRC checker and error detector determine lockstep is recoverable from a mismatch error if an error in the processors is detected within a timeout interval from the detection of the mismatch error. Nguyen, [0023], [0030]; Ex. 1002, ¶¶152-153; Section II.C.2.

Nguyen further discloses "determining if a lockstep mismatch has occurred" because Nguyen determines if the lockstep is recoverable after detecting a mismatch error. Ex. 1002, ¶¶154-155. For example, in Nguyen's determining if lockstep is recoverable: (1) Nguyen's FRC checker's compare unit compares the data from both cores, and it sets a status flag to indicate whether the data matches or mismatches ("determining if a lockstep mismatch has occurred"); and (2) determining if Nguyen's error detector detects an error in the processors before (recoverable) or after (non-recoverable) the timeout interval of the timer unit expires. Ex. 1002, ¶155; Nguyen, [0056]-[0057]; Section II.C.2.

As explained in Section IV.B, a POSITA would have been motivated to incorporate Nguyen's determination of if lockstep is recoverable in Bigbee's FRC logic to increase system fault tolerance and availability and would have had a reasonable expectation of success in doing so. Ex. 1002, ¶156.

D. Overview of Bigbee-Nguyen-Goodrum Combination

Claims 9-12 all depend from claim 1. The elements of claim 1 are taught by Bigbee-Nguyen. Section IV.C. The additional elements of claims 9-12 are taught by Goodrum and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Goodrum. Ex. 1002, ¶157.

Goodrum recognizes that a fault-tolerant scheme is desirable to ensure that a multiprocessor system continues to function despite the occurrence of a failure in a processor. Goodrum, 2:15-31. Goodrum further recognizes the criticality of a boot processor in a multiprocessor system. Goodrum, 1:17-20, 2:15-31. Typically one processor in a multiprocessor system is assigned the role of boot processor, which is responsible for initializing the system and turning on the remaining processors. Goodrum, 2:15-31, 9:39-50. If the boot processor fails during startup, then it may not turn on the other processors, leaving the system incapacitated. Goodrum, 2:15-31; Ex. 1002, ¶158.

To provide a fault-tolerant system, Goodrum describes a hot spare boot circuit that detects a failure of a designated boot processor and reassigns the role of boot processor to a backup processor. Goodrum, Abstract, 1:17-20, 2:15-64, 3:7-31, 5:66-6:3, 6:28-57, 8:11-16, 8:60-64, 9:39-63, 17:7-16, 18:34-49, Fig. 2. Goodrum distinguishes the boot processor from the backup processor using first and second identifier values, respectively. Goodrum, 8:14-16, 17:7-19. In response to detecting a failure, the hot spare circuit disables the boot processor, modifies the identifier value of the backup processor to designate the backup processor as the new boot processor, and then causes the new boot processor to complete system power on operations. Goodrum, 3:7-31, 9:39-63, 18:34-49; Ex. 1002, ¶159.

A POSITA would have understood the benefits to the Bigbee-Nguyen system of applying Goodrum's teaching to identify and switch to a backup processor for a failed boot processor. In particular, based on the teachings of Goodrum and Bigbee-Nguyen, a POSITA would have understood the desirability of detecting an error associated with a boot processor and the ability to identify a replacement backup processor both at startup and after startup. While Goodrum focuses on detecting boot processor errors during startup, Bigbee-Nguyen describes continuously monitoring its processors, and idling and restarting a processor for which a loss of lockstep is detected. After startup, if a failed boot processor is not switched to a backup processor, then the system may attempt to power up using the failed boot processor which Goodrum explains would leave the system incapacitated. Further, a POSITA would have understood that it would be beneficial for a recovered processor to then be used as a backup processor for the new boot processor, based on Goodrum's teaching regarding the benefits of maintaining a backup boot processor. Ex. 1002, ¶160.

Accordingly, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen) to use the original boot processor as a backup processor for the new boot processor (based on the teachings of Goodrum to provide a spare processor). By doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or reset the processor for which a loss of lockstep was detected. Ex. 1002, ¶161. The addition of Goodrum does not remove the functionality of Bigbee-Nguyen that reads on claim 1, as described above.

A POSITA would have understood that Bigbee-Nguyen's firmware-based processor error handling routines, which already detect a loss of lockstep in the processors, would be modified to determine if a processor error occurred in the processor identified as the boot processor and if so, to cause a backup processor take over the role of boot processor, and to use the recovered initial boot processor as a backup to the new boot processor based on Goodrum's teachings. Combining the teachings of Bigbee-Nguyen and Goodrum in this manner is nothing more than applying a known technique (Goodrum's technique for identifying a boot processor and a backup processor, and designating the backup processor as the new boot processor when the originally-designated boot processor has an error detected) to a known device (Bigbee-Nguyen's system having multiple processors in lockstep that can suffer a loss of lockstep fault involving a boot processor) ready for improvement (the ability to designate a non-faulty processor as the boot processor when there is a fault with the boot processor and to use the recovered original boot processor as backup processor) to yield a predictable result (Bigbee-Nguyen-Goodrum which, when a loss of lockstep is detected that involves a boot processor, identifies another processor to act as the boot processor and then uses the recovered processor as a backup, yielding better system reliability). Ex. 1002, ¶162.

A POSITA would have expected success in combining Goodrum with Bigbee-Nguyen because this combination merely involves routine programming skills that a POSITA would have had. As noted above (Section IV.B.2), a POSITA would know how to use ASL to program Bigbee-Nguyen's firmware. As acknowledged by Bigbee, control methods for handling processor errors may be written in ASL and ACPI provides the ability to store information regarding the hardware devices (e.g., processors) present in a system. Bigbee, [0016]-[0022]. A POSITA would therefore use ASL or other ACPI features to program Bigbee-Nguyen's firmware to, for example, (1) associate different identifiers to a boot processor and backup processor, (2) determine if a processor with an error detected is designated as a boot processor based on the identifier, and (3) if so, to cause a backup processor to take over the role of boot processor and to use the recovered boot processor as a spare processor. Thus, a POSITA would have expected success in combining Goodrum with Bigbee-Nguyen as set forth above. Ex. 1002, ¶163.

E. 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

As explained in Section IV.C, Bigbee-Nguyen teaches the method of claim 1. Claim 9 further requires determining if the detected loss of lockstep involved a system boot processor. As explained above, determining whether an error occurs in a boot processor is taught by Goodrum. Section IV.D.

Goodrum teaches a system whereby the primary processor is designated as such, e.g., by having a particular identifier. Goodrum, 3:7-31; 8:14-16, 17:7-19. Goodrum also notes that an identifier of CPU0 is often used to designate the boot processor. Goodrum, 2:39-50. As explained above, Goodrum identifies the boot processor so that it can switch to a spare processor in the event an error or failure is detected. Goodrum, Abstract, 1:17-20, 2:15-64, 3:7-31, 5:66-6:3, 6:28-57, 8:11-16, 8:60-64, 9:39-63, 17:7-16, 18:34-49. Motivations for incorporating this teaching into Bigbee-Nguyen are discussed above. Section IV.D; Ex. 1002, ¶164.

Accordingly, in Bigbee-Nguyen-Goodrum, when a loss of lockstep is detected for a pair of processors, as taught in Bigbee-Nguyen (Section IV.C) the system would also identify whether the pair comprises a boot processor and if so, identify a standby processor as the new boot processor. Ex. 1002, ¶165.

Thus, Bigbee-Nguyen-Goodrum renders claim 9 obvious. Ex. 1002, ¶166.

F. 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

As explained, claim 9 is taught by Bigbee-Nguyen-Goodrum. Claim 10 is not limiting. Section III.B.3. In any event, Goodrum also teaches the additional limitation of claim 10, and this claim is rendered obvious by Bigbee-Nguyen-Goodrum. Ex. 1002, ¶167.

In particular, Goodrum teaches that, when a failure is detected in a designated boot processor, the role of boot processor is reassigned to a backup processor. Goodrum, Abstract, 1:17-20, 2:15-64, 3:7-31, 5:66-6:3, 6:28-57, 8:11-16, 8:60-64, 9:39-63, 17:7-16, 18:34-49; Ex. 1002, ¶168.

As explained above, Goodrum teaches that this beneficially allows for a more fault-tolerant system that can continue to operate, even when the boot processor suffers a failure, in part because it will not attempt to reboot from a malfunctioning processor. Goodrum, 2:15-3:13. Motivations to make this combination are also described above. Section IV.D; Ex. 1002, ¶169.

Accordingly, in Bigbee-Nguyen-Goodrum, when a loss of lockstep is detected, as taught in Bigbee-Nguyen (Section IV.C) and as still occurs in Bigbee-Nguyen-Goodrum, the identifier of the CPU for which the loss of lockstep was detected would indicate whether the processor was the boot processor, and if so, the system would identify an operational spare CPU as the boot processor. This spare processor would then be used for further rebooting, if needed. Ex. 1002, ¶170.

Thus, Bigbee-Nguyen-Goodrum renders claim 10 obvious. Ex. 1002, ¶171.

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

As explained, claim 10 is taught by Bigbee-Nguyen-Goodrum. Claim 11 is not limiting. Section III.B.3. In any event, Bigbee also teaches the additional limitation of claim 11, and this claim is rendered obvious by Bigbee-Nguyen-Goodrum. Ex. 1002, ¶172.

In particular, as explained in Section IV.C.2 with respect to element [1B], Bigbee teaches recovering lockstep for a pair of processors. In Bigbee-Nguyen-Goodrum, the system will recover lockstep after having (as taught in Goodrum) swapped the role of boot processor to the spare processor. Ex. 1002, ¶173. Motivations for adding Goodrum's teachings to swap the processors are described above. Section IV.D.

Thus, claim 11 is rendered obvious by Bigbee-Nguyen-Goodrum. Ex. 1002, ¶174.

H. 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

As explained, claim 11 is taught by Bigbee-Nguyen-Goodrum. Claim 12 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Goodrum also teaches the additional limitation of claim 12. Ex. 1002, ¶175.

As described above for Bigbee-Nguyen, and which also occurs in Bigbee-Nguyen-Goodrum, when the loss of lockstep is recovered, the processor pair for which loss of lockstep has been recovered is inserted and the OS modifies its internal data structures to indicate that the processor is online and available (claimed "holding the recovered pair of processors in idle"). Section IV.C.2. The '958 Patent similarly discloses that a processor is idled when the processor is made available for use but no tasks are assigned to it. Ex. 1001, 6:11-15, 9:23-27, 10:26-34. Applying the teachings of Goodrum, the processor pair would be available for use as a spare boot processor because it would be operational, yet would not be identified as the boot processor (claimed "spare for the processor to which the role of system boot processor was swapped"), and this claim is also rendered obvious by Bigbee-Nguyen-Goodrum. Ex. 1002, ¶176.

Motivations for adding Goodrum's teachings to recognize that the idled processor pair for which lockstep was recovered is available as a spare boot processor are described above. Section IV.D. A POSITA would have recognized that, once recovered, the processor pair which was no longer the boot processor would become the spare processor, because of the reliability benefits of having a spare processor described above. Ex. 1002, ¶177.

Accordingly, claim 12 is rendered obvious by Bigbee-Nguyen-Goodrum. Ex. 1002, ¶178.

V. GROUND B: CLAIMS 9-12 ARE OBVIOUS OVER BIGBEE-NGUYEN-SUFFIN

A. Overview of Bigbee-Nguyen-Suffin Combination

Claims 9-12 all depend from claim 1. The elements of claim 1 are taught by Bigbee-Nguyen. Section IV.C. The additional elements of claims 9-12 are taught by Suffin and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Suffin. Ex. 1002, ¶179.

Suffin teaches the known concept of having a list of processors that may be used for booting the system, determining which of those processors are available, skipping those on the list that are not, and then booting the system using an available processor. Suffin, 2:5-4:5, 11:3-9, 11:23-12:18, 13:1-7, 13:16-14:10, 16:20-17:11, Fig. 4A. Since Suffin's list tracks which processor is the current boot processor and which processors have had an error detected, it detects when the boot processor has an error detected. Further, if boot up is not successful using that processor, then the next processor on the list is used. *Id*. This prevents the system from being unable to reboot due to a failure of the boot processor. And, Suffin teaches that the system can iterate the list so that the backup processor remains the boot processor and the original boot processor is now a backup processor, which helps avoid repetitive failures. *Id.,* 16:20-17:11; Ex. 1002, ¶180.

Suffin teaches that using such a process beneficially allows for "robust operation of a fault-tolerant computer, including initialization and recovery from operational failures." Suffin, 1:1-6, 2:5-17. Suffin also touts the "superior reliability" caused by having such built-in redundancy, and that the use of a second processor for reboot beneficially allows analysis of the faulty processor, because its state characteristics can be copied for later analysis, and the state characteristics of the non-faulty processor can be copied to the faulty processor. *Id.*, 1:12-18, 3:16-4:5, 16:20-17:11; Ex. 1002, ¶181.

Applying Suffin to Bigbee-Nguyen yields a Bigbee-Nguyen-Suffin system that monitors the processors for loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the system identifies a spare processor from Suffin's boot list to act as the boot processor. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a POSITA would have understood the benefits of using the recovered boot processor as a backup processor to the new boot processor. The addition of Suffin to the system does not remove the functionality of Bigbee-Nguyen that reads on claim 1, as described above. Ex. 1002, ¶182.

A POSITA would have been motivated to apply Suffin's teaching to achieve the benefits described in Suffin: superior reliability from having spare processors identified to be used for rebooting when the boot processor fails, and the ability to better diagnose and correct faults with the boot processor. Suffin, 1:1-6, 1:12-18, 2:5-17. This is simply applying a known technique (Suffin's technique of changing boot processors when there is a fault identified in the initial boot processor and if the boot processor is recovered, having it available as a spare processor) to a known device (Bigbee-Nguyen's system having multiple processors in lockstep that can suffer a loss of lockstep fault involving a boot processor) ready for improvement (the ability to boot from a spare, non-faulty processor when there is a fault with the boot processor) to yield predictable results (Bigbee-Nguyen-Suffin which, when a loss of lockstep is detected that involves a boot processor, identifies a spare processor to act as the boot processor, yielding better reliability, fault correction, and diagnosis). Ex. 1002, ¶183.

It also is an example of applying a known technique (Suffin's techniques described above) to improve similar devices (Bigbee-Nguyen's system which, like the system of Suffin, has multiple processors operating in lockstep that are subject to faults) in the same way (when a loss of lockstep is detected that involves a boot processor, identifying another processor to act as the boot processor, yielding better reliability, fault correction, and diagnosis). Ex. 1002, ¶184.

A POSITA would have expected success in combining Suffin with Bigbee-Nguyen because this combination merely involves routine programming skills that a POSITA would have had. As noted above (Sections IV.B.2, IV.D), a POSITA would know how to use ASL to program Bigbee-Nguyen's firmware to modify its processor error handling responses. A POSITA would therefore use ASL or other ACPI features to program Bigbee-Nguyen's firmware to (1) access Suffin's processor list to determine if a processor with an error detected is a boot processor and (2) if so, to cause a backup processor to take over the role of boot processor and use the recovered boot processor as a spare processor. Thus, POSITA would have expected success in combining Suffin with Bigbee-Nguyen as set forth above. Ex. 1002, ¶185.

B. Claim 9

As explained in Section IV.C, Bigbee-Nguyen teaches the method of claim 1. Claim 9 further requires the step of determining if the pair of processors for which the loss of lockstep is detected comprises a system boot processor. As explained above (Section V.A), determining whether an error, such as loss of lockstep, occurs in a boot processor is taught by Suffin. Ex. 1002, ¶186.

In Suffin, the boot list contains processor identifiers and values for each processor indicating whether each processor is functioning properly. Suffin, 12:3-18. Thus, Suffin monitors whether a particular processor, such as the first processor on the boot list which was used to initially boot the system (the boot processor), remains functional, including whether it has suffered a loss of lockstep with another processor. *Id.,* 6:9-11, 7:4-16, 8:1-4, 12:3-18, 13:1-7, 14:15-18, 15:7-18. Suffin teaches that this beneficially allows for a more robust, reliable system that can better diagnose problems with processors and that will not attempt to reboot, if that becomes necessary, from a malfunctioning processor. Motivations to make this combination are described above. Section V.A; Ex. 1002, ¶187.

Accordingly, in Bigbee-Nguyen-Suffin, when a loss of lockstep is detected and the processors are idled, as taught in Bigbee-Nguyen (Section IV.C), the system's boot list would indicate which processor pair was involved with the loss of lockstep, so that a different processor would be used for rebooting, if needed. In the event the loss of lockstep involved processors that included the system boot processor then the system would have identified that the boot processor had a loss of lockstep, idled it as described above with respect to claim 1, and the processor would be treated as unavailable for use as a boot processor until recovered. Ex. 1002, ¶188.

Thus, Bigbee-Nguyen-Suffin renders claim 9 obvious. Ex. 1002, ¶189.

C. Claim 10

As explained, claim 9 is taught by Bigbee-Nguyen-Suffin. Claim 10 is not limiting. Section III.B.3. In any event, Suffin also teaches the additional limitation of claim 10, and this claim is rendered obvious by Bigbee-Nguyen-Suffin. Ex. 1002, ¶190.

In particular, Suffin monitors the functionality of the processors and teaches that a malfunctioning processor (such as the processor for which a loss of lockstep was detected and which is idled in Bigbee-Nguyen-Suffin) will not be used as the boot processor, but instead is replaced by an available processor on the boot list (claimed "swapping the role of system boot processor to another processor in the system"). Suffin, 9:1-4, 11:23-12:18, 13:1-7, 13:16-14:6, 16:9-17:2, Fig. 4A. As explained above, Suffin teaches that this beneficially allows for a more robust, reliable system that can better diagnose problems with processors and that will not attempt to reboot the system from a malfunctioning processor. Motivations to make this combination are described above. Section V.A; Ex. 1002, ¶191.

Accordingly, in Bigbee-Nguyen-Suffin, when a loss of lockstep is detected and the processors are idled, as taught in Bigbee-Nguyen (Section IV.C) and as still occurs in Bigbee-Nguyen-Suffin, the system's boot list would indicate which processors were involved with the loss of lockstep and thus unavailable, and a different processor on the boot list thus would be swapped in as the current boot processor and used for rebooting, if needed. Ex. 1002, ¶192.

Thus, Bigbee-Nguyen-Suffin renders claim 10 obvious. Ex. 1002, ¶193.

D. Claim 11

As explained, claim 10 is taught by Bigbee-Nguyen-Suffin. Claim 11 is not limiting. Section III.B.3. In any event, Bigbee also teaches the additional limitation of claim 11, and this claim is rendered obvious by Bigbee-Nguyen-Suffin. Ex. 1002, ¶194.

In particular, as explained in Section IV.C.2 with respect to element [1B], Bigbee teaches to recover lockstep for a processor pair after it is detected. In Bigbee-Nguyen-Suffin, the system will recover that loss of lockstep after having (as taught in Suffin) detected the error and swapped the role of boot processor to the spare processor when the previous boot processor was idled and thus determined to be unavailable on the boot list. Motivations for adding Suffin's teachings to remove the idled processor from the available processors on the boot list are described above. Section V.A; Ex. 1002, ¶195.

Accordingly, claim 11 is rendered obvious by Bigbee-Nguyen-Suffin. Ex. 1002, ¶196.

E. Claim 12

As explained, claim 11 is taught by Bigbee-Nguyen-Suffin. Claim 10 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Suffin also teaches the additional limitation of claim 12.

As described above for Bigbee-Nguyen, and which also occurs in Bigbee-Nguyen-Suffin, when the loss of lockstep is recovered, the processor pair for which loss of lockstep has been recovered is inserted and the operating system modifies its internal data structures to indicate that the processor pair is online and available (claimed "holding the recovered pair of processors in idle"). Section IV.C. As noted in Section IV.H, the '958 Patent similarly discloses that a processor is idled when the processor is made available for use but no tasks are assigned to it. Ex. 1001, 6:11-15, 9:23-27, 10:26-34. Applying the teachings of Suffin (Suffin, 16:20-17:11, 13:1-7), the processor pair would then be available for use as a spare boot processor on the boot list (claimed "spare for the processor to which the role of system boot processor was swapped"), and this claim is also rendered obvious by Bigbee-Nguyen-Suffin. Ex. 1002, ¶197.

Motivations for adding Suffin's teachings to recognize that the idled processor pair for which lockstep was recovered is available as a spare boot processor are described above. Section V.A. A POSITA would have recognized that, once recovered, the processor pair which was no longer identified as the boot processor would become the spare processor, because of the reliability benefits of having a spare processor described above. Ex. 1002, ¶198.

Accordingly, claim 12 is rendered obvious by Bigbee-Nguyen-Suffin. Ex. 1002, ¶199.

VI. GROUND C: CLAIMS 23-25 AND 12 ARE OBVIOUS OVER BIGBEE-NGUYEN-SUFFIN-SAFFORD

A. Overview of Bigbee-Nguyen-Suffin-Safford Combination

Claims 23-25 possess limitations that are similar to those found in claims 1 and 9-12, but also add that, if loss of lockstep is determined to have occurred in the boot processor, the system will copy a state of the boot processor to the spare boot processor. This limitation is taught by Safford (Safford, Abstract, [0006]-[0007], [0018]-[0021], Figs. 3-4), and claims 23-25 are taught by Bigbee-Nguyen-Suffin-Safford. Ex. 1002, ¶200.

Safford provides a motivation to combine, as it explains a benefit of copying the state of the processor for which a loss of lockstep has been detected is to "enhance recovery from loss of lockstep." Safford, [0006], [0021], Abstract; Ex. 1002, ¶201.

Applying Safford to Bigbee-Nguyen-Suffin yields a Bigbee-Nguyen-Suffin-Safford system that, pursuant to the teachings of Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies its state to the hot spare processor. The addition of Safford to the system does not remove the functionality of Bigbee-Nguyen-Suffin that reads on claims 9-12, as described above. Ex. 1002, ¶202.

A POSITA would have been motivated to apply Safford's teaching to achieve the benefits described in Safford: enhanced, quicker recovery from a loss of lockstep. Safford, Abstract, [0005]-[0007], [0021]. This is applying a known technique (Safford's technique of copying the state of a processor for which a loss of lockstep has been detected to the spare processor) to a known device (Bigbee-Nguyen-Suffin's system having multiple processors in lockstep that can suffer a loss of lockstep fault in a boot processor that causes another, spare processor to become the boot processor) ready for improvement (enhanced, quicker, recovery from a loss of lockstep) to yield predictable results (Bigbee-Nguyen-Suffin-Safford which, when a loss of lockstep is detected that involves a boot processor, identifies a spare processor to act as the boot processor and copies the state of the boot processor to the spare processor). Ex. 1002, ¶203.

It also applies a known technique (Safford's techniques described above) to improve similar devices (Bigbee-Nguyen-Suffin's system which, like Safford, has multiple processors operating in lockstep that are subject to faults, and spare processors that become active in the event a processor has a loss of lockstep detected) in the same way (when a loss of lockstep is detected that involves a boot processor, identifying another processor to act as the boot processor and copying the state of the first processor to the second, yielding enhanced recovery from the loss of lockstep). Ex. 1002, ¶204.

A POSITA would have expected success in combining Safford with Bigbee-Nguyen-Suffin because this combination merely involves routine programming skills. As noted above (Sections IV.B.2, IV.D, V.A), a POSITA would know how to use ASL to program Bigbee-Nguyen-Suffin's firmware to modify its processor error handling responses. Bigbee explains that these processor error responses include saving processor state information and ensuring that the state of a processor for which an error is detected is reinitialized to enable reintroduction. Bigbee, [0021], [0023]-[0024]. A POSITA would therefore use ASL or other ACPI features to program Bigbee-Nguyen-Suffin's firmware to copy to a spare processor the already saved state of a processor for which a loss of lockstep has been detected. Thus, POSITA would have expected success in combining Safford with Bigbee-Nguyen-Suffin as set forth above. Ex. 1002, ¶205.

B. Independent Claim 23

1. Limitation [23A]: A method comprising: establishing, in a multi-processor system, a hot spare processor for a system boot processor

Limitation [23A] is rendered obvious by Bigbee-Nguyen-Suffin. Bigbee-Nguyen-Suffin possesses the functionality of Bigbee-Nguyen, described above in Sections IV.A-C, and adds teachings of Suffin, as described in Section V. In the

Bigbee-Nguyen-Suffin system there are multiple processors, and, per the teachings of Suffin (Suffin, 2:5-4:5, 11:3-8, 11:23-12:18, 13:1-7, 13:16-14:10, Fig. 4A), the system has a boot list that contains a listing of the current boot processor and other available processors, any of which is a claimed "hot spare" processor for the system boot processor, because each such processor can be designated as the boot processor if the boot processor becomes unavailable (i.e., malfunctions). Ex. 1002, ¶206.

Safford also teaches limitation [23A], and this claim limitation is taught by Bigbee-Nguyen-Suffin-Safford. Specifically, Safford teaches, when a loss of lockstep is detected in a multi-processor system, "moving an architected state of the processor unit experiencing the loss of lockstep to a spare processor, wherein the spare processor unit becomes an active processor unit in the computer system." Safford, [0007], [0018]-[0021] (discussion of using "hot standby" processors), Figs. 3-4 (copying architected state of processor to "hot standby" processor, designating new processor as hot standby processor), Abstract. The benefits and motivations for having such a spare processor are described above and apply equally to Safford's teaching of this feature. Sections VI.A, V.A; Ex. 1002, ¶207.

Accordingly, Bigbee-Nguyen-Suffin-Safford teach limitation [23A]. Ex. 1002, ¶208.

2. Limitation [23B]: detecting loss of lockstep for a lockstep pair of processors in said multi-processor system

As described for limitation [1A], this step is taught by Bigbee and is performed in the Bigbee-Nguyen-Suffin-Safford system. Section IV.C.1; Ex. 1002, ¶209.

3. Limitation [23C]: determining if said lockstep pair of processors for which the loss of lockstep is detected is the system boot processor

As described for claim 9, this step is taught by Suffin and is performed in the Bigbee-Nguyen-Suffin-Safford system. Section V.B; Ex. 1002, ¶210.

4. Limitation [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

Limitation [23D] is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Suffin-Safford teaches this step. Ex. 1002, ¶211.

As described above, Safford, like Bigbee-Nguyen-Suffin, relates to a multi-processor system with processors operating in lockstep. Safford, Abstract, [0001], [0006]-[0007], [0018]-[0021], Figs. 2-4. In order to "enhance recovery from loss of lockstep," Safford copies the state of one or more of the processors in a processor pair for which a loss of lockstep has been detected to a hot spare processor unit, which becomes an active processor unit in the computer system. *Id.*. This process is shown, for instance, in Fig. 4, step 220:

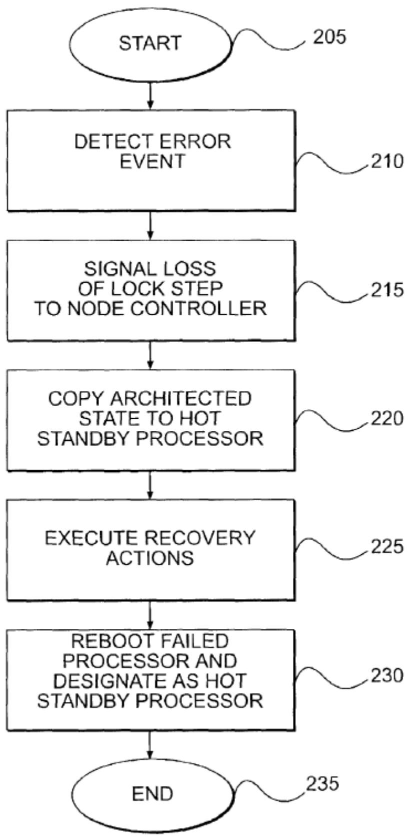

START
205
DETECT ERROR EVENT
210
SIGNAL LOSS OF LOCK STEP TO NODE CONTROLLER
215
COPY ARCHITECTED STATE TO HOT STANDBY PROCESSOR
220
EXECUTE RECOVERY ACTIONS
225
REBOOT FAILED PROCESSOR AND DESIGNATE AS HOT STANDBY PROCESSOR
230
END
235

FIG. 4

Safford, Fig. 4, [0021]. Once the processor with the loss of lockstep is idled and lockstep is recovered, then that processor unit (whose state was copied to the standby processor) becomes the new hot standby processor, as seen in step 230. *Id.*; Ex. 1002, ¶212.

Accordingly, in Bigbee-Nguyen-Suffin-Safford, when the processor pair for which a loss of lockstep has been detected includes the boot processor, the system will copy a state of the system boot processor to the spare processor. Motivations for adding this teaching of Safford are discussed above. Section VI.A. Ex. 1002, ¶213.

5. Limitation [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

Limitation [23E] is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Suffin-Safford teaches this step. Ex. 1002, ¶214.

Limitation [23E] is substantively included in limitation [1B], and it is taught by Bigbee-Nguyen-Suffin-Safford for the same reasons that Bigbee-Nguyen teaches limitation [1B]. Section IV.C.2. The addition of Safford's and Suffin's teachings do not remove the functionality of Bigbee-Nguyen that teaches limitation [1B]. Ex. 1002, ¶215.

Accordingly, Bigbee-Nguyen-Suffin-Safford renders claim 23 obvious. Ex. 1002, ¶216.

C. 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

The additional limitation of claim 24 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Suffin-Safford teaches this step. Ex. 1002, ¶217.

As explained, claim 23 is taught by Bigbee-Nguyen-Suffin-Safford. As explained for claim 11, Bigbee-Nguyen-Suffin teaches that loss of lockstep for the processor pair is recovered after swapping the role of boot processor to the spare processor. Sections V.D, IV.C.2. This functionality remains in Bigbee-Nguyen-Suffin-Safford. Further, Safford explicitly teaches in Figure 4 that the copying occurs before attempting to recover the loss of lockstep, wherein step 225 (attempting to recover lockstep) occurs after step 220 (the copying of the processor to the hot spare):

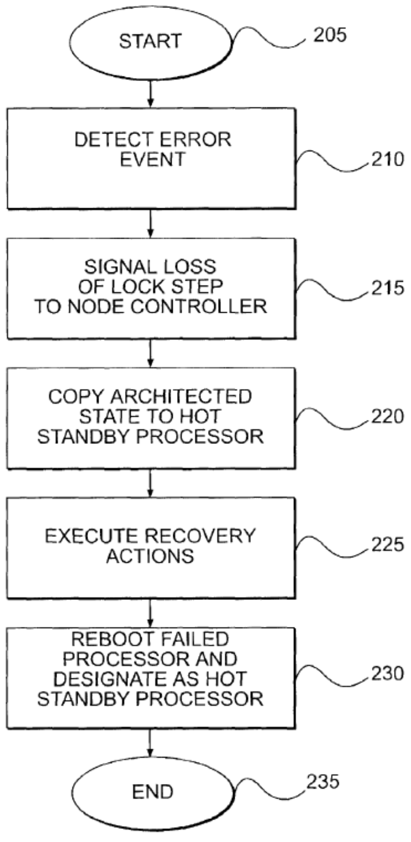

START
205
DETECT ERROR EVENT
210
SIGNAL LOSS OF LOCK STEP TO NODE CONTROLLER
215
COPY ARCHITECTED STATE TO HOT STANDBY PROCESSOR
220
EXECUTE RECOVERY ACTIONS
225
REBOOT FAILED PROCESSOR AND DESIGNATE AS HOT STANDBY PROCESSOR
230
END
235

FIG. 4

Safford, Fig. 4, [0006]-[0007], [0021]. Therefore, in the combined system, the attempted recovery of the loss of lockstep occurs after copying the state of the system boot processor to the hot spare processor. Ex. 1002, ¶218.

The motivations for applying Safford to Bigbee-Nguyen-Suffin are described above. Section VI.A. Accordingly, Bigbee-Nguyen-Suffin-Safford renders claim 24 obvious. Ex. 1002, ¶219.

D. 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

The additional limitation of claim 25 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Suffin-Safford teaches this step. Ex. 1002, ¶220.

As explained, claim 24 is taught by Bigbee-Nguyen-Suffin-Safford. As explained with respect to claim 12, Bigbee-Nguyen-Suffin teach that, when the loss of lockstep is recovered in the boot processor pair, the boot processor pair becomes available as a spare processor. Section V.E. This functionality remains in the Bigbee-Nguyen-Suffin-Safford combination, which renders claim 25 obvious. Ex. 1002, ¶221.

Further, Safford also teaches that, once loss of lockstep is recovered in the processor pair that was idled and copied, that processor pair is designated as the standby processor. Safford, Fig. 4, [0021] (discussing that the recovered processors "become the new 'hot standby' processor pair"). As noted above (Sections VI.A, V.A), a POSITA would have recognized that, once recovered, the processor which was no longer identified as the boot processor would become the spare processor, because of the reliability benefits (taught in Safford) of having a spare processor. The motivations for applying Safford to Bigbee-Nguyen-Suffin are described above (Section VI.A) and the motivations for having a hot spare processor described with respect to the teachings of Suffin and Goodrum also apply to those teachings in Safford. Accordingly, for this reason also, the Bigbee-Nguyen-Suffin-Safford combination renders claim 25 obvious. Ex. 1002, ¶222.

E. Claim 12

As explained, claim 11 is taught by Bigbee-Nguyen-Suffin. Section V.D. Claim 12 is not limiting. Section III.B.3. In any event, as also explained in Section VI.D, Safford also teaches that, after lockstep is recovered for a pair of processors, the recovered pair of processors become the new hot standby processors, which are held in idle until needed. Safford, [0021], [0006]-[0007], Abstract, Fig. 4; Ex. 1001, 6:11-15, 9:23-27, 10:26-34. The motivations for applying Safford to Bigbee-Nguyen-Suffin are described above. Sections VI.A, VI.D. Thus, Bigbee-Nguyen-Suffin-Safford renders claim 12 obvious. Ex. 1002, ¶223.

VII. GROUND D: CLAIMS 23-25 AND 12 ARE OBVIOUS OVER BIGBEE-NGUYEN-GOODRUM-SAFFORD

Ground D adds the teachings of Safford to Bigbee-Nguyen-Goodrum in the same way that they are added to Bigbee-Nguyen-Suffin in Ground C, and the motivations for adding Safford to Bigbee-Nguyen-Goodrum are the same as described above with respect to Bigbee-Nguyen-Suffin. Section VI; Ex. 1002, ¶224.

A. Independent Claim 23

1. Limitation [23A]

Limitation [23A] is taught by Bigbee-Nguyen-Goodrum. Bigbee-Nguyen-Goodrum possesses the functionality of Bigbee-Nguyen, described above in Section IV.A-C, and adds teachings of Goodrum, as described in Section IV.D-H. In the Bigbee-Nguyen-Goodrum system there are multiple processors, and, per the teachings of Goodrum (Goodrum, 1:17-20, 3:10-13; 8:60-64; 9:39-50; 18:34-49), the system identifies one of the processors as the current boot processor and another available processor is a claimed "hot spare" processor for the system boot processor, because it can be designated as the boot processor if the boot processor becomes unavailable (i.e., malfunctions). Ex. 1002, ¶225.

As described in Section VI.B.1, Safford also teaches limitation [23A], and motivations to add this teaching to Bigbee-Nguyen-Goodrum are the same as described for combining it with Bigbee-Nguyen-Suffin. Sections IV.D, VI; Ex. 1002, ¶226.

Accordingly, Bigbee-Nguyen-Goodrum-Safford teach the method of limitation [23A]. Ex. 1002, ¶227.

2. Limitation [23B]

As described for limitation [1A], this step is taught by Bigbee and is performed in the Bigbee-Nguyen-Goodrum-Safford system. Section IV.C.1; Ex. 1002, ¶228.

3. Limitation [23C]

As described for claim 9, this step is taught by Goodrum and is performed in the Bigbee-Nguyen-Goodrum-Safford system. Section IV.E; Ex. 1002, ¶229.

4. Limitation [23D]

Limitation [23D] is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Goodrum-Safford teaches this step. Ex. 1002, ¶230.

As described in Section VI.B.4, Safford, in order to "enhance recovery from loss of lockstep," copies the state of one or more of the processors in a processor pair for which a loss of lockstep has been detected to a hot spare processing unit, which becomes an active processor unit in the computer system. *Id.*, Abstract, [0006]-[0007], [0018]-[0021], Fig. 4; Ex. 1002, ¶231.

Accordingly, in Bigbee-Nguyen-Goodrum-Safford, when the processor pair for which a loss of lockstep has been detected includes the boot processor, the system will copy a state of the system boot processor to the hot spare processor. Motivations for adding this teaching of Safford are discussed in the introduction to this ground and in Section VI.A. Bigbee-Nguyen-Goodrum-Safford thus teaches this limitation. Ex. 1002, ¶232.

5. Limitation [23E]

Limitation [23E] is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Goodrum-Safford teaches this step. Ex. 1002, ¶233.

Limitation [23E] is substantively included in limitation [1B], and it is taught by Bigbee-Nguyen-Goodrum-Safford for the same reasons that Bigbee-Nguyen teaches limitation [1B]. Section IV.C.2. The addition of Safford's and Goodrum's teachings do not remove the functionality of Bigbee-Nguyen that teaches limitation [1B]. Ex. 1002, ¶234.

Accordingly, Bigbee-Nguyen-Goodrum-Safford renders claim 23 obvious. Ex. 1002, ¶235.

B. Claim 24

The additional limitation of claim 24 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Goodrum-Safford teaches this step. Ex. 1002, ¶236.

As explained, claim 23 is taught by Bigbee-Nguyen-Goodrum-Safford. As explained for claim 11, Bigbee-Nguyen-Goodrum teaches that loss of lockstep for the processor pair is recovered after swapping the role of boot processor to the spare processor. Sections IV.G, IV.C.2. This functionality remains in Bigbee-Nguyen-Goodrum-Safford. Further, Safford explicitly teaches in Figure 4 that the copying occurs before attempting to recover the loss of lockstep, wherein step 225 (attempting to recover lockstep) occurs after step 220 (the copying of the processor to the hot spare):

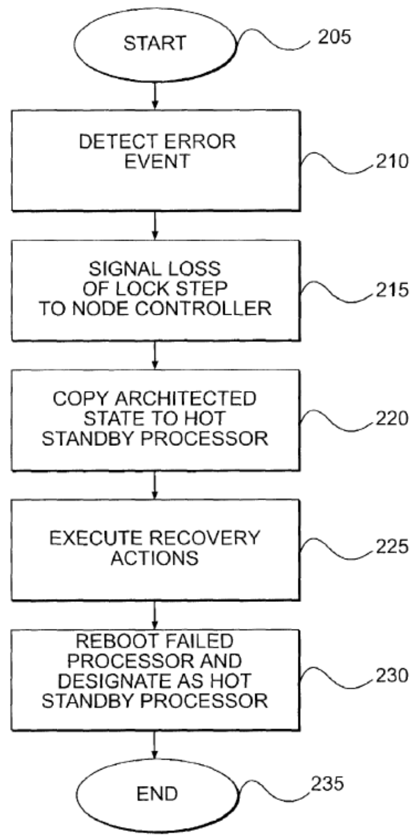

START
205
DETECT ERROR EVENT
210
SIGNAL LOSS OF LOCK STEP TO NODE CONTROLLER
215
COPY ARCHITECTED STATE TO HOT STANDBY PROCESSOR
220
EXECUTE RECOVERY ACTIONS
225
REBOOT FAILED PROCESSOR AND DESIGNATE AS HOT STANDBY PROCESSOR
230
END
235

FIG. 4

Safford, Fig. 4, [0006]-[0007], [0021]. Therefore, in the combined system, the attempted recovery of the loss of lockstep occurs after copying the state of the system boot processor to the spare processor. Ex. 1002, ¶237.

The motivations for applying Safford to Bigbee-Nguyen-Goodrum are described above. Accordingly, Bigbee-Nguyen-Goodrum-Safford renders claim 24 obvious. Ex. 1002, ¶238.

C. Claim 25

The additional limitation of claim 25 is not limiting. Section III.B.3. In any event, Bigbee-Nguyen-Goodrum-Safford teaches this step. Ex. 1002, ¶239.

As explained, claim 24 is taught by Bigbee-Nguyen-Goodrum-Safford. As explained with respect to claim 12, Bigbee-Nguyen-Goodrum teaches that, when the loss of lockstep is recovered in the boot processor pair, the boot processor pair becomes available as a spare processor. Section IV.H. This functionality remains in the Bigbee-Nguyen-Goodrum-Safford combination, which renders claim 25 obvious. Ex. 1002, ¶240.

Further, Safford also teaches that, once loss of lockstep is recovered in the processor pair that was idled and copied, that processor pair is designated as the standby processor. Safford, Fig. 4, [0021] (discussing that the recovered processors "become the new 'hot standby' processor pair"). As noted above (Sections IV.D, V.A, V.E, VI.A, VII introduction), a POSITA would have recognized that, once recovered, the processor which was no longer identified as the boot processor would become the spare processor, because of the reliability benefits (taught in Safford) of having a spare processor. For this reason also, the Bigbee-Nguyen-Goodrum-Safford combination renders claim 25 obvious. Ex. 1002, ¶241.

D. Claim 12

As explained, claim 11 is taught by Bigbee-Nguyen-Goodrum. Section IV.G. Claim 12 is not limiting. Section III.B.3. In any event, as explained in Section VII.C, Safford also teaches that, after lockstep is recovered for a pair of processors, the recovered pair of processors become the new hot standby processors, which are held in idle until needed. Safford, [0021], [0006]-[0007], Abstract, Fig. 4; Ex. 1001, 6:11-15, 9:23-27, 10:26-34. The motivations for applying Safford to Bigbee-Nguyen-Goodrum are described above. Section VII.A, VII.C. Thus, Bigbee-Nguyen-Goodrum-Safford renders claim 12 obvious. Ex. 1002, ¶242.

VIII. 314(A) DISCRETION DOES NOT APPLY

A. *Fintiv* Discretionary Denial Is Unwarranted

The *Fintiv* factors set forth in the Director's June 21, 2022 Guidance Memorandum do not warrant discretionary denial.

Factor One (whether a stay may be granted)

Factor one favors institution. The parallel proceeding has been stayed pending resolution of the *ex parte* reexamination and IPRs of the '958 Patent. Ex. 1027.

Factor Two (proximity of trial date)

Factor two favors institution. The parallel proceeding has been stayed and thus the trial date will be reset.

Factor Three (investment in parallel proceeding)

Factor three favors institution. The investment by the District Court has been limited to holding a scheduling conference and ruling on the motion to stay. PO has served infringement contentions but no discovery has taken place.

Factor Four (overlap between issues)

Factor four favors institution. The parallel proceeding has been stayed and there should be no overlap of issues as Petitioners will be subject to 35 U.S.C. § 315(e)(2).

Factor Five (petitioner and defendant are same party)

Petitioners are defendants in the District Court case.

Factor Six (other circumstances including merits)

Factor six favors institution. Petitioners present compelling challenges that merit institution.

B. *General Plastic* Discretionary Denial Is Unwarranted

The factors set forth in *Gen. Plastic Indus. Co. v. Canon Kabushiki Kaisha*, IPR2016-01357, Paper 19 (PTAB Sept. 6, 2017) do not warrant discretionary denial.

Factor One (same petitioner, same claims)

Factor one favors institution. While Petitioners filed a prior IPR against the ’958 Patent (IPR2024-00727, “Joinder IPR”), it is a joinder petition to which Petitioners could not add additional claims or grounds. *Facebook, Inc. v. Windy City Innovations, LLC*, 953 F.3d 1313, 1325 (Fed. Cir. 2020). Petitioners are not related to the petitioners in the joined IPR (IPR2023-01373). *Cf. Ford Motor Co. v. Neo Wireless LLC*, IPR2023-00763, Paper 28 at 11 (PTAB Mar. 22, 2024) (Director review finding that court-ordered pretrial coordination between parties having different products did not support discretionary denial under *General Plastic*); Trial Practice Guide at 17 (participation in a joint defense group does not make parties into RPIs or privies).

The present petition challenges *different* claims (9-12, 23-25) than the Joinder IPR (1-8, 13-22). *See Xilinx Inc v Arbor Global Strategies LLC*, IPR2020-01568, Paper 12 at 4-5 (PTAB Mar. 5, 2021). While certain of the challenged claims (9-12) depend from previously-challenged claim 1, the grounds herein for challenging the limitations of claim 1 are identical to those in the Joinder IPR (and will effectively be controlled by the Joinder IPR outcome).

Factor Two (prior art)

Factor two is neutral. While Petitioners were aware of the new references relied on herein (Goodrum, Safford, and Suffin) at the time the Joinder IPR was filed, they could not add references and still seek joinder.

Factor Three (POPR and institution decision received)

Factor three is neutral. While the POPR and institution decision in the Joinder IPR were available before this petition was filed, the POPR was directed solely to *Fintiv* factors and the institution decision was directed to different claims. IPR2023-01373, Paper 7 (Dec. 22, 2023).

Factors Four and Five (time elapsed)

Factors four and five favor institution. Petitioners received PO's infringement contentions identifying the asserted claims (including the claims challenged here) on April 1 and filed this Petition approximately twelve weeks later. Ex. 1028, p. 1; Ex. 1029, p. 1. The claims challenged herein necessitated Petitioners relying on the new references.

Factors Six and Seven (Board Resources)

This Petition would not be an inefficient use of the Board's resources as it challenges different claims than the Joinder IPR. As discussed above, the limitations of claim 1 that are incorporated into claims 9-12 will be resolved in the Joinder IPR thus reducing the amount of work for the Board in this IPR.

IX. 325(D) DISCRETION DOES NOT APPLY

A. The Present Grounds Were Not Considered During Prosecution

Section 325(d) denial would be improper because this Petition presents new grounds not previously considered by the Office. *Advanced Bionics, LLC v. Med-El Elektromedizinische Geräte GmbH*, IPR2019-01469, Paper 6 (PTAB Feb. 13, 2020); *Becton, Dickinson & Co. v. B. Braun Melsungen AG*, IPR2017-01586, Paper 8 (PTAB Dec. 15, 2017).

Under *Advanced Bionics* step 1 and *Becton, Dickinson* factors (a), (b), and (d), Nguyen, Goodrum, and Suffin were not before the Examiner during prosecution. Ex. 1004.

Safford-959 was considered as a secondary reference in combination with Marshall (U.S. Patent No. 5,915,082). Ex. 1004, pp. 117-125. In the Final Office Action, the Examiner indicated the subject matter of original dependent claim 8, depending from original claim 7—collectively reciting "determining if a lockstep mismatch has occurred between the lockstep pair of processors" in "determining if lockstep is recoverable"—was allowable. Ex. 1004, pp. 123, 138. Applicant subsequently incorporated the allowable subject matter of claims 7 and 8 into independent claim 1 to achieve allowance over Marshall-Safford-959. *Id.*, p. 104.

Applicant cited Bigbee among 17 references listed in an information disclosure statement ("IDS") filed with a request for continued examination ("RCE") after two Notices of Allowance. Ex. 1004, pp. 41, 49, 77. The Examiner signed the IDS and issued a Notice of Allowance without discussion of any of the cited references. *Id.*, pp. 12, 29, 72.

Even assuming Bigbee was substantively considered by the Office during prosecution, the combinations of Bigbee set forth in this Petition were not considered. Ex. 1004.

Under *Advanced Bionics* step 2 and *Becton, Dickinson* factors (c), (e), and (f), the Office made a material error by not using Bigbee in a rejection of any claims. Although the Examiner signed the IDS listing Bigbee, the Examiner did not discuss or rely on Bigbee prior to allowance. Ex. 1004. Bigbee was relied upon by another

Examiner as anticipating claims 1-18 and 30-38 of U.S. Patent Application No. 10/972,888 ("the '888 Application")[6], which the Applicant cancelled in response to the rejection. Ex. 1025, pp. 27-29, 36-40. The '958 Examiner rejected the claims of the '958 Patent as not patentably distinct from the rejected claims in the '888 Application. Ex. 1004, pp. 166-167. The '958 Examiner's silence with respect to Bigbee before allowing the closely-related claims of the '958 Patent reflects error. Further, the Examiner did not have the opportunity to consider Bigbee as combined in this Petition, and thus could not appreciate the combinations' relevance.

B. Unified Patent's *Ex Parte* Reexamination Is Not Cumulative of this Petition

Unified Patents, LLC ("Unified") filed an *ex parte* reexamination ("EPR") of the '958 Patent, challenging claims 1-25. Ex. 1026. Unified proposed the following rejections:

Ground	Claims	Basis	Reference(s)
1	1-8, 13-22	§ 102	Bigbee
2	1-8, 13-22	§ 103	Bigbee
3	1-8, 13-22	§ 103	Safford-959, Marisetty
4	9-12, 23-25	§ 103	Safford-959, Cepulis, Marisetty

[6] The '958 Patent incorporates by reference the '888 Application. Ex. 1001, 1:26-28.

Reexamination was ordered on November 29, 2023, an office action issued on March 19, 2024, PO filed a response on May, 20, 2024, and the EPR was stayed in view of IPR2023-01373 on June 21, 2024.

Unified's EPR presents materially different grounds that are not substantially the same as this Petition because it does not rely on Nguyen, Goodrum, or Suffin.

X. GROUNDS FOR STANDING & FEE PAYMENT

Petitioner hereby certifies that the '958 Patent is available for *inter partes* review, and that Petitioner is not barred or estopped from requesting *inter partes* review on the grounds herein.

Pursuant to 37 C.F.R. § 42.103, the undersigned authorizes the charge of any required fees to Deposit Account No. 19-0733.

XI. CONCLUSION

Petitioner respectfully submits that *inter partes* review of claims 9-12 and 23-25 of U.S. Patent No. 7,502,958 be instituted.

BANNER & WITCOFF, LTD.

Dated: June 24, 2024

By: /John R. Hutchins/

John R. Hutchins
Reg. No. 43,686
Banner & Witcoff, Ltd.
1100 13th Street, NW
Suite 1200
Washington, DC 20005

CERTIFICATION UNDER 37 CFR § 42.24(d)

Under the provisions of 37 CFR § 42.24(d), the undersigned hereby certifies that the word count for the foregoing Petition for Inter Partes Review totals 13,969, as counted by the Word Count feature of Microsoft Word, which is less than the 14,000 allowed under 37 CFR § 42.24(a)(1)(i).

Pursuant to 37 C.F.R. § 42.24(a)(1), this count does not include the table of contents, the table of authorities, mandatory notices under § 42.8, the certificate of service, this certification of word count, the claims listing appendix, or appendix of exhibits.

Dated: June 24, 2024

By: /John R. Hutchins/

John R. Hutchins
Reg. No. 43,686

CERTIFICATE OF SERVICE

Pursuant to 37 C.F.R. § 42.105, I hereby certify that I caused a true and correct copy of the Petition for *Inter Partes* Review in connection with U.S. Patent No. 7,502,958 and supporting evidence to be served via FedEx Priority Overnight on June 24, 2024, on the following:

Antonio Papageorgiou
Lombard Geliebter LLP
230 PARK AVENUE 4TH FL WEST
New York, NY 10169

Atlantic IP (199393)
c/o Lombard Geliebter LLP
1325 Avenue of the Americas
28th Floor
New York, NY 10019

Dated: June 24, 2024

By: /John R. Hutchins/

John R. Hutchins
Reg. No. 43,686

CLAIM LISTING APPENDIX

U.S. Pat. No. 7,502,958

Designation	Claim Language
Claim 1	
[1A]	1. A method comprising: detecting loss of lockstep for a lockstep pair of processors;
[1B]	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
[1C]	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.
Claim 9	
[9A]	9. The method of claim 1 further comprising:
[9B]	determining if said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor.
Claim 10	
[10A]	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:
[10B]	swapping the role of system boot processor to another processor in the system.
Claim 11	
[11A]	11. The method of claim 10 further comprising:
[11B]	after said swapping, recovering the lockstep for the lockstep pair of processors.
Claim 12	
[12A]	12. The method of claim 11 further comprising:
[12B]	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.

Designation	Claim Language
Claim 23	
[23A]	23. A method comprising: establishing, in a multi-processor system, a hot spare processor for a system boot processor;
[23B]	detecting loss of lockstep for a lockstep pair of processors in said multi-processor system;
[23C]	determining if said lockstep pair of processors for which the loss of lockstep is detected is the system boot processor;
[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
[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.
Claim 24	
[24A]	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:
[24B]	attempting to recover lockstep between the lockstep pair of processors after copying the state of the system boot processor to the hot spare processor.
Claim 25	
[25A]	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:
[25B]	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.