allocated in block 504. For instance, an amount of SRAM may be allocated for storing the registers from the good processor.

PAGE 9/14 * RCVD AT 12/12/2007 5:22:43 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-1/14 * DNIS:2738300 * CSID:303 297 2266 * DURATION (mm-ss):04-18

REMARKS

The specification has been amended. No claims have been amended. Claims 1-30 remain in the application. Reexamination and reconsideration is respectfully requested.

Amendments to the Specification

In the office action mailed 09/12/2007, the examiner required amendments to the specification to provide application numbers corresponding to attorney docket numbers for co-pending applications. Paragraphs [0001], [0019], [0021], [0023], [0024], [0041], [0059] and [0056] in the specification have been amended to substitute patent application serial numbers, published application numbers, and issued patent numbers, for the originally submitted attorney docket numbers referencing co-pending applications. No new matter has been presented. The amendments are submitted to satisfy a formality, and they do not have any effect on the scope of the claims.

Double Patenting Rejections

Claims 1-30 have been rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 1-32 of commonly owned U.S. application 10/973,077 and claims 1-26 of commonly owned U.S. application 10/972,796. Since none of the referenced applications have issued, the applicant will treat this as a provisional double patenting rejection, and no action will be taken at this time. If allowable subject matter is found in at least one of the referenced applications, the applicant will take appropriate action, including a terminal disclaimer if appropriate.

PAGE 10/14 * RCVD AT 12/12/2007 5:22:43 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-1/14 * DNIS:2738300 * CSID:303 297 2266 * DURATION (mm-ss):04-18

Rejections Under 35 U.S.C. § 102

Claims 1-4, 7, 9-23, and 29 have been rejected under 35 U.S.C. § 102(e) as allegedly being anticipated by U.S. Patent 7,085,959 (Safford). Applicant respectfully traverses and requests reconsideration of the Examiner's rejection.

From MPEP 2131, a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference. Applicant respectfully submits that Safford does not describe the boot processor elements of the present claims, as discussed below.

Claims 1, 16, 24, and 29

Independent claim 1 specifies determining that the processor for which said LOL is detected is assigned a role of boot processor; and switching the role of boot processor to a spare processor without shutting down the system's operating system. Independent claim 16 specifies system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system; and switching the role of boot processor to said spare processor. Independent claim 24 specifies system firmware operable, responsive to the detection of LOL for a first of said processor modules, to determine whether said first processor module is assigned a role of system boot processor; wherein if determined that said first processor module is assigned the role of system boot processor, said system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system. Independent claim 29 specifies means for determining whether the processor for which said LOL is detected is assigned the role of boot processor; and means for switching the role of boot processor to a spare processor without shutting down the system's operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor. Safford does not discuss boot processors, the system determining whether a processor is the boot processor, or switching the role of boot processor from one processor to another.

PAGE 11/14 * RCVD AT 12/12/2007 5:22:43 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-1/14 * DNIS:2738300 * CSID:303 297 2266 * DURATION (mm-ss):04-18

In the present application and in Safford, one processor may be halted, and sufficient run-time state information (called architected state in Safford) is transferred from the first processor to a second processor, so that the second processor may resume computation from the point where the first processor is halted. However, changing the designation of which processor is the boot processor requires steps that are different than transferring run-time state, and may be independent from transferring run-time state. That is, changing the designation of which processor is the boot processor may occur without transferring the run-time state. For example, in paragraph [0022], a spare processor that is designated the new boot processor may be idled, so that the process of transferring run-time state as discussed in Safford does not occur at all.

The boot processor is determined by the system, externally from the processors. For example, from paragraphs [0043] through [0045], as the system is booted, all processors are idling (executing do-nothing loops), and the system causes one processor identified by the system (the boot processor) to branch to the entry point of an operating system loader program. In contrast, in Safford, transfer of the architected state of a processor is a transfer of information from one processor to another, as required for one processor to start processing instructions at the point that another processor was halted, while the system is running. Safford does not discuss transfer of system information (which processer is the boot processor) that remains permanent and survives a system reset. Safford discusses rebooting individual processors while a system is running (for example, column 4, lines 57-60, and claim3), but that is not the same as rebooting the entire system, which is initiated by the one designated boot processor after a system reset or power shutdown.

Claims 2-4, 7, 9-15, and 17-23

Claims 2-4, 7, and 9-15 are allowable at least as ultimately depending from allowable base claim 1. Claims 17-23 are allowable at least as ultimately depending from allowable claim 16. Claims 2 and 13-15, 17, and 19-20 are allowable on further independent grounds as discussed below.

PAGE 12/14 * RCVD AT 12/12/2007 5:22:43 PM [Eastern Standard Time] * SVR:USPTO-EFXRF-1/14 * DNIS:2738300 * CSID:303 297 2266 * DURATION (mm-ss):04-18

Claim 2, dependent on claim 1, further specifies the spare processor is a hot spare that is held in idle until switched the role of boot processor. Safford does discuss boot processors or switching to the role of boot processor.

Claim 13, dependent on claim 1, further specifies switching the role of boot processor to said spare processor comprises changing a system identifier of the spare processor to that of the boot processor. Safford does not discuss system identifiers of processors or changing a system identifier to that of the boot processor.

Claim 14, dependent on claim13, further specifies the system identifier is used by the system's operating system to recognize the boot processor. Safford does not discuss an operating system using a system identifier to recognize the boot processor.

Claim 15, dependent on claim 3, further specifies changing a system identifier of said boot processor for which said LOL is detected to a different value, which causes the system's operating system to not recognize the processor for which said LOL is detected as the boot processor. Safford does not discuss changing a system identifier of the boot processor or an operating system not recognizing a processor as the boot processor.

Claim 17, dependent on claim 16, further specifies the spare processor is a hot spare that is held in idle until switched the role of boot processor. Safford does not discuss switching a hot spare to the role of boot processor, or switching a hot spare that is idle to the role of boot processor.

Claim 19, dependent on claim 18, further specifies causing the system's operating system to idle the active spare. Safford does not discuss operating systems or operating systems idling a spare processor.

Claim 20, dependent on claim 16, further specifies said switching causes the system's operating system to recognize said spare processor as said boot processor without shutting down said operating system. Safford does not discuss operating systems or operating systems recognizing a spare processor as the boot processor.

Rejections Under 35 U.S.C. § 103

Claims 5-6, 8, 24-28, and 30 have been rejected under 35 U.S.C. § 103(a) as allegedly being unpatentable over Safford in view of U.S. Patent Number 5,915,082 (Marshall). Applicant respectfully traverses and requests reconsideration of the Examiner's rejection.

Applicant respectfully asserts that Safford is not a valid prior art reference that may be used in a rejection under 35 U.S.C. § 103(a). 35 U.S.C. § 103(c) states:

> (c) Subject matter developed by another person, which qualifies as prior art only under one or more of sub-sections (e), (f), and (g) of section 102 of this title, shall not preclude patentability under this section where the subject matter and the claimed invention were, at the time the invention was made, owned by the same person or subject to an obligation of assignment to the same person.

Please note that Safford and this application are assigned to the same entity, Hewlett-Packard Development Company, L.P. Safford was filed before, but did not issue until after the filing date of the current application. Therefore, the disclosures of Safford are available only as 35 U.S.C. § 102(e)-type prior art. 35 U.S.C. § 103(c) specifies that Safford "shall not preclude patentability" of the claimed invention. Accordingly, the rejection of claims 5-6, 8, 24-28, and 30 should therefore be withdrawn.

In view of the above, it is believed all of the pending claims are now in condition for allowance.

Respectfully submitted,
KLAAS, LAW, O'MEARA & MALKIN, P.C.

By: /Augustus W. Winfield/

Augustus W. Winfield, Reg. No. 34,046
1999 Broadway, Suite 2225
Denver, Colorado 80202
Telephone: (303) 298-9888

Date: December 11, 2007

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

Inventor(s): Scott L. Michaelis et al.

Confirmation No.: 1079

Application No.: 10/972,588

Examiner: Le, Dieu Minh T.

Filing Date: October 25, 2004

Group Art Unit: 2114

RECEIVED
CENTRAL FAX CENTER
DEC 12 2007

Title: SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR

Mail Stop Amendment
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL LETTER FOR RESPONSE/AMENDMENT

Transmitted herewith is/are the following in the above-identified application:

- [X] Response/Amendment
- [ ] New fee as calculated below
- [X] No additional fee
- [ ] Other
- [ ] Petition to extend time to respond
- [ ] Supplemental Declaration

Fee$

CLAIMS AS AMENDED BY OTHER THAN A SMALL ENTITY						
(1) FOR	(2) CLAIMS REMAINING AFTER AMENDMENT	(3) NUMBER EXTRA	(4) HIGHEST NUMBER PREVIOUSLY PAID FOR	(5) PRESENT EXTRA	(6) RATE	(7) ADDITIONAL FEES
TOTAL CLAIMS	30	MINUS	30	= 0	X $50	$ 0
INDEP. CLAIMS	4	MINUS	4	= 0	X $200	$ 0
☐ FIRST PRESENTATION OF A MULTIPLE DEPENDENT CLAIM					+ $360	$ 0
EXTENSION FEE	☐ 1st Month $120	☐ 2nd Month $450	☐ 3rd Month $1020	☐ 4th Month $1590		$ 0
					OTHER FEES	$
					TOTAL ADDITIONAL FEE FOR THIS AMENDMENT	$ 0

Charge $ 0 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees.

[X] A duplicate copy of this transmittal letter is enclosed.

I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571) 273-8300.
Date of facsimile: December 12, 2007

Typed Name: Angela Trousset
Signature: Angela Trousset

Respectfully submitted,
Scott L. Michaelis et al.

By _____
Jay K. Malkin, Esq.
Attorney/Agent for Applicant(s)
Reg No. : 31,393
Date : December 12, 2007
Telephone : (303) 298-9888

Rev 10/06a(TransAmdSnd)

RECEIVED
CENTRAL FAX CENTER
DEC 12 2007

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

Inventor(s): Scott L. Michaelis et al.

Confirmation No.: 1079

Application No.: 10/972,588

Examiner: Le, Dieu Minh T.

Filing Date: October 25, 2004

Group Art Unit: 2114

Title: SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR

Mail Stop Amendment
Commissioner For Patents
PO Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL LETTER FOR RESPONSE/AMENDMENT

Transmitted herewith is/are the following in the above-identified application:

- [X] Response/Amendment
- [ ] New fee as calculated below
- [X] No additional fee
- [ ] Other
- [ ] Petition to extend time to respond
- [ ] Supplemental Declaration

Fee$

CLAIMS AS AMENDED BY OTHER THAN A SMALL ENTITY						
(1) FOR	(2) CLAIMS REMAINING AFTER AMENDMENT	(3) NUMBER EXTRA	(4) HIGHEST NUMBER PREVIOUSLY PAID FOR	(5) PRESENT EXTRA	(6) RATE	(7) ADDITIONAL FEES
TOTAL CLAIMS	30	MINUS	30	= 0	X $50	$ 0
INDEP. CLAIMS	4	MINUS	4	= 0	X $200	$ 0
☐ FIRST PRESENTATION OF A MULTIPLE DEPENDENT CLAIM					+ $360	$ 0
EXTENSION FEE	☐ 1st Month $120	☐ 2nd Month $450	☐ 3rd Month $1020	☐ 4th Month $1590		$ 0
					OTHER FEES	$
					TOTAL ADDITIONAL FEE FOR THIS AMENDMENT	$ 0

Charge $ 0 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees.

[X] A duplicate copy of this transmittal letter is enclosed.

I hereby certify that this paper is being transmitted to the Patent and Trademark Office facsimile number (571) 273-8300.
Date of facsimile: December 12, 2007

Typed Name: Angela Trousset
Signature: Angela Trousset

Respectfully submitted,
Scott L. Michaelis et al.
By
Jay K. Meglin, Esq.
Attorney/Agent for Applicant(s)
Reg No.: 31,393
Date: December 12, 2007
Telephone: (303) 298-9888

Rev 10/06e(TransAmdPac)

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/972,588	10/25/2004	Scott L. Michaelis	200404973-1	1079

22879 7590 09/12/2007

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
LE, DIEU MINH T	

ART UNIT	PAPER NUMBER
2134	

MAIL DATE	DELIVERY MODE
09/12/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/972,588	MICHAELIS ET AL.	
	Examiner	Art Unit	
	Dieu-Minh Le	2114	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *25 October 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-30* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-30* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *10/25/07* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *10/25/07*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Part III DETAILED ACTION

Specification

1. This Office Action is in response to the application 10/972,588 filed on 10/25/04.

2. Claims 1-30 are presented for examination.

3. The disclosure is objected to because of the following informalities: Applicant is required to update all the related applications with the Serial No. or U.S. Patents No. on pages 1, 6-8, 13, 15, 22, and 24.

Appropriate correction is required.

Double Patenting Rejections

4. Claims 1-30 are rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 1-32 of U.S. application 10/973,077; and claims 1-26 of U.S. application 10/972,796. Although the conflicting claims are not identical, they are not patentably distinct from each other because the claimed subject matter contains obvious modifications to previous claims 1-32 of U.S. application 10/973,077; and claims 1-26 of U.S. application 10/972,796.

As to claims 1, 16, 24 and 29, these claims include limitations of: detecting loss of lockstep, detecting a role of boot processor, which already included in claims 1-32 of U.S. application 10/973,077; and claims 1-26 of U.S. application 10/972,796. It is well settled that the omission of an element and its function [i.e.,system firmware] is an obvious expedient if the remaining elements perform the same function as before. In re Karlson, 136, USPQ 184 (CCPA 1963). Also note Ex parte Rainu, 168 USPQ 375 (Bd. App. 1969). Therefore, omitting various elements from the previous claimed subject matter would have been obvious to one of ordinary skill in the art in this case since the remaining elements do in fact perform the same

functions as before. Elimination/Changing of an element or its function will not serve as a basis for patentability.

4. The obviousness-type double patenting rejection is a judicially established doctrine based upon public policy and is primarily intended to prevent prolongation of the patent term by prohibiting claims in a second patent not patentably distinct from claims in a first patent. *In re Vogel*, 164 USPQ 619 (CCPA 1970). A timely filed terminal disclaimer in compliance with 37 C.F.R. § 1.321(b) would overcome an actual or provisional rejection on this ground provided the conflicting application or patent is shown to be commonly owned with this application. See 37 C.F.R. § 1.78(d).

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless -
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section

351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. Claims 1-4, 7, 9-23, and 29 are rejected under 35 U.S.C. 102(e) as being anticipated over Safford et al. (U.S. patent 7,085,959 hereafter referred to as Safford).

As per claim 1:

Safford explicitly teaches the invention. Safford teaches:

- A method comprising:

- detecting loss of lockstep (LOL) for a processor in a multi-processor system [col. 4, lines 13-35 and col. 5, claim 7].

- determining that the processor for which said LOL is detected is assigned a role of boot processor [col. 5, claim 3].

- switching the role of boot processor to a spare processor without shutting down the system's operating system [col. 5, claim 1].

As per claim 2:

Safford explicitly teaches the invention. Safford further teaches:

- wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor [abstract, col. 4, lines 19-29 and col. 5, claim 6].

As per claim 3:

Safford explicitly teaches the invention. Safford further teaches:

- the spare processor is an active spare [col. 5, claim 1].

As per claim 4:

Safford explicitly teaches the invention. Safford further teaches:

- wherein said switching further comprises: causing the system's operating system to idle the active spare [col. 5, claims 1 and 6].

This is clearly shown that Safford's method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] teaching capabilities are corresponding to Applicant's invention.

As per claim 7:

Safford explicitly teaches the invention. Safford further teaches:

- determining the processor that is assigned the role of spare processor [col. 5, claim 1].

As per claim 9:

Safford explicitly teaches the invention. Safford further teaches:

- determining step is performed by system firmware (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51].

As per claim 10:

Safford explicitly teaches the invention. Safford further teaches:

- switching step is performed by system firmware (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51].

As per claim 11:

Safford explicitly teaches the invention. Safford further teaches:

- determining one of the lockstep pair of processors that is not the cause of said LOL [col. 4, lines 30-63 and col. 5, claim 1].

- copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to said spare processor [col. 5, claims 1, 6, and 13].

As per claim 12:

Safford explicitly teaches the invention. Safford further teaches:

- copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to each of said lockstep pair of processors of said spare processor [col. 5, claims 1, 6, and 13].

As per claims 13-15:

Safford explicitly teaches the invention. Safford further teaches:

- changing a system identifier of the spare processor to that of the boot processor [col. 4, lines 30-63 and col. 5, claim 3];

changing a system identifier of said boot processor for which said LOL is detected to a different value, which

causes the system's operating system to not recognize the processor for which said LOL is detected as the boot processor [col. 2, lines 55-67; col. 4, lines 54-63; col. 5, claim 3].

This is clearly shown that Safford's method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] teaching capabilities are corresponding to Applicant's invention.

As per claim 16:

Safford explicitly teaches the invention. Safford further teaches:

- system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51].
- determining one of said lockstep pair of processors that is not the cause of said LOL [col. 4, lines 30-63 and col. 5, claim 1].

- copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to a spare processor [col. 5, claims 1, 6, and 13].

- switching the role of boot processor to said spare processor [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51].

As per claim 17:

Safford explicitly teaches the invention. Safford further teaches:

- wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor [abstract, col. 4, lines 19-29 and col. 5, claim 6].

As per claim 18:

Safford explicitly teaches the invention. Safford further teaches:

- the spare processor is an active spare [col. 5, claim 1].

As per claim 19:

Safford explicitly teaches the invention. Safford further teaches:

- causing the system's operating system to idle the active spare[col. 5, claims 1 and 6].

As per claim 20:

Safford explicitly teaches the invention. Safford further teaches:

- wherein said switching causes the system's operating system to recognize said spare processor as said boot processor without shutting down said operating system [col. 4, lines 30-63 and col. 5, claim 1].

As per claims 21-23:

Safford explicitly teaches the invention. Safford further teaches:

- determining step is performed by system firmware (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51];
- copy step is performed by system firmware (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51];
- switching step is performed by system firmware (i.e., hardware/software) [col. 1, lines 10-31; col. 2, lines 55 through col. 3, lines 51].

As per claim 29:

Due to the similarity of claim 29 to claim 1 except for ***a system*** comprising MEANS for detecting loss of lockstep, MEANS for the determining whether the processor for which LOL is detected, MEANS for switching the role of boot processor, etc… instead of ***a method*** comprising detecting loss of lockstep, determining whether the processor for which LOL is detected, switching the role of boot processor, etc…; therefore, these claims are also rejected under the same rationale applied against claim 1. **In addition, all of the limitations have been noted in the rejection as per claim 1. Such as,** Safford explicilty illustrates ***switching or changing spare processor to an active processor via an architecture state*** (i.e., ***system information state***) [col. 5, claim 1].

This is clearly shown that Safford's method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] teaching capabilities are corresponding to Applicant's invention.

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 5-6, 8, 24-28, and 30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Safford et al. (U.S. patent 7,085,959 hereafter referred to as Safford) in view of Marshall et al. (U.S. patent 5,915,082 hereafter referred to as Marshall).

As per claim 5:

Safford substantially teaches the invention. Safford teaches:

- A method comprising:

- detecting loss of lockstep (LOL) for a processor in a multi-processor system [col. 4, lines 13-35 and col. 5, claim 7];

- spare processor [col. 5, claim 1].

Safford does not explicitly address:

- master and slave processors.

However, Safford does disclose capability of:

- A method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] comprising:

- ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 55-57].

In addition, Marshall explicitly teaches:

- A method comprising:

- An error detection and fault isolation for lockstep processor systems [abstract, col. 1, lines 10-20] comprising:

- a data connectivity among multi-processors, firmware, OS, error detection and recovery, user interface, etc…. [fig. 2, col. 4, lines 13-35];

- ***configuring and error signaling used to enhance recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 11-24 and col. 12, claim 1] via ***master/slave processors*** [abstract, col. 3, lines 1-15].

Therefore, it would have been obvious to a person having ordinary skill in the art at the time of Applicant's invention to first realizing Safford's ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** as being the master and slave processors as claimed by Applicant. This is because Safford's mutli-data processors /fault tolerant system explicitly performed data failure detection and recovery (i.e., ***loss of lockstep*** and/or ***fail-over***) within plurality of processors via ***state information, parity information***, and ***boot information*** processes. By utilizing these capabilities, the multi-data processing system can be directed or redirected promptly and functioned properly during failover recovery process in supporting the system operation; second, by applying the ***configuring and error signaling used to enhance recovery from loss of lockstep in a multi-processors***

system via ***master/slave processors*** as taught by Marshall in conjunction with the method and apparatus for recovery from loss of lock step as taught by Safford, the multi-processors within fault tolerant/failure recovery networking system including master/slave capability (i.e., failover) can enhance its operation performance, more specifically to ensuring the error detected, corrected, and replaced (i.e., backup) in proper and efficient manner via its state information functionality.

This modification would have been obvious because a person having ordinary skill in the art would have been motivated to do so to improve the multi-data processors, failover, and/or fault-tolerant system operation availability and network/system performance therein with a mechanism to enhance the data connectivity, data debugging, data reliability, and data throughput which eventually will increase its performance, such as data throughput between internal and external devices.

As per claims 6, 8, 28, 30:

Safford substantially teaches the invention. Safford teaches:

- wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor [abstract, col. 4, lines 19-29 and col. 5, claim 6];

- the spare processor is an active spare [col. 5, claim 1].
- wherein said switching further comprises: causing the system's operating system to idle the active spare [col. 5, claims 1 and 6].

Safford does not explicitly address:
- information stored to non-volatile data storage and an Itanium Processor Family (IPF) processor.

However, Safford does disclose capability of:
- A method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] comprising:
- ***information stored in the memory*** [col. 1, lines 60 through col. 2, lines 10] and ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 55-57].

In addition, Marshall explicitly teaches:
- A method comprising:
- An error detection and fault isolation for lockstep processor systems [abstract, col. 1, lines 10-20] comprising:

- a data connectivity among multi-processors, firmware, OS, error detection and recovery, user interface, etc…. [fig. 2, col. 4, lines 13-35];

- ***memory used to store data*** [col. 3, lines 35-50] ***in supporting the recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 11-24 and col. 12, claim 1] via ***master/slave processors*** [abstract, col. 3, lines 1-15].

Therefore, it would have been obvious to a person having ordinary skill in the art at the time of Applicant's invention to first realizing Safford's ***information stored in the memory*** and ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** as being information stored to non-volatile data storage and an Itanium Processor Family (IPF) processor as claimed by Applicant. This is because Safford's mutli-data processors /fault tolerant system explicitly performed data failure detection and recovery (i.e., ***loss of lockstep*** and/or ***fail-over***) within plurality of processors via ***state information, parity information***, and ***boot information*** processes; second, by applying the ***memory used to store data in supporting the recovery from loss of lockstep in a multi-processors system*** via ***master/slave processors*** as taught

by Marshall in conjunction with the method and apparatus for recovery from loss of lock step as taught by Safford, the multi-processors within fault tolerant/failure recovery networking system including master/slave capability (i.e., failover) can enhance its operation performance, more specifically to ensuring the error detected, corrected, and replaced (i.e., backup) in proper and efficient manner via its state information functionality.

This modification would have been obvious because a person having ordinary skill in the art would have been motivated to do so to improve the multi-data processors, failover, and/or fault-tolerant system operation availability and network/system performance therein.

As per claim 24:

Safford substantially teaches the invention. Safford teaches:

- A method comprising:
- detecting loss of lockstep (LOL) for a processor in a multi-processor system [col. 4, lines 13-35 and col. 5, claim 7];
- an operating system [col. 5, claim 1].

- error detection logic operable to detect loss of lockstep (LOL) for at least one of said processor modules [col. 4, lines 13-35 and col. 5, claim 7].

- system firmware operable, responsive to the detection of LOL for a first of said processor modules, to determine whether said first processor module is assigned a role of system boot processor [col. 5, claim 3].

- wherein if determined that said first processor module is assigned the role of system boot processor, said system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system [col. 5, claim 1].

Safford does not explicitly address:

- master and slave processors.

However, Safford does disclose capability of:

- A method and apparatus for recovery from loss of lock step [abstract, col. 2, lines 13-32] comprising:

- ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 55-57].

In addition, Marshall explicitly teaches:

- A method comprising:

- An error detection and fault isolation for lockstep processor systems [abstract, col. 1, lines 10-20] comprising:

- a data connectivity among multi-processors, firmware, OS, error detection and recovery, user interface, etc…. [fig. 2, col. 4, lines 13-35];

- ***configuring and error signaling used to enhance recovery from loss of lockstep in a multi-processors system*** [abstract, col. 2, lines 11-24 and col. 12, claim 1] via ***master/slave processors*** [abstract, col. 3, lines 1-15].

Therefore, it would have been obvious to a person having ordinary skill in the art at the time of Applicant's invention to first realizing Safford's ***master/checker pair used in supporting the recovery from loss of lockstep in a multi-processors system*** as being the master and slave processors as claimed by Applicant. This is because Safford's mutli-data processors /fault tolerant system explicitly performed data failure detection and recovery (i.e., ***loss of lockstep*** and/or ***fail-over***) within plurality of processors via ***state information, parity information***, and ***boot information*** processes. By utilizing

these capabilities, the multi-data processing system can be directed or redirected promptly and functioned properly during failover recovery process in supporting the system operation; second, by applying the ***configuring and error signaling used to enhance recovery from loss of lockstep in a multi-processors system*** via ***master/slave processors*** as taught by Marshall in conjunction with the method and apparatus for recovery from loss of lock step as taught by Safford, the multi-processors within fault tolerant/failure recovery networking system including master/slave capability (i.e., failover) can enhance its operation performance, more specifically to ensuring the error detected, corrected, and replaced (i.e., backup) in proper and efficient manner via its state information functionality.

This modification would have been obvious because a person having ordinary skill in the art would have been motivated to do so to improve the multi-data processors, failover, and/or fault-tolerant system operation availability and network/system performance therein with a mechanism to enhance the data connectivity, data debugging, data reliability, and data throughput which eventually will increase its performance, such as data throughput between internal and external devices.

As per claim 25:

Safford substantially teaches the invention. Safford teaches:

- wherein the another processor module is a processor module assigned a role of spare processor [col. 5, claim 1].

As per claim 26:

Safford substantially teaches the invention. Safford teaches:

- wherein the another processor module is held in idle until said system firmware causes said operating system to recognize said another processor module as said system boot processor [abstract, col. 4, lines 19-29 and col. 5, claim 6].

As per claim 27:

Safford substantially teaches the invention. Safford teaches:

- wherein the system firmware causes the operating system to idle the another processor module before causing said operating system to recognize said another processor module as said system boot processor [col. 4, lines 30-63 and col. 5, claim 1].

onclusion

9. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

10. A shortened statutory period for response to this action is set to expired THREE (3) months, ZERO days from the date of this letter. Failure to respond within the period for response will cause the application to be abandoned. 35 U.S.C. 133.

11. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Dieu-Minh Le whose telephone number is (703)305-9408. The examiner can normally be reached on Monday - Thursday from 8:30 AM to 6:30 PM.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Dieu-Minh Le whose telephone number is (571) 272-3660. The examiner can normally be reached on Monday - Thursday from 8:30 AM to 5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Scott Baderman can be reached on (571)272-3644. The Tech Center 2100 phone number is (571) 272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DIEU-MINH THAI LE
PRIMARY EXAMINER
ART UNIT 2114

DML
09/05/07

PATENT APPLICATION

Sheet 1 of 3

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO. 10/972,588	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP 2114	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
AM	1A	5,226,152	07/06/1993	Klug et al.	
	1B	5,249,188	09/28/1993	McDonald	
	1C	5,751,932	05/12/1998	Horst et al.	
	1D	5,758,058	05/26/1998	Milburn	
	1E	5,764,660	06/09/1998	Mohat	
	1F	6,065,135	05/16/2000	Marshall et al.	
	1G	6,148,348	11/14/2000	Garnett et al.	
	1H	6,473,869	10/29/2002	Bissett et al.	
	1I	6,560,682	05/06/2003	Miller et al.	
✓	1J	6,604,177	08/05/2003	Kondo et al.	
AM	1K	6,615,366	09/02/2003	Grochowski et al.	

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	1L					
	1M					
	1N					
	1O					
	1P					

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

AM	1Q	Attorney Docket No. 200404974-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR MAINTAINING IN A MULTI-PROCESSOR SYSTEM A SPARE PROCESSOR THAT IS IN LOCKSTEP FOR USE IN RECOVERING FROM LOSS OF LOCKSTEP FOR ANOTHER PROCESSOR"
AM	1R	Attorney Docket No. 200404944-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR"
AM	1S	Attorney Docket No. 200404946-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION"

EXAMINER	DATE CONSIDERED
David Lade	8/29/07

Rev 05/04 (PTO1449)

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO. 10/972,588	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP 2114	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
DM	2A	6,625,749	09/23/2003	Quach	
	2B	6,687,851	02/03/2004	Somers et al.	
	2C	6,754,787	06/22/2004	Miller et al.	
	2D	2002/0144177	10/03/2002	Kondo et al.	
	2E	2003/0070050	04/10/2003	Miller et al.	
	2F	2004/0006722	01/08/2004	Safford	
↓	2G	2004/0019771	01/29/2004	Quach	
DM	2H	2004/0153857	08/05/2004	Yamazaki et al.	
	2I				
	2J				
	2K				

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	2L					
	2M					
	2N					
	2O					
	2P					

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

DM	2Q	Attorney Docket No. 200404941-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION"
DM	2R	Attorney Docket No. 200404943-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR CONFIGURING LOCKSTEP MODE OF A PROCESSOR MODULE"
DM	2S	Attorney Docket No. 200404971-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR REESTABLISHING LOCKSTEP FOR A PROCESSOR MODULE FOR WHICH LOSS OF LOCKSTEP IS DETECTED"

EXAMINER	DATE CONSIDERED
David Huckabee	8/29/07

Rev 05/04 (PTO1449)

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO. 10/972,588	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP 2114	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	3A				
	3B				
	3C				
	3D				
	3E				
	3F				
	3G				
	3H				
	3I				
	3J				
	3K				

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	3L					
	3M					
	3N					
	3O					
	3P					

OTHER REFERENCES (Including Author, Title, Date, Pertinent Pages, etc.)

AM	3Q	Attorney Docket No. 200404970-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR SYSTEM FIRMWARE CAUSING AN OPERATING SYSTEM TO IDLE A PROCESSOR"
AM	3R	Attorney Docket No. 200404972-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR REINTRODUCING A PROCESSOR MODULE TO AN OPERATING SYSTEM AFTER LOCKSTEP RECOVERY"
	3S	

EXAMINER	DATE CONSIDERED
[signature]	8/29/07

Rev 05/04 (PTO1449)

Notice of References Cited	Application/Control No. 10/972,588	Applicant(s)/Patent Under Reexamination MICHAELIS ET AL.	
	Examiner Dieu-Minh Le	Art Unit 2114	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-7,085,959	08-2006	Safford, Kevin David	714/11
*	B	US-5,915,082	06-1999	Marshall et al.	714/11
*	C	US-7,251,746	07-2007	Fox et al.	714/13
*	D	US-6,625,749	09-2003	Quach, Nhon	714/10
*	E	US-6,920,581	07-2005	Bigbee et al.	714/10
*	F	US-2005/0172164	08-2005	Fox et al.	714/013
*	G	US-2003/0140281	07-2003	Fu et al.	714/38
*	H	US-2005/0240829	10-2005	Safford et al.	714/043
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

Bib Data Sheet

CONFIRMATION NO. 1079

SERIAL NUMBER	FILING OR 371(c) DATE	CLASS	GROUP ART UNIT	ATTORNEY DOCKET NO.
10/972,588	10/25/2004 RULE	714	2114	200404973-1

APPLICANTS

Scott L. Michaelis, Plano, TX;
Anurupa Rajkumari, Round Rock, TX;
William B. McHardy, Fort Colins, CO;

** CONTINUING DATA ********** N/A

** FOREIGN APPLICATIONS ********** N/A

IF REQUIRED, FOREIGN FILING LICENSE GRANTED **
01/14/2005

Foreign Priority claimed	35 USC 119 (a-d) conditions met	Verified and Acknowledged	STATE OR COUNTRY	SHEETS DRAWING	TOTAL CLAIMS	INDEPENDENT CLAIMS
☐ yes ☑ no	☐ yes ☑ no ☐ Met after Allowance	Examiner's Signature / Initials	TX	4	30	4

ADDRESS
22879

TITLE
System and method for switching the role of boot processor to a spare processor responsive to detection of loss of lockstep in a boot processor

FILING FEE RECEIVED	FEES: Authority has been given in Paper
1058	No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:

- ☐ All Fees
- ☐ 1.16 Fees (Filing)
- ☐ 1.17 Fees (Processing Ext. of time)
- ☐ 1.18 Fees (Issue)
- ☐ Other _____
- ☐ Credit

Index of Claims

Application/Control No.	Applicant(s)/Patent under Reexamination
10/972,588	MICHAELIS ET AL.
Examiner	**Art Unit**
Dieu-Minh Le	2114

√	Rejected	–	(Through numeral) Cancelled	N	Non-Elected	A	Appeal
=	Allowed	÷	Restricted	I	Interference	O	Objected

Claim		Date								
Final	Original	9/6/07								
	1	v								
	2	v								
	3	v								
	4	v								
	5	v								
	6	v								
	7	v								
	8	v								
	9	v								
	10	v								
	11	v								
	12	v								
	13	v								
	14	v								
	15	v								
	16	v								
	17	v								
	18	v								
	19	v								
	20	v								
	21	v								
	22	v								
	23	v								
	24	v								
	25	v								
	26	v								
	27	v								
	28	v								
	29	v								
	30	v								
	31									
	32									
	33									
	34									
	35									
	36									
	37									
	38									
	39									
	40									
	41									
	42									
	43									
	44									
	45									
	46									
	47									
	48									
	49									
	50									

Claim		Date								
Final	Original									
	51									
	52									
	53									
	54									
	55									
	56									
	57									
	58									
	59									
	60									
	61									
	62									
	63									
	64									
	65									
	66									
	67									
	68									
	69									
	70									
	71									
	72									
	73									
	74									
	75									
	76									
	77									
	78									
	79									
	80									
	81									
	82									
	83									
	84									
	85									
	86									
	87									
	88									
	89									
	90									
	91									
	92									
	93									
	94									
	95									
	96									
	97									
	98									
	99									
	100									

Claim		Date								
Final	Original									
	101									
	102									
	103									
	104									
	105									
	106									
	107									
	108									
	109									
	110									
	111									
	112									
	113									
	114									
	115									
	116									
	117									
	118									
	119									
	120									
	121									
	122									
	123									
	124									
	125									
	126									
	127									
	128									
	129									
	130									
	131									
	132									
	133									
	134									
	135									
	136									
	137									
	138									
	139									
	140									
	141									
	142									
	143									
	144									
	145									
	146									
	147									
	148									
	149									
	150									

Search Notes

Application/Control No.	Applicant(s)/Patent under Reexamination
10/972,588	MICHAELIS ET AL.
Examiner	**Art Unit**
Dieu-Minh Le	2114

SEARCHED			
Class	Subclass	Date	Examiner
714	10	8/27/2007	DML
	11		
	12		
✓	13	✓	✓
712	31	8/27/2007	DML

INTERFERENCE SEARCHED			
Class	Subclass	Date	Examiner

SEARCH NOTES (INCLUDING SEARCH STRATEGY)		
	DATE	EXMR
EAST search files USPGPUB, USPAT, USOCR, EPO, JPO, DERWENT, IBM-TDB.	8/27/2007	DML
Inventor names search.	8/27/2007	DML

PALM Intranet

Application Number [ ] Submit

IDS Flag Clearance for Application 10972588

IDS Information

Content	Mailroom Date	Entry Number	IDS Review	Last Modified	Reviewer
WIDS	2004-10-25	14	Y ☑	2007-09-07 12:09:30.0	MLe

Update

10/972,588

EAST Search History

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	2222	lockstep lock-step	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:21
L3	25008	(loss adj 1) LOL	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:22
L4	51	(detect$3 or track$4) adj 3	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:23
L5	1547	(boot$4 or initializ$4) adj processor	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:41
L6	246	spare adj processor	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:27
L7	12	4 and 5 and 6	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:30
L8	25	(michaelis-scott$ or myer-sylvia$ or mchardy-william$).in.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:14
L10	632	(714/10).ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:33

EAST Search History

L11	95	(712/31).ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:08
L12	52536	(switch$4 chang$4 failover fail-over "fail over") same (spare slave redundan$4 backup back-up "back up" second$4) same processor	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:27
L13	81	12 same 5	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:28
L14	168219	without same ("shut$4 down" stop$4 idle interrupt$4) same (OS "operating system")	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:33
L15	6	13 and 14	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:33
L16	20342	multi-processor "multi processor" and 3 and 5	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:45
L17	176	16 and 12 and 14	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:46
L18	7	17 and 6	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 16:46

EAST Search History

S1	14	(("7085959") or ("20050172164") or ("5675794") or ("6438687") or ("6920581") or ("5915082") or ("6625749")).PN.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/08/26 17:54
S2	2	("7085959").PN.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2007/08/27 15:05
S3	730	(714/11).ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/27 15:06
S4	337	(714/12).ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/29 15:32
S5	606	(714/13).ccls.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/27 15:07
S6	36	("20020144177"|"20030070050"|"20040006722"|"20040019771"|"20040153857"|"5226152"|"5249188"|"5751932"|"5758058"|"5764660"|"6065135"|"6148348"|"6473869"|"6560682"|"6604177"|"6615366"|"6625749"|"6687851"|"6754787").PN.	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/27 15:08
S7	44	("20020144177"|"20030070050"|"20040006722"|"20040019771"|"20040153857"|"6625749"|"6687851"|"6754787")".PN"	US-PGPUB; USPAT; USOCR; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2007/08/27 15:09

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

22264 U.S. PTO
10/972588
102504

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design
(X) original patent application,
( ) continuation-in-part application

INVENTOR(S): **Scott L. Michaelis et al.**

TITLE: **System And Method For Switching The Role Of Boot Processor To A Spare Processor Responsive To Detection Of Loss Of Lockstep In A Boot Processor**

Enclosed are:

(X) The Declaration and Power of Attorney. (X) signed ( ) unsigned or partially signed

(X) 4 sheets of drawings (one set) ( ) Associate Power of Attorney

( ) Form PTO-1449 (X) Information Disclosure Statement and Form PTO-1449

( ) Priority document(s) ( ) (Other) _____ (fee $_____)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	30 — 20	10	X $18	$ 180
INDEPENDENT CLAIMS	4 — 3	1	X $88	$ 88
ANY MULTIPLE DEPENDENT CLAIMS	0		$300	$ 0
BASIC FEE: Design ($350.00); Utility ($790.00)				$ 790
TOTAL FILING FEE				$ 1,058
OTHER FEES				$
TOTAL CHARGES TO DEPOSIT ACCOUNT				$ 1,058

Charge $ 1,058 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EV 375345140US

Date of Deposit Oct. 25, 2004

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Commissioner for Patents, Alexandria, VA 22313-1450.

By _____

Typed Name: Phyllis Ewing

Respectfully submitted,

Scott L. Michaelis et al.

By _____

Jody C. Bishop

Attorney/Agent for Applicant(s)
Reg. No. 44,034

Date: Oct. 25, 2004

Telephone No.: (214) 855-8007

Rev 10/04 (TransNew)

- Attach as First Page to Transmitted Papers -

Application No. : Not Yet Assigned | Attorney Docket No.: 200404973-1

Certificate of Express Mailing Under 37 CFR 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail, Airbill No. EV 375345140US in an envelope addressed to:

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

on October 25, 2004
Date


Signature

Phyllis Ewing
Typed or printed name of person signing Certificate

Registration Number, if applicable

(214) 855-8000
Telephone Number

Note: Each paper must have its own certificate of mailing, or this certificate must identify each submitted paper.

Assignment (copy) 2 pages
Assignment Recordation Cover Sheet - 2 pages (1 orig. + 1 copy)
Certificate of Express Mail - 1 page

25465991.1

Docket No. 200404973-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR U.S. LETTERS PATENT

Title:

SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR

Inventors:

Scott L. Michaelis
1524 Bamburgh Drive
Plano, TX 75075
Citizenship: U.S.A.

Anurupa Rajkumari
2511 Echo Wood Place
Round Rock, TX 78681
Citizenship: India

William B. McHardy
4023 Sunstone Drive
Fort Collins, CO 80525
Citizenship: U.S.A.

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

25442975.1

SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application is related to the following concurrently filed and commonly assigned U.S. Patent Applications: Serial No. _____ [Attorney Docket No. 200404944-1] titled "SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR"; Serial No. _____ [Attorney Docket No. 200404943-1] titled "SYSTEM AND METHOD FOR CONFIGURING LOCKSTEP MODE OF A PROCESSOR MODULE"; Serial No. _____ [Attorney Docket No. 200404946-1] titled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION"; Serial No. _____ [Attorney Docket No. 200404973-1 titled "SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR"; Serial No. _____ [Attorney Docket No. 200404971-1] titled "SYSTEM AND METHOD FOR REESTABLISHING LOCKSTEP FOR A PROCESSOR MODULE FOR WHICH LOSS OF LOCKSTEP IS DETECTED"; Serial No. _____ [Attorney Docket No. 200404970-1] titled "SYSTEM AND METHOD FOR SYSTEM FIRMWARE CAUSING AN OPERATING SYSTEM TO IDLE A PROCESSOR"; Serial No. _____ [Attorney Docket No. 200404972-1] titled "SYSTEM AND METHOD FOR REINTRODUCING A PROCESSOR MODULE TO AN OPERATING SYSTEM AFTER LOCKSTEP RECOVERY"; and Serial No. _____ [Attorney Docket No. 200404974-1] titled "SYSTEM AND METHOD FOR MAINTAINING IN A MULTI-PROCESSOR SYSTEM A SPARE PROCESSOR THAT IS IN LOCKSTEP FOR USE IN RECOVERING FROM LOSS OF LOCKSTEP FOR ANOTHER PROCESSOR", the disclosures of which are hereby incorporated herein by reference.

DESCRIPTION OF RELATED ART

[0002] Silent Data Corruption ("SDC") is a difficult problem in the computing industry. In general, SDC refers to data that is corrupt, but which the system does not detect as being corrupt. SDCs primarily occur due to one of two factors: a) a broken hardware unit or b) a

"cosmic" event that causes values to change somewhere in the system. Broken hardware means that a "trusted" piece of hardware is silently giving wrong answers. For example, the arithmetic unit in a processor is instructed to add 1+1 and it returns the incorrect answer 3 instead of the correct answer 2. An example of a cosmic event is when a charged particle (e.g., alpha particle or cosmic ray) strikes a region of a computing system and causes some bits to change value (e.g., from a 0 to a 1 or from a 1 to a 0).

[0003] Numerous techniques have been developed for detecting SDC to prevent the SDC from remaining "silent" or "undetected" within a system, as well as preventing such SDC from propagating through the system. Examples of these techniques include parity-based mechanisms and error correcting codes (ECCs) on buses and memory locations, as well as checksums and/or cyclic redundancy checks (CRC) over regions of memory. Parity-based mechanisms are often employed in processors, wherein a parity bit is associated with each block of data when it is stored. The parity bit is set to one or zero according to whether there is an odd or even number of ones in the data block. When the data block is read out of its storage location, the number of ones in the block is compared with the parity bit. A discrepancy between the values indicates that the data block has been corrupted. ECCs are parity-based mechanisms that track additional information for each data block. The additional information allows the corrupted bit(s) to be identified and corrected.

[0004] Parity/ECC mechanisms have been employed extensively for caches, memories, and similar data storage arrays. In the remaining circuitry on a processor, such as data paths, control logic, execution logic, and registers (the "execution core"), it is more difficult to apply parity/ECC mechanisms for SDC detection. Thus, there is typically some unprotected area on a processor in which data corruption may occur and the parity/ECC mechanisms do not prevent the corrupted data from actually making it out onto the system bus. One approach to SDC detection in an execution core (or other unprotected area of the processor chip) is to employ "lockstep processing." Generally, in lockstep processing two processors are paired together, and the two processors perform exactly the same operations and the results are compared (e.g., with an XOR gate). If there is ever a discrepancy between the results of the lockstep processors, an error is signaled. The odds of two processors experiencing the exact same error at the exact same moment (e.g., due to a cosmic event occurring in both processors at exactly the same time or due to a mechanical failure occurring in each processor at exactly the same time) is nearly zero.

[0005] A pair of lockstep processors may, from time to time, lose their lockstep. "Loss of lockstep" (or "LOL") is used broadly herein to refer to any error in the pair of lockstep processors. One example of LOL is detection of data corruption (e.g., data cache error) in one of the processors by a parity-based mechanism and/or ECC mechanism. Another example of LOL is detection of the output of the paired processors not matching, which is referred to herein as a "lockstep mismatch." It should be recognized that in some cases the data in the cache of a processor may become corrupt (e.g., due to a cosmic event), which once detected (e.g., by a parity-based mechanism or ECC mechanism of the processor) results in LOL. Of course, unless such corrupt data is acted upon by the processor, the output of that processor will not fail to match the output of its paired processor and thus a "lockstep mismatch" will not occur. For example, suppose that a value of "1" is stored to first location of cache in each of a pair of lockstep processors and a value of "1" is also stored to a second location of cache in each of the pair of lockstep processors. Further suppose that a cosmic event occurs for a first one of the processors, resulting in the first location of its cache being changed from "1" to "0", and thus corrupted. This data corruption in the first processor is a LOL for the pair. An error detection mechanism of this first processor may detect the data corruption, thus detecting the LOL. If the processors are instructed to act on the data of their first cache locations, then a lockstep mismatch will occur as the output of each of the processors will not match. For instance, if the processors each add the data stored to the first location of their respective cache with the data stored to the second location of their respective cache, the first processor (having the corrupt data) will output a result of "1" (0+1=1) while the second processor outputs a result of "2" (1+1=2), and thus their respective outputs will not match.

[0006] By employing such techniques as parity-based error detection mechanisms and output comparisons for lockstep paired processors, SDC detection can be enhanced such that practically no SDC occurring in a processor goes undetected (and thus such SDC does not remain "silent") but instead results in detection of LOL. However, the issue then becomes how best for the system to respond to detected LOL. The traditional response to detected LOL has been to crash the system to ensure that the detected error is not propagated through the system. That is, LOL in one pair of lockstep processors in a system halts processing of the system even if other processors that have not encountered an error are present in the system. However, with the increased desire for many systems to maintain high availability, crashing the system each time LOL is detected is not an attractive proposition. This is particularly unattractive for large

systems having many processors because cosmic events typically occur more frequently as the processor count goes up, which would result in much more frequent system crashes in those large systems. High availability is a major desire for many customers having large, multi-processor systems, and thus having their system crash every few weeks is not an attractive option. Of course, permitting corrupt data to propagate through the system is also not a viable option.

[0007] Prior solutions attempting to resolve at least some detected SDCs without requiring the system to be crashed have been Operating System ("OS") centric. That is, in certain solutions the OS has been implemented in a manner to recover from a detected LOL without necessarily crashing the system. This OS-centric type of solution requires a lot of processor and platform specific knowledge to be embedded in the OS, and thus requires that the OS provider maintain the OS up-to-date as changes occur in later versions of the processors and platforms in which the OS is to be used. This is such a large burden that most commonly used OSs do not support lockstep recovery.

[0008] Certain solutions have attempted to recover from a LOL without involving the OS in such recovery procedure. For instance, in one technique upon LOL being detected, firmware is used to save the state of one of the processors in a lockstep pair (the processor that is considered "good") to memory, and then both processors of the pair are reset and reinitialized. Thereafter, the state is copied from the memory to each of the processors in the lockstep pair. This technique makes the processors unavailable for an amount of time without the OS having any knowledge regarding this unavailability, and if the amount of time required for recovery is too long, the system may crash. That is, typically, if a processor is unresponsive for X amount of time, the OS will assume that the processor is hung and will crashdump the system so that the problem can be diagnosed. Further, in the event that a processor in the pair cannot be reset and reinitialized (e.g., the processor has a physical problem and fails to pass its self-test), this technique results in crashing the system.

BRIEF SUMMARY OF THE INVENTION

[0009] According to one embodiment, a method comprises detecting loss of lockstep (LOL) for a processor in a multi-processor system. The method further comprises determining that the processor for which the LOL is detected is assigned the role of boot

processor, and switching the role of boot processor to a spare processor without shutting down the system's operating system.

[0010] According to one embodiment, a method comprises system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system. The method further comprises determining one of the lockstep pair of processors that is not the cause of the LOL, and copying the state of the determined one of the lockstep pair of processors that is not the cause of the LOL to a spare processor. The method further comprises switching the role of boot processor to the spare processor.

[0011] According to one embodiment, a system comprises a plurality of processor modules that each include a master processor and a slave processor that operate in lockstep. The system further comprises an operating system, and error detection logic operable to detect loss of lockstep (LOL) for at least one of the processor modules. The system further comprises system firmware operable, responsive to detection of LOL for a first of the processor modules, to determine whether the first processor module is assigned a role of system boot processor. If determined that the first processor module is assigned the role of system boot processor, the system firmware is further operable to cause the operating system to recognize another processor module as system boot processor without shutting down the operating system.

[0012] According to one embodiment, a system comprises means for detecting loss of lockstep (LOL) for a processor in a multi-processor system. The system further comprises means for determining whether the processor for which the LOL is detected is assigned the role of boot processor, and means for switching the role of boot processor to a spare processor without shutting down the system's operating system if determined that the processor for which the LOL is detected is assigned the role of boot processor.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] FIGURE 1 shows an example multi-processor system in which firmware utilizes a spare processor for recovering from LOL for the system's boot processor;

[0014] FIGURE 2 shows a block diagram of one embodiment implemented for the IA-64 processor architecture;

[0015] FIGURE 3 shows an example system having multi-processor cells in which an embodiment for switching the role of boot processor to a spare processor responsive to detected LOL in the boot processor may be employed;

[0016] FIGURE 4 shows an exemplary operational flow diagram for switching the role of boot processor to a hot spare processor according to one embodiment;

[0017] FIGURE 5 shows an exemplary operational flow diagram for switching the role of boot processor to an active spare processor according to one embodiment; and

[0018] FIGURE 6 shows a more general operational flow diagram for certain embodiments.

DETAILED DESCRIPTION OF THE INVENTION

[0019] As described further herein and in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404941-1] titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," the disclosure of which is incorporated herein by reference, certain techniques are provided for recovering from LOL detected for a boot processor in a multi-processor system, in which a spare processor is utilized for such recovery. For instance, upon LOL being detected for the system's boot processor, the spare processor assumes the role of boot processor, and then recovery of lockstep for the original boot processor is attempted. Accordingly, for such lockstep recovery technique, a spare processor is used for attempting to recover from LOL detected for the boot processor. Exemplary techniques are described in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404944-1] titled "SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR," the disclosure of which is incorporated herein by reference, for establishing a spare processor for the boot processor. Such exemplary techniques include techniques for establishing a hot spare processor that is held in reserve until such time as it is used to assume the role of boot processor responsive to a detected LOL in the boot processor. The exemplary techniques further include techniques for establishing an active spare processor that is not held in reserve, but is instead made available to the OS for executing instructions during normal system

operation and is dynamically transformed into a hot spare (i.e., it is idled) such that it can assume the role of the boot processor responsive to detection of LOL in the boot processor.

[0020] Embodiments are provided herein for switching the role of boot processor to an established spare processor responsive to detection of LOL in the boot processor. For instance, exemplary actions by system firmware for switching the role of boot processor to an established spare processor and thus causing the OS to recognize the established spare processor as the system's boot processor are provided in accordance with certain embodiments hereof.

[0021] As described further in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404944-1] titled "SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR," in certain embodiments, information (referred to herein as a "device tree") is stored to a multi-processor system, wherein such information identifies the processor in the system that is the boot processor and identifies the processor that is the spare for the boot processor. The information is stored such that it is accessible by the system's firmware, and the firmware, upon detecting LOL for the boot processor, determines the spare processor from the device tree and assigns the spare processor the role of boot processor. The firmware then attempts to recover lockstep for the processor for which LOL was detected. If the recovery of lockstep is successful, the processor for which lockstep was recovered (i.e., the former boot processor) may be assigned the role of spare for the current boot processor. Thus, the multi-processor system may recover from LOL detected for the boot processor without requiring a crash of the system (at least for certain types of "recoverable" LOLs).

[0022] In some implementations described herein, the spare processor for the boot processor is a "hot" spare, in which it is idled and not made available to the OS for processing instructions during normal system operation. That is, the hot spare processor is held in reserve, idling until such time as it assumes the role of boot processor responsive to detected LOL for the boot processor. In other implementations, the spare processor is not held in reserve, but is instead an "active" processor that is made available to the OS for executing instructions during normal system operation. Upon LOL being detected for the boot processor, the designated spare processor is dynamically transformed into a hot spare (i.e., it is idled) such that it can assume the role of the boot processor. For instance, upon the system's firmware detecting a LOL for the

boot processor, the firmware accesses the device tree and determines the designated spare processor, and then the firmware instructs the OS to idle the designated spare processor (i.e., stop scheduling instructions for the processor) to make the "active" spare a "hot" spare, which then assumes the role of boot processor.

[0023] Concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404941-1] titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," discloses a technique in which lockstep can be recovered for processors (particularly, non-boot processors) without requiring use of a spare processor. Rather, upon LOL being detected for a lockstep pair of processors, the firmware instructs the OS to idle (or "eject") the lockstep pair of processors for which LOL was detected and then the firmware attempts to reestablish lockstep (e.g., by resetting the processors). If successful in reestablishing lockstep, the firmware triggers the OS to recognize the lockstep pair of processors as again being available for servicing instructions.

[0024] However, for various reasons, in certain system architectures problems arise in attempting to idle (or eject) the boot processor from the system. Thus, in certain embodiments, a spare processor is used for recovering from LOL for the system's boot processor. Again, embodiments are provided herein for switching the role of boot processor to an established spare processor responsive to detection of LOL for the boot processor. In certain embodiments, the spare processor is used for recovering from LOL only in the case in which LOL is detected for the boot processor. For instances in which LOL is detected for non-boot processors, the lockstep may be recovered without the use of a spare by using the recovery technique described in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404941-1] titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," for example.

[0025] Turning to FIGURE 1, an example embodiment of a system 10 in which a spare processor is used in recovering from detected loss of lockstep (LOL) for a system boot processor is shown. System 10 includes OS 11, as well as master processor 12A and slave processor 12B (collectively referred to as a lockstep processor pair 12). In certain implementations the lockstep processor pair 12 may be implemented on a single silicon chip, which is referred to as a "dual core processor" in which master processor 12A is a first core and slave processor 12B is a second core. Further, lockstep processor pair 12 may be referred to as a

processor or CPU "module" because it includes a plurality of processors (12A and 12B) in such module. As described further herein, to certain portions of system 10, such as its OS 11, the processor module 12 may appear as a single processor because slave processor 12B is merely used for redundant processing for checking the output of master processor 12A. In this example, lockstep processor pair 12 is designated as the boot processor for system 10. As described further herein, in certain embodiments lockstep processor pair 12 is designated as the system boot processor by storing information to the system that specifies such designation.

[0026] Master processor 12A includes cache 14A, and slave processor 12B includes cache 14B. OS 11 and lockstep processor pair 12 are communicatively coupled to bus 16. Typically, master processor 12A and slave processor 12B are coupled to bus 16 via an interface that allows each of such processors to receive the same instructions to process, but such interface only communicates the output of master processor 12A back onto bus 16. The output of slave processor 12B is used solely for checking the output of mater processor 12A.

[0027] In this example, master processor 12A includes error detect logic 13A, and slave processor 12B includes error detect logic 13B. While shown as included in each of the processors 12A and 12B in this example, in certain embodiments the error detect logic 13A and 13B may be implemented external to processors 12A and 12B. Error detect logic 13A and 13B include logic for detecting errors, such as data cache errors, present in their respective processors 12A and 12B. Examples of error detect logic 13A and 13B include known parity-based mechanisms and ECC mechanisms. Error detect logic 13C is also included, which may include an XOR (exclusive OR) gate, for detecting a lockstep mismatch between master processor 12A and slave processor 12B. As mentioned above, a lockstep mismatch refers to the output of master processor 12A and slave processor 12B failing to match. While shown as external to the lockstep processor pair 12 in this example, in certain embodiments error detect logic 13C may be implemented on a common silicon chip with processors 12A and 12B.

[0028] Lockstep mismatch is one way of detecting a LOL between the master processor 12A and slave processor 12B. A detection of an error by either of error detect logic 13A and 13B also provides detection of LOL in the processors 12A and 12B. Because the detection of LOL by error detect logic 13A and 13B may occur before an actual lockstep mismatch occurs, the detection of LOL by error detect logic 13A and 13B may be referred to as a detection of a "precursor to lockstep mismatch". In other words, once an error (e.g., corrupt

data) is detected by error detect logic 13A or 13B, such error may eventually propagate to a lockstep mismatch error that is detectable by error detect logic 13C.

[0029] System 10 further includes a second lockstep processor pair 15 that is communicatively coupled to bus 16. Lockstep processor pair 15 includes master processor 15A and slave processor 15B. Master processor 15A includes cache 17A, and slave processor 15B includes cache 17B. As with error detect logic 13A-13C implemented for lockstep processor pair 12, lockstep processor pair 15 has error detect logic 16A-16C. While only two lockstep processor pairs 12 and 15 are shown for simplicity in the example of FIGURE 1, system 10 may include any number of such lockstep processor pairs. As one specific example, in one implementation system 10 includes 64 lockstep processor pairs, wherein the master processors of the pairs perform parallel processing for the system. In the specific example of FIGURE 1, lockstep processor pair 15 is designated as a spare for boot processor 12, as described further herein.

[0030] In certain embodiments described herein, techniques are provided for switching the role of boot processor to the designated spare processor responsive to detection of LOL for the boot processor. Thus, the spare processor is used for recovery when LOL is detected for the system's boot processor. That is, the designated spare processor assumes the role of boot processor in the event that LOL is detected for the boot processor. In accordance with certain embodiments, upon LOL being detected for a boot processor, a hot spare processor (i.e., an idling processor that is available in the system) is transferred the role of boot processor, and then the old boot processor having LOL is reset to reestablish its lockstep. In other embodiments, a processor is designated as a spare processor for the boot processor, but such designated spare processor is not held in reserve (idled) during normal system operation. Rather, the designated spare is utilized as an active processing resource made available to OS 11 for processing instructions that OS 11 schedules for such "active" spare processor. Upon LOL being detected for the boot processor, the designated spare processor is idled (thereby, transforming such active processor into a hot spare), and is transferred the role of boot processor. Thereafter, the former boot processor for which the LOL was detected may be reset to reestablish its lockstep. Thus, in certain embodiments, a designated spare processor is not required to be held in reserve, but is instead an available resource for use within the system and is only transformed into an actual spare processor when needed for recovering from LOL for the boot processor.

[0031] Firmware 18 is also included in system 10, which in this embodiment is invoked upon an error being detected by any of the error detect logics 13A-13C and 16A-16C. In certain embodiments, processors 12A, 12B, 15A, and 15B are processors from the Itanium Processor Family (IPF). IPF is a 64-bit processor architecture co-developed by Hewlett-Packard Company and Intel Corporation, which is based on Explicitly Parallel Instruction Computing (EPIC). IPF is a well-known family of processors. IPF includes processors such as those having the code names of MERCED, MCKINLEY, and MADISON. In addition to supporting a 64-bit processor bus and a set of 128 registers, the 64-bit design of IPF allows access to a very large memory (VLM) and exploits features in EPIC. While a specific example implementation of one embodiment is described below for the IPF architecture, embodiments described herein for establishing a spare processor for recovering from LOL detected for a boot processor are not limited in application to an IPF architecture, but may be applied as well to other architectures (e.g., 32-bit processor architectures, etc.).

[0032] Processor architecture generally comprises corresponding supporting firmware, such as firmware 18 of system 10. For example, as described further below in conjunction with the specific example of FIGURE 2, the IPF processor architecture comprises such supporting firmware as Processor Abstraction Layer (PAL), System Abstraction Layer (SAL), and Extended Firmware Interface (EFI). Such supporting firmware may enable, for example, the OS to access a particular function implemented for the processor. For instance, the OS may query the PAL as to the size of the cache implemented for the processor, etc. Other well-known functions provided by the supporting firmware (SAL, EFI) include, for example: (a) performing I/O configuration accesses to discover and program the I/O Hardware (SAL_PCI_CONFIG_READ and SAL_PCI_CONFIG_WRITE); (b) retrieving error log data from the platform following a Machine Check Abort (MCA) event (SAL_GET_STATE_INFO); (c) accessing persistent store configuration data stored in non-volatile memory (EFI variable services: GetNextVariableName, GetVariable and SetVariable); and accessing the battery-backed real-time clock/calendar (EFI GetTime and SetTime). Accordingly, the supporting firmware, such as the PAL, is implemented to provide an interface to the processor(s) for accessing the functionality provided by such processor(s). Each of those interfaces provide standard, published procedure calls that are supported.

[0033] In the example system 10 of FIGURE 1, firmware 18 utilizes the above-mentioned hot spare technique for recovering from LOL for a boot processor. That is, processor

15 is designated as a hot spare for boot processor 12, and such processor 15 is held as an idle, spare processor and used in the manner described further herein for assuming the role of boot processor in the event that a LOL is detected for boot processor 12. Of course, as described further herein, the above-mentioned technique in which a designated spare is not held in reserve but is instead dynamically transformed into a spare processor may be implemented by firmware 18 in alternative embodiments. As described further herein, in certain embodiments, a device tree is stored to a data storage device (e.g., SRAM) such that it is accessible to the firmware 18, wherein the device tree includes information identifying processor 12 as the system's boot processor and identifying processor 15 as the spare. The information specifying the boot processor and/or spare may be received as input from a user in certain embodiments. For instance, in certain embodiments a user interface is provided that enables a system administrator to designate the boot processor and/or spare processor in the multi-processor system. In other embodiments, the firmware 18 selects the boot processor and/or spare.

[0034] Suppose that during the system boot-up procedure, master processor 12A assumes the role of boot processor, as shown in the example of FIGURE 1. In this example embodiment, lockstep processor pair 15 is held as a hot spare for recovering from a LOL that may be detected for the boot processor 12A. Of course, additional lockstep processor pairs may be included in system 10 (not specifically shown in the example of FIGURE 1), and those additional lockstep processor pairs, which are not the system boot processor, may recover from LOL in a different manner that does not utilize a hot spare processor. Thus, in accordance with certain embodiments hereof, hot spare processor pair 15 is not needed for recovering from LOL detected for any non-boot processor, but may instead be used for recovery of LOL only if such LOL is detected for the boot processor (processor 12A in this example).

[0035] Firmware 18 is included in this example, and upon detection of LOL by any of error detect logics 13A-13C, firmware 18 determines, in operational block 101, whether the detected LOL is a recoverable LOL. That is, firmware 18 determines in block 101 whether the detected LOL is of a type from which the firmware can recover lockstep for the lockstep processor pair 12 without crashing the system. As described further herein, lockstep is recoverable for certain detected LOLs (which may be referred to as "recoverable LOLs"), while lockstep is not recoverable for other detected LOLs (which may be referred to as "non-recoverable LOLs"). If the lockstep is not recoverable from the detected LOL, then in the example of FIGURE 1 firmware 18 crashes the system in block 102.

[0036] If the lockstep is recoverable, then operation advances to block 103 whereat the firmware determines whether the processor for which the LOL was detected is the system's boot processor. Because the LOL is detected for the lockstep processor pair 12 in this example, firmware 18 determines whether master processor 12A is the system's boot processor. This can be determined, for example, by accessing the device tree 203 of FIGURE 2 (described further below). The device tree may, in certain implementations, include a field for each processor indicating whether such processor is the system's boot processor. This field in the device tree may be set by the firmware during the boot-up process to identify the corresponding processor that is used as the system's boot processor or it may be pre-set by a system administrator (e.g., via a user interface), as examples. If determined in block 103 that the master processor 12A is not the system's boot processor, then operation advances in this example to block 104, and a lockstep recovery process is performed that does not utilize hot spare 15.

[0037] In the specific example shown in FIGURE 1, if the lockstep is determined to be recoverable in block 101 and the processor for which the LOL is detected is determined in block 103 to not be the system boot processor, firmware 18 cooperates with OS 11 via standard OS methods to recover the lockstep. For instance, in the example embodiment of FIGURE 1, Advanced Configuration and Power Interface (ACPI) methods are used by firmware 18 to cooperate with OS 11. Accordingly, no processor or platform specific knowledge is required to be embedded in OS 11, but instead any ACPI-compatible OS may be used, including without limitation HP-UX and Open VMS operating systems. Example techniques that may be used for recovering from LOL for non-boot processors of a system are described further in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404941-1] titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," the disclosure of which is incorporated herein by reference.

[0038] In the example embodiment of FIGURE 1, if determined in block 103 that the processor for which the LOL is detected is not the system boot processor, firmware 18 triggers OS 11 to idle the master processor 12A in operational block 104. In this embodiment, firmware 18 utilizes an ACPI method 105 to "eject" master processor 12A, thereby triggering OS 11 to idle the master processor 12A (i.e., stop scheduling tasks for the processor). Of course, by idling master processor 12A, slave processor 12B will in turn be idled. Thus, idling master processor 12A results in idling the lockstep processor pair 12. In this example embodiment, OS

11 is not aware of the presence of slave processor 12B, but is instead aware of master processor 12A. The interface of lockstep processor pair 12 to bus 16 manages copying to slave processor 12B the instructions that are directed by OS 11 to master processor 12A. Thus, firmware 18 need not direct OS 11 to eject slave processor 12B, as OS 11 is not aware of such slave processor 12B in this example implementation. Again, by idling master processor 12A, slave processor 12B is also idled as it merely receives copies of the instructions directed to master processor 12A. Of course, if in a given implementation OS 11 is aware of slave 12B, firmware 18 may be implemented to also direct OS 11 to idle such slave processor 12B in a manner similar to that described for idling master processor 12A.

[0039] Firmware 18 then attempts to recover lockstep for the lockstep processor pair 12 in block 106. For instance, firmware 18 resets the processor pair 12. During such reset of processor pair 12, system 10 can continue to operate on its remaining available processors. Once the processor pair 12 is reset and lockstep is recovered, firmware 18 reintroduces master processor 12A to OS 11 in operational block 107. In this embodiment, firmware 18 updates the ACPI device table information for master processor 12A to indicate that such master processor 12A is "present, functioning and enabled." As discussed in the ACPI 2.0 specification for the _STA status method of a device, the _STA (status) object returns the status of a device, which can be one of the following: enabled, disabled, or removed. In this respect, in the result code (bitmap) bit 0 is set if the device is present; bit 1 is set if the device is enabled and decoding its resources; bit 2 is set if the device should be shown in the UI; bit 3 is set if the device is functioning properly (cleared if the device failed its diagnostics); bit 4 is set if the battery is present; and bits 5-31 are reserved. A device can only decode its hardware resources if both bits 0 and 1 are set. If the device is not present (bit 0 cleared) or not enabled (bit 1 cleared), then the device must not decode its resources. Bits 0, 1 and 3 are the "present, enabled and functioning" bits mentioned above. Firmware 18 utilizes an ACPI method 108 to trigger OS 11 to "check for" master processor 12A, thereby reintroducing the master processor 12A to OS 11. As a result of checking for master processor 12A, OS 11 will recognize that such master processor 12A is again available and will thus begin scheduling tasks for master processor 12A once again.

[0040] In view of the above, a recovery technique is provided that does not require utilizing a spare processor for recovering from LOL detected for non-boot processors. For various reasons, in certain system architectures problems arise in attempting to idle (or eject) the boot processor from the system. Thus, in certain embodiments, a hot spare processor is used for

recovering from LOL for the system's boot processor. For instance, in the example of FIGURE 1, if determined in block 103 that the master processor 12A is the system's boot processor, operation advances to block 109 whereat the state of the "good" processor in the lockstep processor pair 12 is copied over to each of processors 15A and 15B in the spare lockstep processor pair 15. Exemplary techniques that may be utilized by firmware 18 for switching the role of boot processor to the hot spare are described further herein below.

[0041] Thus, operational block 109 essentially makes the spare processor pair 15 the system's boot processor, and then in certain embodiments firmware 18 resets the lockstep processor pair 12 and reestablishes its lockstep in block 110. Once lockstep is reestablished for lockstep processor pair 12, that pair is held in operational block 111 as a hot spare for the new boot processor pair 15. That is, firmware 18 updates the device tree 203 of FIGURE 2 to reflect that lockstep processor pair 12 is a hot spare for the boot processor pair 15. Thus, should a recoverable LOL be detected for the new boot processor 15, the above process may be used by firmware 18 to make the spare lockstep processor pair 12 the boot processor and then recover lockstep for pair 15. An example technique for resetting the lockstep processor pair 12 and reestablishing its lockstep in which the lockstep processor pair 12 is then held as a hot spare is described further in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404974-1] titled "SYSTEM AND METHOD FOR MAINTAINING IN A MULTI-PROCESSOR SYSTEM A SPARE PROCESSOR THAT IS IN LOCKSTEP FOR USE IN RECOVERING FROM LOSS OF LOCKSTEP FOR ANOTHER PROCESSOR", the disclosure of which is hereby incorporated herein by reference.

[0042] FIGURE 2 shows a block diagram of one embodiment of the above system 10, which is implemented for the IPF processor architecture and is labeled as system 10_A. The quintessential model of the traditional IPF architecture is given in the Intel IA-64 Architecture Software Developer's Manual, Volume 2: IA-64 System Architecture, in section 11.1 *Firmware Model*, the disclosure of which is hereby incorporated herein by reference. Accordingly, in this example embodiment of system 10_A, firmware 18, labeled as firmware 18_A, includes processor abstraction layer (PAL) 201 and platform/system abstraction layer (SAL) 202. In general, PAL 201 is firmware provided by Intel for its processors, and SAL 202 is developed by an original equipment manufacturer (OEM) for the specific system/platform in which the processors are to be employed. PAL 201, SAL 202, as well as an extended firmware interface (EFI) layer (not

shown), together provide, among other things, the processor and system initialization for an OS boot in an IPF system.

[0043] The boot-up process of a traditional IPF system, for example, proceeds as follows: When the system is first powered on, there are some sanity checks (e.g., power on self-test) that are performed by microprocessors included in the system platform, which are not the main system processors that run applications. After those checks have passed, power and clocks are given to a boot processor (which may, for example, be master processor 12A). The boot processor begins executing code out of the system's Read-Only Memory (ROM) (not specifically shown in FIGURE 2). The code that executes is the PAL 201, which gets control of system 10. PAL 201 executes to acquire all of the processors in system 10_A (recall that there may be many lockstep processor pairs, such as pairs 12 and 15) such that the processors begin executing concurrently through the same firmware.

[0044] After it has performed its duty of initializing the processor(s), PAL 201 passes control of system 10_A to SAL 202. It is the responsibility of SAL 202 to discover what hardware is present on the system platform, and initialize it to make it available for the OS 11. When main memory is initialized and functional, the firmware 18_A is copied into the main memory. Then, control is passed to EFI (not shown), which is responsible for activating boot devices, which typically includes the disk. The EFI reads the disk to load a program into memory, typically referred to as an operating system loader. The EFI loads the OS loader into memory, and then passes it control of system 10_A by branching the boot processor into the entry point of such OS loader program.

[0045] The OS loader program then uses the standard firmware interfaces to discover and initialize system 10_A further for control. One of the things that the OS loader typically has to do in a multi-processor system is to retrieve control of the other processors (those processors other than the boot processor). For instance, at this point in a multi-processor system, the other processors may be executing in do-nothing loops. In an ACPI-compatible system, OS 11 makes ACPI calls to parse the ACPI tables to discover the other processors of a multi-processor system in a manner as is well-known in the art. Then OS 11 uses the firmware interfaces to cause those discovered processors to branch into the operating system code. At that point, OS 11 controls all of the processors and the firmware 18_A is no longer in control of system 10_A.

[0046] As OS 11 is initializing, it has to discover from the firmware 18_A what hardware is present at boot time. And in the ACPI standards, it also discovers what hardware is present or added or removed at run-time. Further, the supporting firmware (PAL, SAL, and EFI) are also used during system runtime to support the processor. For example, OS 11 may access a particular function of master processor 12A via the supporting firmware 18_A, such as querying PAL 201 for the number, size, etc., of the processor's cache 14A. Some other well-known firmware functions that OS 11 may employ during runtime include: (a) PAL 201 may be invoked to configure or change processor features such as disabling transaction queuing (PAL_BUS_SET_FEATURES); (b) PAL 201 may be invoked to flush processor caches (PAL_CACHE_FLUSH); (c) SAL 202 may be invoked to retrieve error logs following a system error (SAL_GET_STATE_INFO, SAL_CLEAR_STATE_INFO); (d) SAL 202 may be invoked as part of hot-plug sequences in which new I/O cards are installed into the hardware (SAL_PCI_CONFIG_READ, SAL_PCI_CONFIG_WRIT); (e) EFI may be invoked to change the boot device path for the next time the system reboots (SetVariable); (f) EFI may be invoked to change the clock/calendar hardware settings; and (g) EFI may be invoked to shutdown the system (ResetSystem).

[0047] A "device tree" is provided, which is shown as device tree 203 in this example. Device tree 203 is stored in SRAM (Scratch RAM) on the cell, which is RAM that is reinitialized. Firmware 18A builds the device tree 203 as it discovers what hardware is installed in the system. Firmware then converts this information to the ACPI tables format and presents it to OS 11 so that OS 11 can know what is installed in the system. The ACPI device tables (not shown) are only consumed by OS 11 at boot time, so they are never updated as things change. For OS 11 to find the current status, it calls an ACPI "method" to discover the "current status". The _STA method described above is an example of such an ACPI method. When _STA is called, the AML can look for properties on the device specified in the firmware device tree and convert that into the Result Code bitmap described above. So, if lockstep has been lost on a processor, firmware 15A will set the device tree property that indicates loss of lockstep, then when OS 11 calls _STA for that device, the "lockstep lost" property directs the AML code to return to "0" in the "functioning properly" bit so that OS 11 can know there is a problem with that processor.

[0048] If a lost lockstep is recovered in accordance with the recovery technique described herein, firmware 18A can indicate that lockstep has been recovered in the device tree

203. Then when _STA is called on that device responsive to the OS receiving the "check for device" ACPI method, the present, enabled and functioning bits will all be set and OS 11 will know the CPU is safe to use.

[0049] Further, device tree 203 includes information designating which processor is the boot processor, as well as a spare processor for the boot processor. A simple example of device tree 203 is shown below in Table 1:

Device	Status	Lockstep Enabled	Boot Processor?	Spare for Boot Processor?
Processor A	Present, Enabled, and Functioning	Yes	Yes	No
Processor B	N/A	Yes	No	Yes
Processor C	Present, Enabled, and Functioning	Yes	No	No
...	...	...	...	...

Table 1

[0050] In the above example of Table 1, 3 processors "Processor A," "Processor B," and "Processor C" are identified in the device tree as being included in the multi-processor system. The status of each of processors Processor A and Processor B is identified as "Present, Enabled, and Functioning." In this example Processor B is a hot spare, so it is held in reserve and does not show up in the static ACPI tables that are built when the system boots. Accordingly the OS does not know about Processor B and does not have a device handle to call the _STA method on. As described above, the _STA method is a known ACPI method for checking the status of the corresponding device. Therefore, the status of Processor B is designated N/A (not applicable) in this example because the _STA method will not be called for this processor as long as it is the hot spare. Of course, upon such spare assuming the role of boot processor, its status is updated to "Present, Enabled, and Functioning." In other embodiments in which Processor B is an active spare, the OS will be aware of such processor and accordingly its status may be also indicated as "Present, Enabled, and Functioning."

[0051] Further, lockstep is identified in this example as enabled for these processors. In certain embodiments, the system administrator can designate, via user interface 20, whether lockstep mode is to be enabled for the system's processors. Further, Processor A is designated as the system's boot processor in the example device tree shown in Table 1, and Processor B is designated as the spare processor. Thus, upon detection of LOL for one of the

processors in the system, the firmware can access this device tree to determine if the processor for which LOL is detected is the system's boot processor; and if determined that the processor for which LOL is detected is the system's boot processor, the firmware can further determine from this device tree the designated spare processor to use in the lockstep recovery process.

[0052] In the example of FIGURE 2, a user interface 20 is included, which allows a user (e.g., system administrator) to designate the system's boot processor and spare processor. In certain embodiments, the user interface 20 further allows the user to designate other information, such as whether lockstep mode is enabled for the processors.

[0053] One example system in which firmware 18_A of FIGURE 2 may be employed is shown in FIGURE 3. FIGURE 3 shows system 30 that includes multi-processor cells A-D, labeled 31_A-31_D, respectively. Cell A 31_A is shown in more detail, and should be understood that cells B-D 31_B-31_D have substantially the same architecture as that of cell A 31_A in this example. The architecture of cell A 31_A described hereafter corresponds to that of Hewlett-Packard's Superdome™ systems. As shown, multi-processor cell A 31_A includes coherency controller 304 that is communicatively coupled to two buses, labeled 305 and 306, respectively. A plurality of processor modules are included within cell A 31_A, shown as CPU 0, CPU 1, CPU 2 and CPU 3 (and labeled 300-303, respectively). More particularly, a plurality of processors modules are communicatively coupled to each of the buses 305 and 306. In this example, CPU 0 (300) and CPU 1 (301) are each communicatively coupled to the first bus 305, and CPU 2 (302) and CPU 3 (303) are each communicatively coupled to the second bus 306. It should be understood that each of CPU 0 (300), CPU 1 (301), CPU 2 (302), and CPU 3 (303) is in actuality a lockstep processor pair, such as the lockstep processor pair 12 of FIGURES 1-3. Thus, for instance, CPU 0 (300) includes both a master processor and a slave processor, as with processors 12A and 12B included in the lockstep processor pair 12 of FIGURES 1-2.

[0054] Coherency controller 304 decodes the address an interrupt is targeted towards and determines which bus (305 or 306) the interrupt packet should be delivered to. Each CPU on the destination bus sees the interrupt and compares the interrupt target address with its internal "LID" register to determine if it is the targeted CPU. If the match is correct, the CPU responds to the interrupt. If the match is not correct, the CPU ignores the interrupt packet.

[0055] As shown, the example firmware 18_A described above with FIGURES 1-2 may be employed to manage the recovery from detected LOLs for the processors of cells A-D

(31_A-31_D). In this example architecture, when designating a spare for the boot processor, such spare is selected to be another processor communicatively coupled to the same bus as the boot processor. For instance, suppose that during the boot-up of system 30, CPU 2 (302) is designated as the boot processor. Accordingly, firmware 18_A will identify another processor that is communicatively coupled to bus 306 to designate as a spare for such boot processor, such as CPU 3 (303). During runtime if a LOL is detected for CPU 2 (302), then firmware 18_A may operate according to the process described above in FIGURE 1 to switch the designation of boot processor to a designated spare, CPU 3 (303). In performing this switch the LID register of the CPU that lost lockstep is copied into the LID register of the spare CPU and the spare CPU pair will then assume the ID of the failed CPU and start responding to interrupts directed to that ID. The LID of the failed CPU is quickly re-written to a different value so that it will no longer respond to interrupts directed to that ID. CPU 2 (302) can then be reset to recover its lockstep, and such CPU 2 (302) may then be designated as a spare for the new boot processor CPU 3 (303).

[0056] It should be recognized that in the example system of FIGURE 3, if a hot spare were used for recovering from LOL for all processors, rather than just for the boot processor, much of the potential processing resources would be required to be held in reserve as spare resources. For instance, a hot spare would be required on each of the buses of each cell. Because there are two processors coupled to each bus in this example (i.e., CPU 0 and CPU 1 coupled to bus 305, and CPU 2 and CPU 3 coupled to bus 306), reserving a hot spare processor for each bus would result in holding half of the total processing resources in reserve. This is an undesirably expensive and wasteful solution. Thus, according to the illustrated embodiment, a hot spare processor is reserved only for the boot processor. Thus, continuing with the above example in which CPU 2 is the boot processor and CPU 3 is held as a "hot" spare, both of CPU 0 and CPU 1 are active (i.e., neither are held in spare). Further, none of the processors of cells B-D need to be held in reserve as hot spares. In the event of a recoverable LOL occurring for any of non-boot processors of the cells, the lockstep recovery process of FIGURE 1 (i.e., operational blocks 104-108) can be utilized without requiring a spare for lockstep recovery for those non-boot processors.

[0057] While in the above example embodiment, a hot spare CPU is maintained for recovery from LOL for the system's boot processor, in an alternative embodiment a hot spare need not be held as such, but rather a processor can be dynamically made a "spare" when needed

for recovery from LOL encountered for the system boot processor. That is, assuming as in the above example of FIGURE 3 that CPU 2 is the system's boot processor, CPU 3 does not have to be held in reserve (idled) as a hot spare for CPU 2, but instead CPU 3 can be active and dynamically idled and made a "spare" upon detection of LOL in CPU 2. Accordingly, CPU 3 may be designated as a spare for the boot processor, but such CPU 3 need not be held in reserve during normal system operation. Instead, the CPU 3 may be dynamically (during system run-time) transformed into a spare processor (e.g., by causing the OS to idle such processor) in the event that a LOL is detected for the boot processor.

[0058] Thus, the resource of this spare CPU need not be wasted during normal runtime, but can be dynamically turned into a spare processor that is available for recovering from LOL encountered on CPU 2 in the manner described above. As an example of this embodiment, the firmware may send an "eject request" on a healthy processor pair (e.g., CPU 3), and lie to the OS by indicating (e.g., in response to an ACPI _STA method for the healthy processor pair) that such processor pair is not functioning. This would cause the OS to eject that processor pair as though it had encountered a LOL. This processor would become the "hot spare" and be used to replace the boot processor. The system boot processor role would be assumed by the idled healthy processor pair, just as described above in the example in which the healthy CPU 3 is held idle for the system boot processor, CPU 2. The only difference being that now the firmware had to take some initial action to turn the healthy processor pair into a "hot spare" by idling it and making it appear to the OS as though it was not functioning. Thus, the LIDs would be swapped. That is, the LID register of the boot processor that lost lockstep is copied into the LID register of the now spare processor pair (CPU 3 in this example), and the LID register of this spare processor pair would be copied into the LID register of the original boot processor. Lockstep is then reestablished on the original boot processor, and it can be reintroduced to the OS with the ID of the original non-boot processor that replaced it as the system boot processor. The processor that was formerly the boot processor may be designated, in device tree 203, as the spare for the now boot processor.

[0059] Turning to FIGURE 4, an exemplary operational flow diagram for switching the role of boot processor to a hot spare processor according to one embodiment is shown. In this case, an LOL has been detected for the boot processor. In operational block 401, the system firmware allocates space to save the status of the good processor of the lockstep pair of processors for which LOL was detected. Exemplary techniques for determining which of the

lockstep pair of processors had the error are disclosed in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404946-1] titled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION," the disclosure of which is hereby incorporated herein by reference. Accordingly, an appropriate amount of data storage space for storing the cache of the "good" processor in the lockstep pair of processors is allocated in block 401. For instance, an amount of SRAM may be allocated for storing the registers from the good processor.

[0060] In operational block 402, the state of the good processor is stored to the allocated storage space. For instance, in this illustrative example, PAL is called to dump the processor state of the good processor of the current boot processor (CBP) module (for which LOL was detected) to the allocated storage space.

[0061] In operational block 403, the LID of the current boot processor (for which LOL was detected) is reprogrammed to a different value. For instance, the current boot processor may be assigned a LID that is reserved just for this situation (i.e., for switching the role of boot processor). Accordingly, the boot processor for which LOL was detected will not respond to the interrupts addressed for its original LID because the boot processor's LID value is reprogrammed to a different value.

[0062] In operational block 404, the state of the good processor that was stored in block 402 to the allocated storage space is stored to the hot spare processor. In this example, the PAL is called to "Restore Processor State" on the hot spare processor, wherein the PAL is provided a pointer to the address to which the good processor's state was stored in block 402. Accordingly, all the values from that storage space will be read into the registers of the spare processor. As a result, the LID value of the spare processor is programmed with the value of the boot processor that was saved to the allocated storage space. Accordingly, the spare processor automatically, as a result of receiving this stored state, receives the Local ID corresponding to the boot processor for interrupt handling.

[0063] In operational block 405, the system firmware updates its device tree information to reflect that the hot spare is assuming the role of boot processor. The system firmware further updates its device tree, in this example, to indicate that the former boot

processor (for which LOL was detected) is assuming the role of hot spare for the new boot processor.

[0064] In operational block 406, PAL_MC_RESUME (which is a standard PAL call for resuming normal execution after a machine check (MC)) is called for the hot spare (which is assuming the role of boot processor) to cause the OS to recognize the hot spare processor as available for processing instructions, wherein the OS recognizes the hot spare as the boot processor. Thus, the hot spare has now assumed the role of the boot processor, taking the place of the former boot processor for which LOL was detected. Accordingly, in operational block 407 the former hot spare (which has assumed the role of boot processor) resumes normal operation as the system's boot processor with its lockstep protection enabled. Thus, in this exemplary embodiment, upon detection of LOL for a boot processor, the role of boot processor is switched to a hot spare that has lockstep protection, wherein the system effectively recovers lockstep for its boot processor without requiring a system shut down (i.e., without shutting down the OS). In certain implementations, the firmware may go on to take action to attempt to reestablish lockstep for the former boot processor (e.g., by resetting the former boot processor), and if successful in reestablishing its lockstep, such former boot processor may be held in reserve as a hot spare processor for the new boot processor in the event that LOL is detected for this new boot processor.

[0065] Turning to FIGURE 5, an exemplary operational flow diagram for switching the role of boot processor to an active spare processor according to one embodiment is shown. Again, an LOL has been detected for the boot processor. In operational block 501, the system firmware, responsive to such detection of LOL for the boot processor, examines the device tree to determine which processor module is assigned the role of active spare. In operational block 502 the system firmware (e.g., SAL) asserts a General Purpose Event Interrupt (GPE) by writing to the appropriate GPE register in the ACPI register space. This causes the OS to execute its interrupt handler, which will run an AML Method associated with this particular interrupt. In this example, this AML method that is executed responsive to the interrupt generated by the GPE includes an AML instruction called "Notify". This "Notify" operation indicates the determined active spare processor object and that an Eject is being requested as arguments. Accordingly, an AML method is used to request the OS to eject the active spare that was determined in block 501. In response, the OS idles the active spare processor and returns control over the active spare processor to the system firmware, in block 503.

[0066] In operational block 504, the system firmware allocates space to save the status of the good processor of the lockstep pair of processors for which LOL was detected (i.e., the lockstep pair of processors of the CBP). Exemplary techniques for determining which of the lockstep pair of processors had the error are disclosed in concurrently filed and commonly assigned U.S. Patent Application Serial No. _____ [Attorney Docket No. 200404946-1] titled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION," the disclosure of which is hereby incorporated herein by reference. Accordingly, an appropriate amount of data storage space for storing the cache of the "good" processor in the lockstep pair of processors is allocated in block 504. For instance, an amount of SRAM may be allocated for storing the registers from the good processor.

[0067] In operational block 505, the state of the good processor is stored to the allocated storage space. For instance, in this illustrative example, PAL is called to dump the processor state of the good processor of the current boot processor (CBP) module (for which LOL was detected) to the allocated storage space.

[0068] In operational block 506, the LID of the current boot processor (for which LOL was detected) is reprogrammed to a different value. For instance, the current boot processor may be assigned a LID that is reserved just for this situation (i.e., for switching the role of boot processor). Accordingly, the boot processor for which LOL was detected will not respond to the interrupts addressed for its original LID because the boot processor's LID value is reprogrammed to a different value.

[0069] In operational block 507, the state of the good processor that was stored in block 505 to the allocated storage space is stored to the idled spare processor. In this example, the PAL is called to "Restore Processor State" on the active spare processor, wherein the PAL is provided a pointer to the address to which the good processor's state was stored in block 505. Accordingly, all the values from that storage space will be read into the registers of the spare processor. As a result, the LID value of the spare processor is programmed with the value of the boot processor that was saved to the allocated storage space. Accordingly, the spare processor automatically, as a result of receiving this stored state, receives the Local ID (LID) corresponding to the boot processor for interrupt handling.

[0070] In operational block 508, the system firmware updates its device tree information to reflect that the active spare is assuming the role of boot processor. The system firmware further updates its device tree, in this example, to indicate that the former boot processor (for which LOL was detected) is assuming the role of active spare for the new boot processor.

[0071] In operational block 509, PAL_MC_RESUME (which is a standard PAL call for resuming normal execution after a machine check (MC)) is called for the active spare (which is assuming the role of boot processor) to cause the OS to recognize the active spare processor as available for processing instructions, wherein the OS recognizes the active spare as the boot processor. Thus, the active spare has now assumed the role of the boot processor, taking the place of the former boot processor for which LOL was detected. Accordingly, in operational block 510 the former active spare (which has assumed the role of boot processor) resumes normal operation as the system's boot processor with its lockstep protection enabled. Thus, in this exemplary embodiment, upon detection of LOL for a boot processor, an active spare processor is idled, and the role of boot processor is switched to such idled spare that has lockstep protection, wherein the system effectively recovers lockstep for its boot processor without requiring a system shut down (i.e., without shutting down the OS). In certain implementations, the firmware may go on to take action to attempt to reestablish lockstep for the former boot processor (e.g., by resetting the former boot processor), and if successful in reestablishing its lockstep, such former boot processor may be reintroduced to the OS as an active spare processor for the new boot processor in the event that LOL is detected for this new boot processor.

[0072] The exemplary procedures of FIGURES 4 and 5 switch the role of boot processor to a spare processor in accordance with certain embodiments. While exemplary operational flows are provided in FIGURES 4 and 5, embodiments hereof are not limited to those illustrative examples. For instance, FIGURE 6 shows a more general operational flow diagram for certain embodiments. In operational block 601, LOL is detected for a processor in a multi-processor system. In operational block 602, the system (e.g., system firmware) determines whether the processor for which LOL is detected is the system's boot processor. As described above, this is determined in certain embodiments by accessing the node of the device tree corresponding to the processor for which LOL is detected and determining if the information in that node of the device tree indicates that the processor is the system's boot processor.

[0073] In operational block 603, responsive to determining that the processor for which LOL is detected is the system's boot processor, the system's firmware switches the role of boot processor to a spare processor, which may be either a hot spare or an active spare, as described above. Accordingly, the system's OS recognizes the spare processor as the system boot processor. In view of the above, embodiments hereof enable the role of boot processor to be dynamically switched, during system runtime, to a spare processor responsive to a detected LOL for the boot processor, thus enabling recovery from the LOL without requiring that the system be crashed or re-booted.

CLAIMS

What is claimed is:

1. A method comprising:

detecting loss of lockstep (LOL) for a processor in a multi-processor system;

determining that the processor for which said LOL is detected is assigned a role of boot processor; and

switching the role of boot processor to a spare processor without shutting down the system's operating system.

2. The method of claim 1 wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor.

3. The method of claim 1 wherein the spare processor is an active spare.

4. The method of claim 3 wherein said switching further comprises:

causing the system's operating system to idle the active spare.

5. The method of claim 1 wherein the spare processor is a processor module that includes a master and slave processors that operate in lockstep.

6. The method of claim 1 wherein said determining that the processor for which said LOL is detected is assigned the role of boot processor comprises:

accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of boot processor.

7. The method of claim 1 further comprising:

determining the processor that is assigned the role of spare processor.

8. The method of claim 7 wherein said determining the processor that is assigned the role of spare processor comprises:

accessing information stored to non-volatile data storage that identifies one of the processors in said multi-processor system that is assigned the role of spare processor.

9. The method of claim 1 wherein said determining step is performed by system firmware.

10. The method of claim 1 wherein said switching step is performed by system firmware.

11. The method of claim 1 wherein the processor for which said LOL is detected includes a lockstep pair of processors, and wherein said switching comprises:

determining one of the lockstep pair of processors that is not the cause of said LOL; and

copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to said spare processor.

12. The method of claim 11 wherein said spare processor includes a lockstep pair of processors, and wherein said copying comprises:

copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to each of said lockstep pair of processors of said spare processor.

13. The method of claim 1 wherein said switching the role of boot processor to said spare processor comprises:

changing a system identifier of the spare processor to that of the boot processor.

14. The method of claim 13 wherein the system identifier is used by the system's operating system to recognize the boot processor.

15. The method of claim 13 further comprising:

changing a system identifier of said boot processor for which said LOL is detected to a different value, which causes the system's operating system to not recognize the processor for which said LOL is detected as the boot processor.

16. A method comprising:

system firmware determining that loss of lockstep (LOL) is detected for a lockstep pair of processors that are assigned the role of boot processor in a system;

determining one of said lockstep pair of processors that is not the cause of said LOL;

copying the state of the determined one of said lockstep pair of processors that is not the cause of said LOL to a spare processor; and

switching the role of boot processor to said spare processor.

17. The method of claim 16 wherein the spare processor is a hot spare that is held in idle until switched the role of boot processor.

18. The method of claim 16 wherein the spare processor is an active spare.

19. The method of claim 18 further comprising:

causing the system's operating system to idle the active spare.

20. The method of claim 16 wherein said switching causes the system's operating system to recognize said spare processor as said boot processor without shutting down said operating system.

21. The method of claim 16 wherein said determining step is performed by system firmware.

22. The method of claim 16 wherein said copying step is performed by system firmware.

23. The method of claim 16 wherein said switching step is performed by system firmware.

24. A system comprising:

a plurality of processor modules that each include a master processor and a slave processor that operate in lockstep;

an operating system;

error detection logic operable to detect loss of lockstep (LOL) for at least one of said processor modules; and

system firmware operable, responsive to the detection of LOL for a first of said processor modules, to determine whether said first processor module is assigned a role of system boot processor;

wherein if determined that said first processor module is assigned the role of system boot processor, said system firmware is further operable to cause said operating system to recognize another processor module as system boot processor without shutting down said operating system.

25. The system of claim 24 wherein the another processor module is a processor module assigned a role of spare processor.

26. The system of claim 24 wherein the another processor module is held in idle until said system firmware causes said operating system to recognize said another processor module as said system boot processor.

27. The system of claim 24 wherein the system firmware causes the operating system to idle the another processor module before causing said operating system to recognize said another processor module as said system boot processor.

28. The system of claim 24 further comprising:

non-volatile data storage storing information that assigns the role of system boot processor to one of said plurality of processor modules.

29. A system comprising:

means for detecting loss of lockstep (LOL) for a processor in a multi-processor system;

means for determining whether the processor for which said LOL is detected is assigned the role of boot processor; and

means for switching the role of boot processor to a spare processor without shutting down the system's operating system if determined that the processor for which said LOL is detected is assigned the role of boot processor.

30. The system of claim 29 wherein said processor in said multi-processor system is an Itanium Processor Family (IPF) processor.

ABSTRACT

According to one embodiment, a method comprises detecting loss of lockstep (LOL) for a processor in a multi-processor system. The method further comprises determining that the processor for which the LOL is detected is assigned the role of boot processor, and switching the role of boot processor to a spare processor without shutting down the system's operating system. In another embodiment, a method comprises system firmware determining that an LOL is detected for a lockstep pair of processors that are assigned the role of boot processor in a system. The method further comprises determining one of the lockstep pair of processors that is not the cause of the LOL, and copying the state of the determined one of the lockstep pair of processors that is not the cause of the LOL to a spare processor. The method further comprises switching the role of boot processor to the spare processor.

FIG. 1

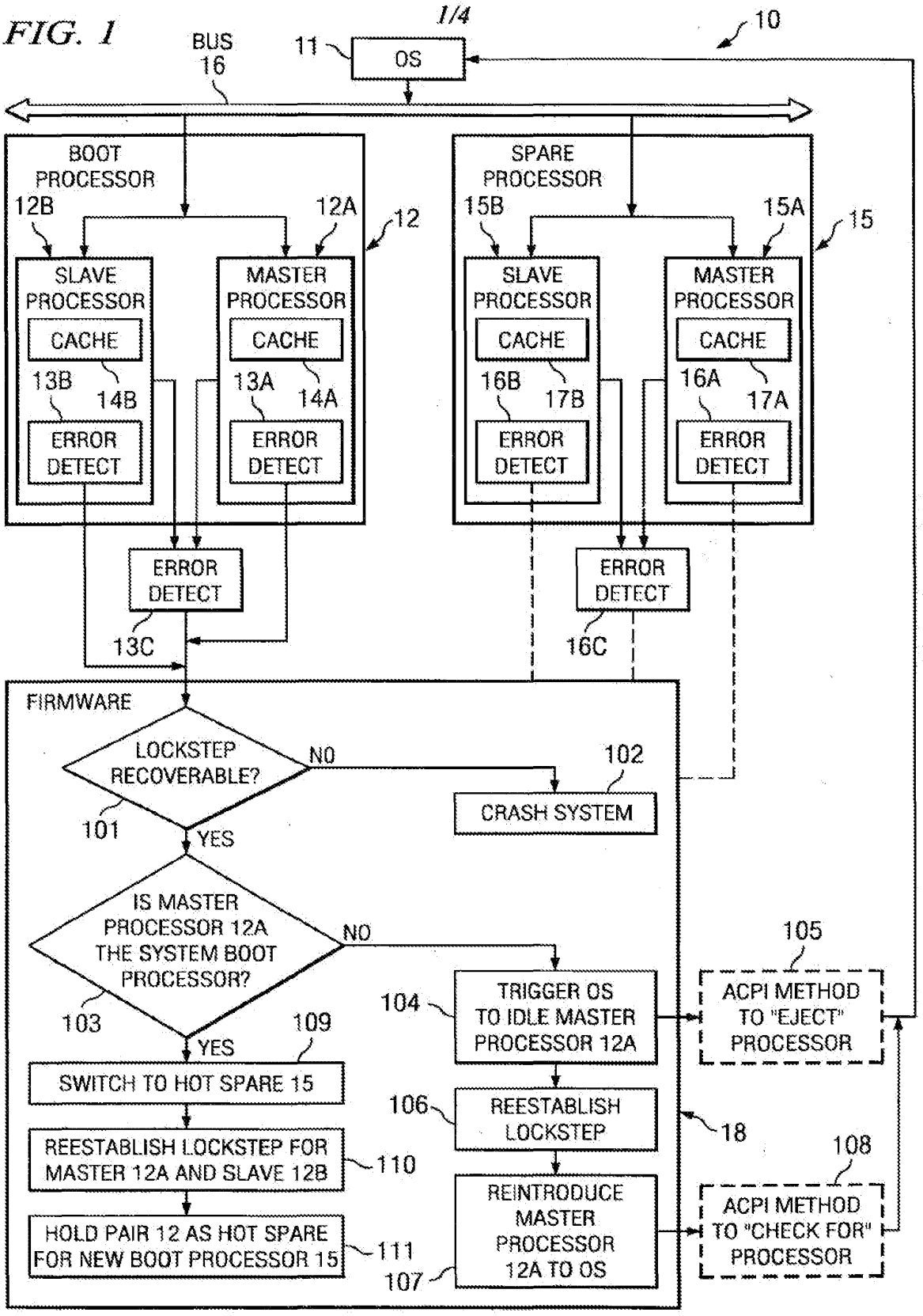

BUS
16
11
OS
10
BOOT PROCESSOR
12B
12A
12
SPARE PROCESSOR
15B
15A
15
SLAVE PROCESSOR
CACHE
13B
14B
ERROR DETECT
MASTER PROCESSOR
CACHE
13A
14A
ERROR DETECT
SLAVE PROCESSOR
CACHE
16B
17B
ERROR DETECT
MASTER PROCESSOR
CACHE
16A
17A
ERROR DETECT
ERROR DETECT
13C
ERROR DETECT
16C
FIRMWARE
LOCKSTEP RECOVERABLE?
NO
102
CRASH SYSTEM
101
YES
IS MASTER PROCESSOR 12A THE SYSTEM BOOT PROCESSOR?
NO
105
103
YES
109
104
TRIGGER OS TO IDLE MASTER PROCESSOR 12A
ACPI METHOD TO "EJECT" PROCESSOR
SWITCH TO HOT SPARE 15
106
REESTABLISH LOCKSTEP
18
REESTABLISH LOCKSTEP FOR MASTER 12A AND SLAVE 12B
110
108
REINTRODUCE MASTER PROCESSOR 12A TO OS
ACPI METHOD TO "CHECK FOR" PROCESSOR
HOLD PAIR 12 AS HOT SPARE FOR NEW BOOT PROCESSOR 15
111
107

FIG. 2

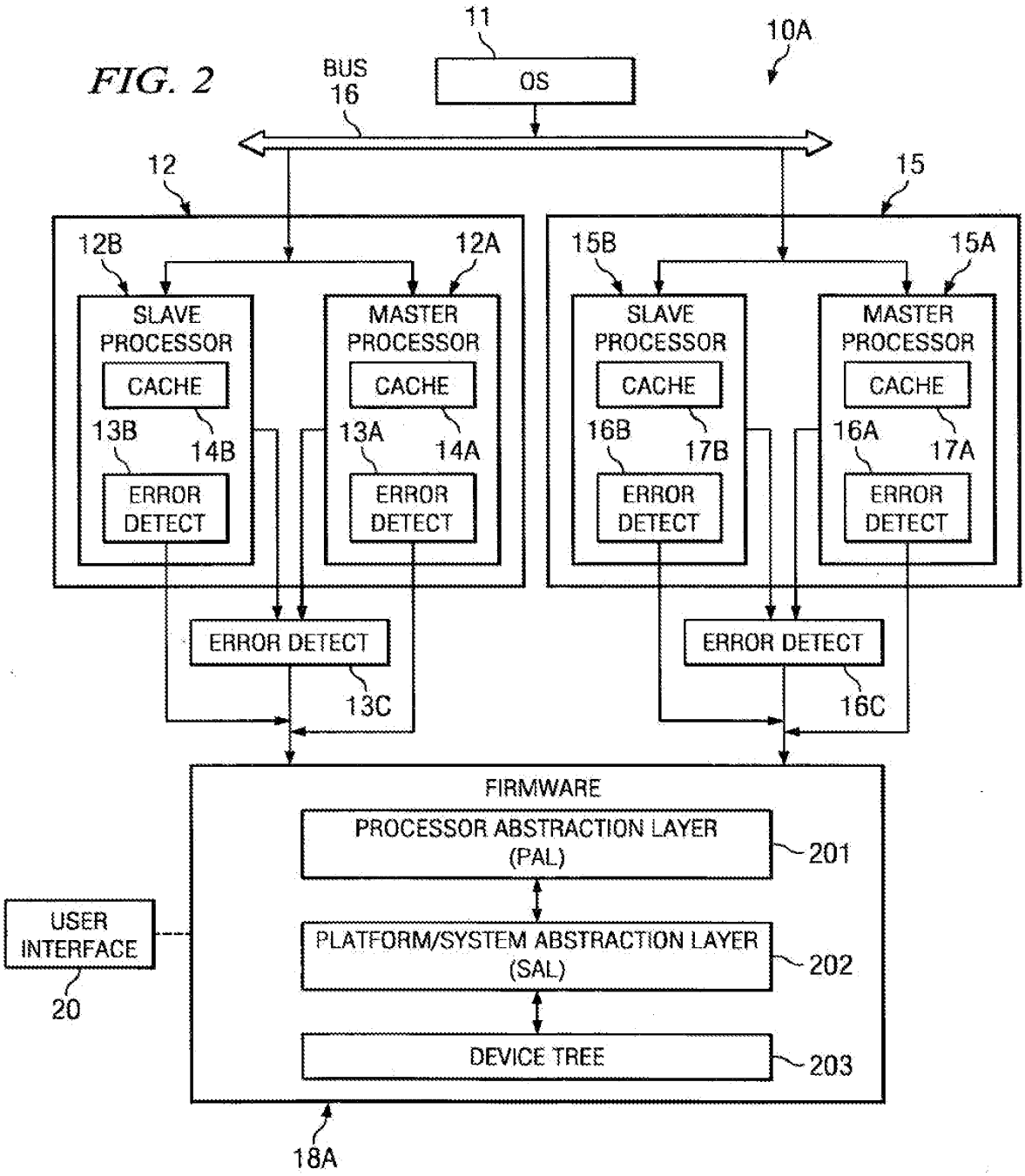

11
10A
BUS
16
OS
12
15
12B
12A
15B
15A
SLAVE PROCESSOR
CACHE
13B
14B
ERROR DETECT
MASTER PROCESSOR
CACHE
13A
14A
ERROR DETECT
SLAVE PROCESSOR
CACHE
16B
17B
ERROR DETECT
MASTER PROCESSOR
CACHE
16A
17A
ERROR DETECT
ERROR DETECT
13C
ERROR DETECT
16C
FIRMWARE
PROCESSOR ABSTRACTION LAYER (PAL)
201
PLATFORM/SYSTEM ABSTRACTION LAYER (SAL)
202
DEVICE TREE
203
USER INTERFACE
20
18A

FIG. 3

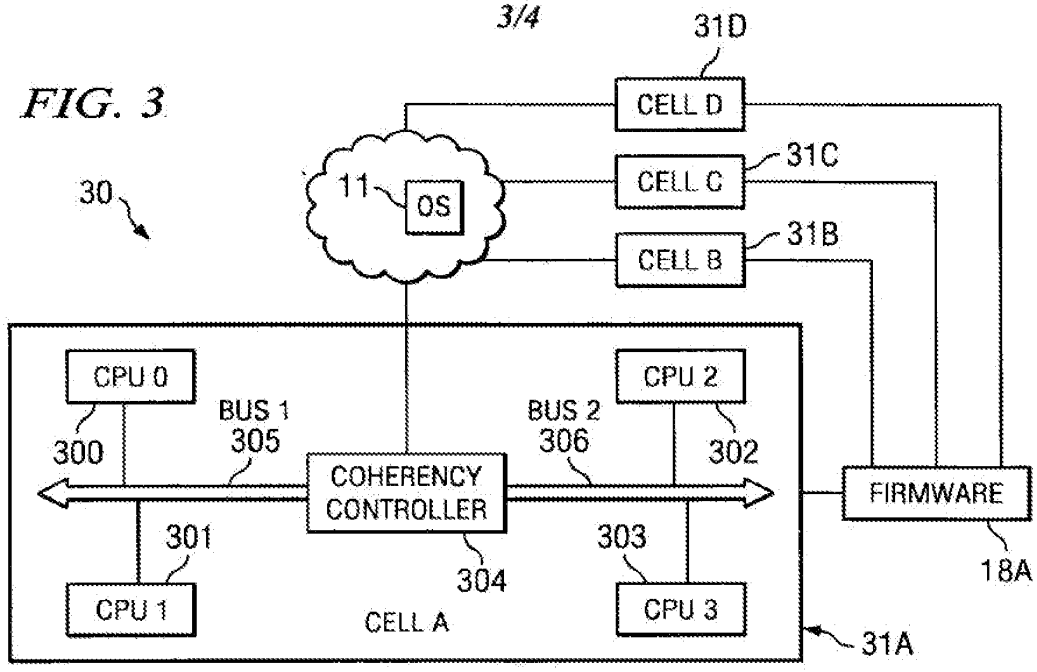

31D
CELL D
31C
CELL C
31B
CELL B
30
11
OS
CPU 0
300
BUS 1
305
CPU 2
BUS 2
306
302
COHERENCY CONTROLLER
FIRMWARE
18A
301
CPU 1
304
303
CPU 3
CELL A
31A

FIG. 4

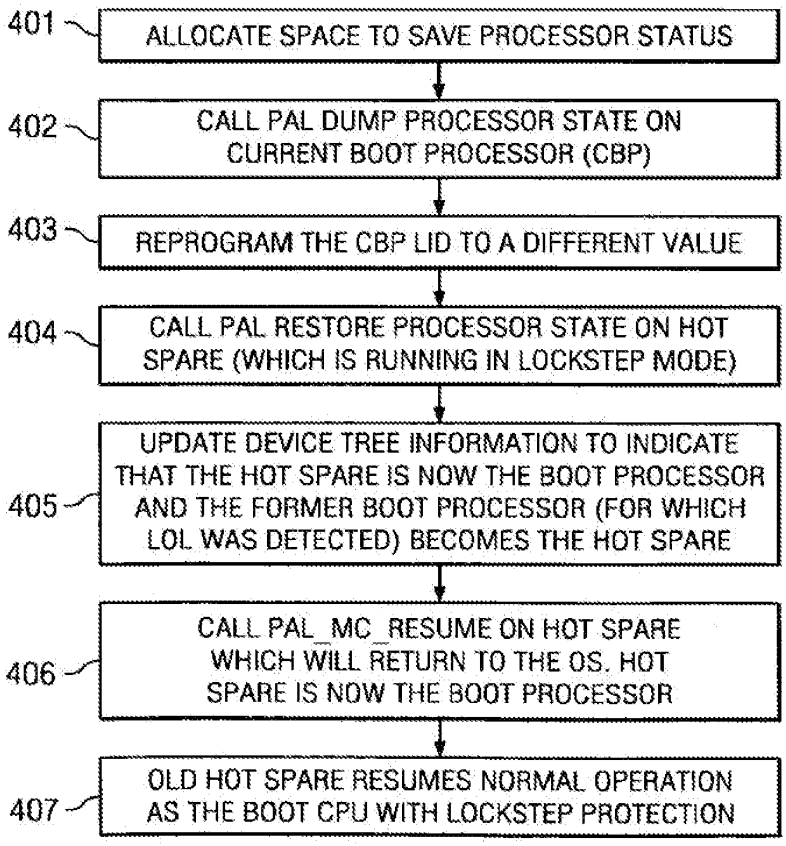

401
ALLOCATE SPACE TO SAVE PROCESSOR STATUS
402
CALL PAL DUMP PROCESSOR STATE ON CURRENT BOOT PROCESSOR (CBP)
403
REPROGRAM THE CBP LID TO A DIFFERENT VALUE
404
CALL PAL RESTORE PROCESSOR STATE ON HOT SPARE (WHICH IS RUNNING IN LOCKSTEP MODE)
405
UPDATE DEVICE TREE INFORMATION TO INDICATE THAT THE HOT SPARE IS NOW THE BOOT PROCESSOR AND THE FORMER BOOT PROCESSOR (FOR WHICH LOL WAS DETECTED) BECOMES THE HOT SPARE
406
CALL PAL_MC_RESUME ON HOT SPARE WHICH WILL RETURN TO THE OS. HOT SPARE IS NOW THE BOOT PROCESSOR
407
OLD HOT SPARE RESUMES NORMAL OPERATION AS THE BOOT CPU WITH LOCKSTEP PROTECTION

FIG. 5

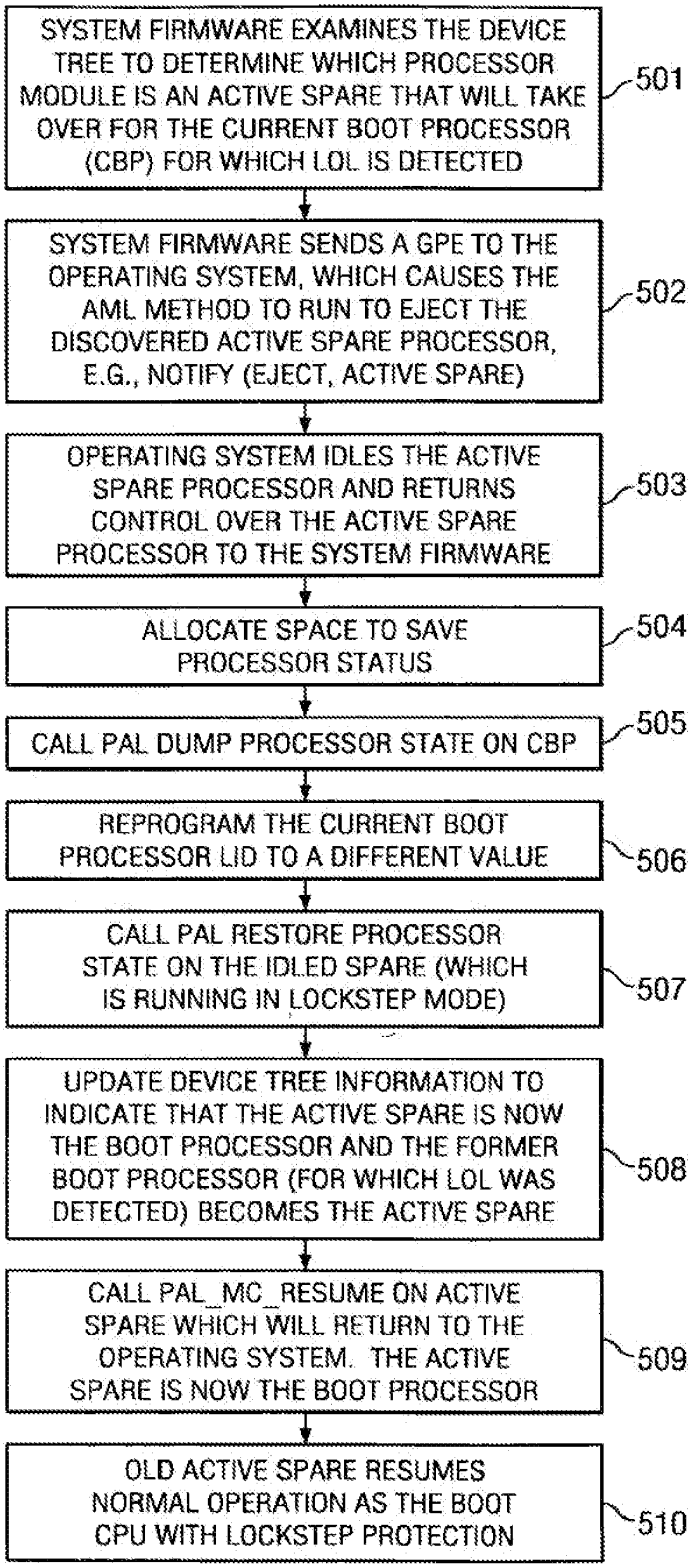
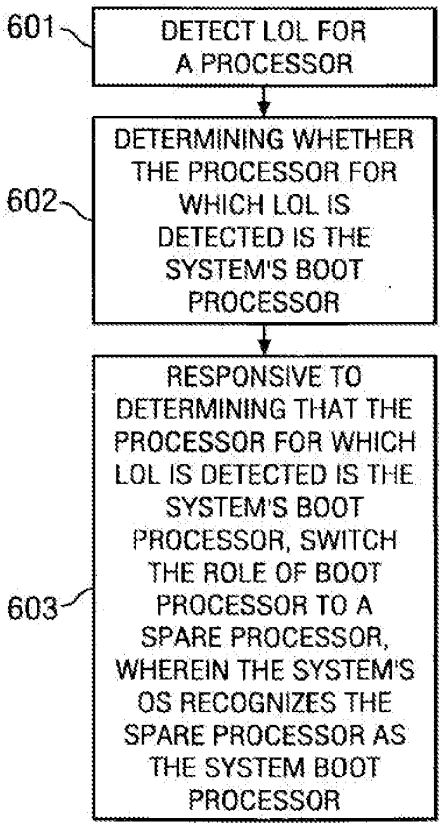

SYSTEM FIRMWARE EXAMINES THE DEVICE TREE TO DETERMINE WHICH PROCESSOR MODULE IS AN ACTIVE SPARE THAT WILL TAKE OVER FOR THE CURRENT BOOT PROCESSOR (CBP) FOR WHICH LOL IS DETECTED
501
SYSTEM FIRMWARE SENDS A GPE TO THE OPERATING SYSTEM, WHICH CAUSES THE AML METHOD TO RUN TO EJECT THE DISCOVERED ACTIVE SPARE PROCESSOR, E.G., NOTIFY (EJECT, ACTIVE SPARE)
502
OPERATING SYSTEM IDLES THE ACTIVE SPARE PROCESSOR AND RETURNS CONTROL OVER THE ACTIVE SPARE PROCESSOR TO THE SYSTEM FIRMWARE
503
ALLOCATE SPACE TO SAVE PROCESSOR STATUS
504
CALL PAL DUMP PROCESSOR STATE ON CBP
505
REPROGRAM THE CURRENT BOOT PROCESSOR LID TO A DIFFERENT VALUE
506
CALL PAL RESTORE PROCESSOR STATE ON THE IDLED SPARE (WHICH IS RUNNING IN LOCKSTEP MODE)
507
UPDATE DEVICE TREE INFORMATION TO INDICATE THAT THE ACTIVE SPARE IS NOW THE BOOT PROCESSOR AND THE FORMER BOOT PROCESSOR (FOR WHICH LOL WAS DETECTED) BECOMES THE ACTIVE SPARE
508
CALL PAL_MC_RESUME ON ACTIVE SPARE WHICH WILL RETURN TO THE OPERATING SYSTEM. THE ACTIVE SPARE IS NOW THE BOOT PROCESSOR
509
OLD ACTIVE SPARE RESUMES NORMAL OPERATION AS THE BOOT CPU WITH LOCKSTEP PROTECTION
510

FIG. 6


601
DETECT LOL FOR A PROCESSOR
602
DETERMINING WHETHER THE PROCESSOR FOR WHICH LOL IS DETECTED IS THE SYSTEM'S BOOT PROCESSOR
603
RESPONSIVE TO DETERMINING THAT THE PROCESSOR FOR WHICH LOL IS DETECTED IS THE SYSTEM'S BOOT PROCESSOR, SWITCH THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR, WHEREIN THE SYSTEM'S OS RECOGNIZES THE SPARE PROCESSOR AS THE SYSTEM BOOT PROCESSOR

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

System And Method For Switching The Role Of Boot Processor To A Spare Processor Responsive To Detection Of Loss Of Lockstep In A Boot Processor

the specification of which is attached hereto unless the following box is checked:

( ) was filed on _____ as US Application No. or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
			YES: ___ NO: ___
			YES: ___ NO: ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION NUMBER	FILING DATE

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Send Correspondence to:

Customer Number 022879

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

Direct Telephone Calls to:

Richard Stokey

(281) 518-3184

Even though the attorney is listed, please associate HP's Customer Number 022879 with this case.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Scott L. Michaelis Citizenship: U.S.A.

Residence: 1524 Bamburgh Drive, Plano, TX 75075

Post Office Address: 1524 Bamburgh Drive, Plano, TX 75075

Scott L Michaelis
Inventor's Signature

10/20/2004
Date

DECLARATION AND POWER OF ATTORNEY
FOR PATENT APPLICATION (continued)

ATTORNEY DOCKET NO. 200404973-1

Full Name of # 2 joint inventor: Anurupa Rajkumari Citizenship: India

Residence: 2511 Echo Wood Place, Round Rock, TX 78681

Post Office Address: 2511 Echo Wood Place, Round Rock, TX 78681

Anurupa Rajkumari
Inventor's Signature

10/20/04
Date

Full Name of # 3 joint inventor: William B. McHardy Citizenship: U.S.A.

Residence: 4023 Sunstone Drive, Fort Collins, CO 80525

Post Office Address: 4023 Sunstone Drive, Fort Collins, CO 80525

William McHardy
Inventor's Signature

10/19/04
Date

Full Name of # 4 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 5 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 6 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 7 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 8 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

10/27/2004 GWORDOF1 00000043 082025 10972588

01 FC:1001 790.00 DA
02 FC:1201 88.00 DA
03 FC:1202 180.00 DA

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/59033

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2004

Application or Docket Number: 10/972,588

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS 30		
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	30 minus 20=	* 10
INDEPENDENT CLAIMS	4 minus 3 =	* 1
MULTIPLE DEPENDENT CLAIM PRESENT		☐

* If the difference in column 1 is less than zero, enter "0" in column 2

SMALL ENTITY TYPE ☐		OR	OTHER THAN SMALL ENTITY	
RATE	FEE		RATE	FEE
BASIC FEE	395.00	OR	BASIC FEE	790.00
X$ 9=		OR	X$18=	180
X44=		OR	X88=	88
+150=		OR	+300=	
TOTAL		OR	TOTAL	1,058

CLAIMS AS AMENDED - PART II

AMENDMENT A

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

1, 16, 24, 29

SMALL ENTITY		OR	OTHER THAN SMALL ENTITY	
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT B

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

		OR		
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

AMENDMENT C

	(Column 1) CLAIMS REMAINING AFTER AMENDMENT		(Column 2) HIGHEST NUMBER PREVIOUSLY PAID FOR	(Column 3) PRESENT EXTRA
Total	*	Minus	**	=
Independent	*	Minus	***	=
FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM				☐

		OR		
RATE	ADDITIONAL FEE		RATE	ADDITIONAL FEE
X$ 9=		OR	X$18=	
X44=		OR	X88=	
+150=		OR	+300=	
TOTAL ADDIT. FEE		OR	TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.
** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."
***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."
The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

FORM PTO-875 (Rev. 10/04) — Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Scott L. Michaelis et al. **Confirmation No.:**

Application No.: **Examiner:**

Filing Date: **Group Art Unit:**

Title: SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTION OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR

Mail Stop Amendment
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

INFORMATION DISCLOSURE STATEMENT

Sir:

This Information Disclosure Statement is submitted:

(X) under 37 CFR 1.97(b), or
(Within three months of filing national application; or date of entry of national application; or before mailing date of first office action on the merits; whichever occurs last)

( ) under 37 CFR 1.97(c) together with either a:
( ) Statement under 37 CFR 1.97(e), or
( ) a $180.00 fee under 37 CFR 1.17(p), or
(After the CFR 1.97 (b) time period, but before final action or notice of allowance, whichever occurs first)

( ) under 37 CFR 1.97 (d) together with a:
( ) Statement under 37 CFR 1.97(e)(1) or (2), and
( ) a $180.00 fee set forth in 37 CFR 1.17(p).
(Filed after final action, a notice of allowance, on or before payment of the issue fee)

Please charge to Deposit Account **08-2025** the sum of $0.00 . At any time during the pendency of this application, please charge any fees required or credit any overpayment to Deposit Account **08-2025** pursuant to 37 CFR 1.25.

(X) Applicant(s) submit herewith Form PTO 1449 - Information Disclosure Statement together with any required copies of patents, publications or other information of which applicant(s) are aware, which applicant(s) believe(s) may be material to the examination of this application and for which there may be a duty to disclose in accordance with 37 CFR 1.56.

( ) A concise explanation of the relevance of foreign language patents, foreign language publications and other foreign language information listed on PTO Form 1449, as presently understood by the individuals(s) designated in 37 CFR 1.56 (c) most knowledgeable about the content is given on the attached sheet, or where a foreign language patent is cited in a search report or other action by a foreign patent office in a counterpart foreign application, an English language version of the search report or action which indicates the degree of relevance found by the foreign office is listed on form PTO 1449 and is enclosed herewith.

It is requested that the information disclosed herein be made of record in this application.

"Express Mail" label no. **EV 375345140US**

Date of Deposit **Oct. 25, 2004**

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Commissioner for Patents, Alexandria, VA 22313-1450.

By _____

Typed Name: **Phyllis Ewing**

Respectfully submitted,

Scott L. Michaelis et al.

By _____
Jody C. Bishop

Attorney/Agent for Applicant(s)
Reg. No. **44,034**

Date: **Oct. 25, 2004**

Telephone No.: **(214) 855-8007**

Rev 10/04 (IDSXML)

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO.	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	1A	5,226,152	07/06/1993	Klug et al.	
	1B	5,249,188	09/28/1993	McDonald	
	1C	5,751,932	05/12/1998	Horst et al.	
	1D	5,758,058	05/26/1998	Milburn	
	1E	5,764,660	06/09/1998	Mohat	
	1F	6,065,135	05/16/2000	Marshall et al.	
	1G	6,148,348	11/14/2000	Garnett et al.	
	1H	6,473,869	10/29/2002	Bissett et al.	
	1I	6,560,682	05/06/2003	Miller et al.	
	1J	6,604,177	08/05/2003	Kondo et al.	
	1K	6,615,366	09/02/2003	Grochowski et al.	

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	1L					
	1M					
	1N					
	1O					
	1P					

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

	1Q	Attorney Docket No. 200404974-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR MAINTAINING IN A MULTI-PROCESSOR SYSTEM A SPARE PROCESSOR THAT IS IN LOCKSTEP FOR USE IN RECOVERING FROM LOSS OF LOCKSTEP FOR ANOTHER PROCESSOR"
	1R	Attorney Docket No. 200404944-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR"
	1S	Attorney Docket No. 200404946-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION"

EXAMINER	DATE CONSIDERED

Rev 05/04 (PTO1449)

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO.	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	2A	6,625,749	09/23/2003	Quach	
	2B	6,687,851	02/03/2004	Somers et al.	
	2C	6,754,787	06/22/2004	Miller et al.	
	2D	2002/0144177	10/03/2002	Kondo et al.	
	2E	2003/0070050	04/10/2003	Miller et al.	
	2F	2004/0006722	01/08/2004	Safford	
	2G	2004/0019771	01/29/2004	Quach	
	2H	2004/0153857	08/05/2004	Yamazaki et al.	
	2I				
	2J				
	2K				

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	2L					
	2M					
	2N					
	2O					
	2P					

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

	2Q	Attorney Docket No. 200404941-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION"
	2R	Attorney Docket No. 200404943-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR CONFIGURING LOCKSTEP MODE OF A PROCESSOR MODULE"
	2S	Attorney Docket No. 200404971-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR REESTABLISHING LOCKSTEP FOR A PROCESSOR MODULE FOR WHICH LOSS OF LOCKSTEP IS DETECTED"

EXAMINER	DATE CONSIDERED

Rev 05/04 (PTO1449)

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 200404973-1	APPLICATION NO.	CONFIRMATION NO.
	APPLICANT Scott L. Michaelis et al.		
	FILING DATE	GROUP	

REFERENCE DESIGNATION **U.S. PATENT DOCUMENTS**

EXAMINER INITIAL		DOCUMENT NUMBER	PUBLICATION DATE	NAME	Pages, Columns, Lines Where Relevant Passages or Figures Appear
	3A				
	3B				
	3C				
	3D				
	3E				
	3F				
	3G				
	3H				
	3I				
	3J				
	3K				

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	PUBLICATION DATE	NAME OF PATENTEE OR APPLICANT	Pages/Columns/Lines Where Relevant Passages/Figures Appear	Check if Translation attached
	3L					
	3M					
	3N					
	3O					
	3P					

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

	3Q	Attorney Docket No. 200404970-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR SYSTEM FIRMWARE CAUSING AN OPERATING SYSTEM TO IDLE A PROCESSOR"
	3R	Attorney Docket No. 200404972-1, filed concurrently herewith, entitled "SYSTEM AND METHOD FOR REINTRODUCING A PROCESSOR MODULE TO AN OPERATING SYSTEM AFTER LOCKSTEP RECOVERY"
	3S	

EXAMINER	DATE CONSIDERED

Rev 05/04 (PTO1449)

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 200404973-1

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

22264 U.S. PTO
10/972588
102504

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design
(X) original patent application,
( ) continuation-in-part application

INVENTOR(S): **Scott L. Michaelis et al.**

TITLE: **System And Method For Switching The Role Of Boot Processor To A Spare Processor Responsive To Detection Of Loss Of Lockstep In A Boot Processor**

Enclosed are:

(X) The Declaration and Power of Attorney. (X) signed ( ) unsigned or partially signed

(X) 4 sheets of drawings (one set) ( ) Associate Power of Attorney

( ) Form PTO-1449 (X) Information Disclosure Statement and Form PTO-1449

( ) Priority document(s) ( ) (Other) _____ (fee $_____)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	30 — 20	10	X $18	$ 180
INDEPENDENT CLAIMS	4 — 3	1	X $88	$ 88
ANY MULTIPLE DEPENDENT CLAIMS	0		$300	$ 0
BASIC FEE: Design ($350.00); Utility ($790.00)				$ 790
TOTAL FILING FEE				$ 1,058
OTHER FEES				$
TOTAL CHARGES TO DEPOSIT ACCOUNT				$ 1,058

Charge $ 1,058 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EV 375345140US

Date of Deposit Oct. 25, 2004

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Commissioner for Patents, Alexandria, VA 22313-1450.

By _____

Typed Name: Phyllis Ewing

Respectfully submitted,

Scott L. Michaelis et al.

By _____

Jody C. Bishop

Attorney/Agent for Applicant(s)
Reg. No. 44,034
Date: Oct. 25, 2004

Telephone No.: (214) 855-8007

Rev 10/04 (TransNew) - Attach as First Page to Transmitted Papers -

Application No. : Not Yet Assigned | Attorney Docket No.: 200404973-1

Certificate of Express Mailing Under 37 CFR 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail, Airbill No. EV 375345140US in an envelope addressed to:

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

on October 25, 2004
Date

Signature

Phyllis Ewing
Typed or printed name of person signing Certificate

Registration Number, if applicable

(214) 855-8000
Telephone Number

Note: Each paper must have its own certificate of mailing, or this certificate must identify each submitted paper.

Assignment (copy) 2 pages
Assignment Recordation Cover Sheet - 2 pages (1 orig. + 1 copy)
Certificate of Express Mail - 1 page

25465991.1

EXHIBIT
1010

intel

Extensible Firmware Interface Specification

Version 1.10

December 1, 2002

intel

1 Introduction

This *Extensible Firmware Interface* (hereafter known as EFI) *Specification* describes an interface between the operating system (OS) and the platform firmware. The interface is in the form of data tables that contain platform-related information, and boot and runtime service calls that are available to the OS loader and the OS. Together, these provide a standard environment for booting an OS.

The EFI specification is designed as a pure interface specification. As such, the specification defines the set of interfaces and structures that platform firmware must implement. Similarly, the specification defines the set of interfaces and structures that the OS may use in booting. How either the firmware developer chooses to implement the required elements or the OS developer chooses to make use of those interfaces and structures is an implementation decision left for the developer.

The intent of this specification is to define a way for the OS and platform firmware to communicate only information necessary to support the OS boot process. This is accomplished through a formal and complete abstract specification of the software-visible interface presented to the OS by the platform and firmware.

Using this formal definition, a shrink-wrap OS intended to run on Intel® architecture-based platforms will be able to boot on a variety of system designs without further platform or OS customization. The definition will also allow for platform innovation to introduce new features and functionality that enhance platform capability without requiring new code to be written in the OS boot sequence.

Furthermore, an abstract specification opens a route to replace legacy devices and firmware code over time. New device types and associated code can provide equivalent functionality through the same defined abstract interface, again without impact on the OS boot support code.

The EFI specification is primarily intended for the next generation of 32-bit Intel architecture (IA-32) and Itanium®-based computers. Thus, the specification is applicable to a full range of hardware platforms from mobile systems to servers. The specification provides a core set of services along with a selection of protocol interfaces. The selection of protocol interfaces can evolve over time to be optimized for various platform market segments. At the same time the specification allows maximum extensibility and customization abilities for OEMs to allow differentiation. In this, the purpose of EFI is to define an evolutionary path from the traditional "PC-AT"-style boot world into a legacy-API free environment.

1.1 EFI Driver Model Extensions

Access to boot devices is provided through a set of protocol interfaces. The *EFI 1.02 Specification* describes these protocol interfaces in detail. However, it does not describe how these protocol interfaces are produced by the system firmware. The *EFI 1.10 Specification* includes extensions to the *EFI 1.02 Specification* that describe the *EFI Driver Model* along with additional protocol interfaces that provide access to a richer set of boot devices. One purpose of the *EFI Driver Model* is to provide a replacement for "PC-AT"-style option ROMs. It is important to point out that drivers written to the *EFI Driver Model* are designed to access boot devices in the preboot environment. They are not designed to replace the high performance OS specific drivers. The *EFI 1.10 Specification* is designed to be backward compatible with the *EFI 1.02 Specification*. This means that any EFI applications or drivers written to the *EFI 1.02 Specification* will continue to function on system firmware that complies with the *EFI 1.10 Specification*.

The *EFI Driver Model* is designed to support the execution of modular pieces of code, also known as drivers that run in the preboot environment. These drivers may manage or control hardware buses and devices on the platform or they may provide some software derived platform specific service.

The *EFI Driver Model* is designed to extend the *EFI Specification* in a way that supports device drivers and bus drivers. These extensions are provided in the form of new protocols, new boot services, and updated EFI boot services that are backward compatible with their original versions. The *EFI Driver Model* also contains information required by EFI driver writers to design and implement any combination of bus drivers and device drivers that a platform may need to boot an EFI compliant OS.

The *EFI Driver Model* is designed to be generic and can be adapted to any type of bus or device. The *EFI 1.10 Specification* describes how to write PCI bus drivers, PCI device drivers, USB bus drivers, USB device drivers, and SCSI drivers. Additions details are provided that allow EFI drivers to be stored in PCI option ROMs while maintaining compatibility with legacy option ROM images.

One of the design goals in the *EFI 1.10 Specification* is to keep the driver images as small as possible. However, if a driver is required to support multiple processor architectures, a driver object file would have to be shipped for each supported processor architecture. To address this space issue, the *EFI 1.10 Specification* also defines the *EFI Byte Code Virtual Machine*. An EFI driver can be compiled into a single EFI Byte Code Virtual Machine object file. EFI 1.10 complaint firmware must contain an EFI Byte Code interpreter. This allows a single EFI Byte Code object file to be shipped that supports multiple processor architectures. Another space saving technique is the use of compression. The *EFI 1.10 Specification* defines compression and decompression algorithms that may be used to reduce the size of EFI drivers, and thus reduce the overhead when EFI drivers are stored in ROM devices.

The information contained in the *EFI 1.10 Specification* can be used by OSVs, IHVs, OEMs, and firmware vendors to design and implement EFI firmware, EFI drivers that produce standard protocol interfaces, and EFI operating system loaders that can be used to boot EFI compliant operating systems.

1.2 Overview

This EFI 1.10 Specification is organized as listed in Table 1-1.

Table 1-1. Organization of the EFI Specification

Chapter/Appendix	Description
1. Introduction	Introduces the EFI Specification and topics related to using the specification.
2. Overview	Describes the major components of EFI, including the boot manager, firmware core, calling conventions, protocols, and requirements.
3. Boot Manager	Describes the boot manager, which is used to load EFI drivers and EFI applications.
4. EFI System Table	Describes the EFI System Table that is passed to every EFI driver and EFI application.
5. Services — Boot Services	Contains the definitions of the fundamental services that are present in an EFI-compliant system before an OS is booted.
6. Services — Runtime Services	Contains definitions for the fundamental services that are present in an EFI-compliant system before and after an OS is booted.
7. Protocols — EFI Loaded Image	Defines the EFI Loaded Image Protocol that describes an EFI image that has been loaded into memory.
8. Protocols — Device Path Protocol	Defines the device path protocol and provides the information needed to construct and manage device paths in the EFI environment.
9. Protocols — EFI Driver Model	Describes a generic driver model for EFI. This includes the set of services and protocols that apply to every bus and device type. These protocols include the Driver Binding Protocol, the Platform Driver Override Protocol, the Bus Specific Driver Override Protocol, the Driver Diagnostics Protocol, the Driver Configuration Protocol, and the Component Name Protocol.
10. Protocols — Console Support	Defines the Console I/O protocols, which handle input and output of text-based information intended for the system user while executing in the EFI boot services environment. These protocols include the Simple Input Protocol, the Simple Text Output Protocol, the Universal Graphics Adapter (UGA) Protocol, the Simple Pointer Protocol, and the Serial I/O Protocol.

continued

Table 1-1. Organization of the EFI Specification (continued)

Chapter/Appendix	Description
11. Protocols — Bootable Image Support	Defines the protocols that provide access to bootable images while executing in the EFI boot services environment. It also describes the supported disk layouts including MBR, El Torito, and the Guided Partition Table (GPT). These protocols include the Load File Protocol, the Simple File System Protocol, the Disk I/O Protocol, the Block I/O Protocol, and the Unicode Collation Protocol.
12. Protocols — PCI Bus Support	Defines PCI Bus Drivers, PCI Device Drivers, and PCI Option ROM layouts. The protocols described include the PCI Root Bridge I/O Protocol and the PCI I/O Protocol.
13. Protocols — SCSI Bus Support	Defines the SCSI Pass Thru Protocol that is used to abstract access to a SCSI channel that is produced by a SCSI host controller.
14. Protocols — USB Support	Defines USB Bus Drivers and USB Device Drivers. The protocols described include the USB Host Controller Protocol and the USB I/O Protocol.
15. Protocols — Network Support	Defines the protocols that provide access to network devices while executing in the EFI boot services environment. These protocols include the Simple Network Protocol, the PXE Base Code Protocol, and the Boot Integrity services (BIS) Protocol.
16. Protocols — Debugger Support	An optional set of protocols that provide the services required to implement a source level debugger for the EFI environment. The EFI DebugPort Protocol provides services to communicate with a remote debug host. The Debug Support Protocol provides services to hook processor exceptions, save the processor context, and restore the processor context. These protocols can be used in the implementation of a debug agent on the target system that interacts with the remote debug host.
17. Protocols — Compression Algorithm Specification	Describes in detail the EFI compression/decompression algorithm, as well as the EFI Decompress Protocol. The EFI Decompress Protocol provides a standard decompression interface for use at boot time. The EFI Decompress Protocol is used by a PCI Bus Driver to decompress EFI drivers stored in PCI Option ROMs.
18. Protocols — Device I/O Protocol	Defines the Device I/O protocol, which is used by code running in the EFI boot services environment to access memory and I/O.

continued

Table 1-1. Organization of the EFI Specification (continued)

Chapter/Appendix	Description
19. EFI Byte Code Virtual Machine	Defines the EFI Byte Code virtual processor and its instruction set. It also defines how EBC object files are loaded into memory, and the mechanism for transitioning from native code to EBC code and back to native code. The information in this document is sufficient to implement an EFI Byte Code interpreter, an EFI Byte Code compiler, and an EFI Byte Code linker.
A. GUID and Time Formats	Explains format of EFI GUIDs (Guaranteed Unique Identifiers).
B. Console	Describes the requirements for a basic text-based console required by EFI-conformant systems to provide communication capabilities.
C. Device Path Examples	Examples of use of the data structures that defines various hardware devices to the EFI boot services.
D. Status Codes	Lists success, error, and warning codes returned by EFI interfaces.
E. 32/64-Bit UNDI Specification	This appendix defines the 32/64-bit H/W and S/W Universal Network Driver Interfaces (UNDIs).
F. Using the Simple Pointer Protocol	This appendix provides the suggested usage of the Simple Pointer Protocol.
G. Using the EFI SCSI Pass Thru Protocol	This appendix provides an example on how the SCSI Pass Thru Protocol can be used.
H. Compression Source Code	The C source code to an implementation of the EFI Compression Algorithm.
I. Decompression Source Code	The C source code to an implementation of the EFI Decompression Algorithm.
J. EFI Byte Code Virtual Machine Opcode Summary	A summary of the opcodes in the instruction set of the EFI Byte Code Virtual Machine.
K. Alphabetic Function List	Lists all EFI interface functions alphabetically.
References	Lists all necessary and/or useful specifications, web sites, and other documentation that is referenced in this EFI Specification.
Glossary	Briefly describes terms defined or referenced by this specification.
Index	Provides an index to the key terms and concepts in the specification.

1.3 Goals

The "PC-AT" boot environment presents significant challenges to innovation within the industry. Each new platform capability or hardware innovation requires firmware developers to craft increasingly complex solutions, and often requires OS developers to make changes to their boot code before customers can benefit from the innovation. This can be a time-consuming process requiring a significant investment of resources.

The primary goal of the EFI specification is to define an alternative boot environment that can alleviate some of these considerations. In this goal, the specification is similar to other existing boot specifications. The main properties of this specification and similar solutions can be summarized by these attributes:

- *Coherent, scalable platform environment.* The specification defines a complete solution for the firmware to completely describe platform features and surface platform capabilities to the OS during the boot process. The definitions are rich enough to cover the full range of contemporary Intel architecture-based system designs.
- *Abstraction of the OS from the firmware.* The specification defines interfaces to platform capabilities. Through the use of abstract interfaces, the specification allows the OS loader to be constructed with far less knowledge of the platform and firmware that underlie those interfaces. The interfaces represent a well-defined and stable boundary between the underlying platform and firmware implementation and the OS loader. Such a boundary allows the underlying firmware and the OS loader to change provided both limit their interactions to the defined interfaces.
- *Reasonable device abstraction free of legacy interfaces.* "PC-AT" BIOS interfaces require the OS loader to have specific knowledge of the workings of certain hardware devices. This specification provides OS loader developers with something different—abstract interfaces that make it possible to build code that works on a range of underlying hardware devices without having explicit knowledge of the specifics for each device in the range.
- *Abstraction of Option ROMs from the firmware.* This specification defines interfaces to platform capabilities including standard bus types such as PCI, USB, and SCSI. The list of supported bus types may grow over time, so a mechanism to extend to future bus types is included. These defined interfaces and the ability to extend to future bus types are components of the *EFI Driver Model*. One purpose of the *EFI Driver Model* is to solve a wide range of issues that are present in existing "PC-AT" option ROMs. Like OS loaders, drivers use the abstract interfaces so device drivers and bus drivers can be constructed with far less knowledge of the platform and firmware that underlie those interfaces.
- *Architecturally shareable system partition.* Initiatives to expand platform capabilities and add new devices often require software support. In many cases, when these platform innovations are activated before the OS takes control of the platform, they must be supported by code that is specific to the platform rather than to the customer's choice of OS. The traditional approach to this problem has been to embed code in the platform during manufacturing (for example, in flash memory devices). Demand for such persistent storage is increasing at a rapid rate. This specification defines persistent store on large mass storage media types for use by platform support code extensions to supplement the traditional approach. The definition of how this works is made clear in the

specification to ensure that firmware developers, OEMs, operating system vendors, and perhaps even third parties can share the space safely while adding to platform capability.

Defining a boot environment that delivers these attributes could be accomplished in many ways. Indeed several alternatives, perhaps viable from an academic point of view, already existed at the time this specification was written. These alternatives, however, typically presented high barriers to entry given the current infrastructure capabilities surrounding Intel architecture platforms. This specification is intended to deliver the attributes listed above while also recognizing the unique needs of an industry that has considerable investment in compatibility and a large installed base of systems that cannot be abandoned summarily. These needs drive the requirements for the additional attributes embodied in this specification:

- *Evolutionary, not revolutionary.* The interfaces and structures in the specification are designed to reduce the burden of an initial implementation as much as possible. While care has been taken to ensure that appropriate abstractions are maintained in the interfaces themselves, the design also ensures that reuse of BIOS code to implement the interfaces is possible with a minimum of additional coding effort. In other words, on IA-32 platforms the specification can be implemented initially as a thin interface layer over an underlying implementation based on existing code. At the same time, introduction of the abstract interfaces provides for migration away from legacy code in the future. Once the abstraction is established as the means for the firmware and OS loader to interact during boot, developers are free to replace legacy code underneath the abstract interfaces at leisure. A similar migration for hardware legacy is also possible. Since the abstractions hide the specifics of devices, it is possible to remove underlying hardware, and replace it with new hardware that provides improved functionality, reduced cost, or both. Clearly this requires that new platform firmware be written to support the device and present it to the OS loader via the abstract interfaces. However, without the interface abstraction, removal of the legacy device might not be possible at all.
- *Compatibility by design.* The design of the system partition structures also preserves all the structures that are currently used in the "PC-AT" boot environment. Thus it is a simple matter to construct a single system that is capable of booting a legacy OS or an EFI-aware OS from the same disk.
- *Simplifies addition of OS-neutral platform value-add.* The specification defines an open extensible interface that lends itself to the creation of platform "drivers." These may be analogous to OS drivers, providing support for new device types during the boot process, or they may be used to implement enhanced platform capabilities like fault tolerance or security. Furthermore this ability to extend platform capability is designed into the specification from the outset. This is intended to help developers avoid many of the frustrations inherent in trying to squeeze new code into the traditional BIOS environment. As a result of the inclusion of interfaces to add new protocols, OEMs or firmware developers have an infrastructure to add capability to the platform in a modular way. Such drivers may potentially be implemented using high level coding languages because of the calling conventions and environment defined in the specification. This in turn may help to reduce the difficulty and cost of innovation. The option of a system partition provides an alternative to nonvolatile memory storage for such extensions.
- *Built on existing investment.* Where possible, the specification avoids redefining interfaces and structures in areas where existing industry specifications provide adequate coverage. For example, the ACPI specification provides the OS with all the information

UNITED STATES
PATENT AND TRADEMARK OFFICE

P.O. Box 1450
Alexandria, VA 22313 - 1450
www.uspto.gov

ELECTRONIC ACKNOWLEDGEMENT RECEIPT

APPLICATION #	RECEIPT DATE / TIME	ATTORNEY DOCKET #
90/019,244	**09/01/2023 07:01:32 PM ET**	**185945.012400**

Title of Invention

SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR

Application Information

APPLICATION TYPE	Utility / ex parte reexam	PATENT #	7627781
CONFIRMATION #	7031	FILED BY	Judi Cordo
PATENT CENTER #	62727008	FILING DATE	-
CUSTOMER #	56436	FIRST NAMED INVENTOR	Scott L. Michaelis
CORRESPONDENCE ADDRESS	-	AUTHORIZED BY	Andrew Sommer

Documents

TOTAL DOCUMENTS: 24

DOCUMENT	PAGES	DESCRIPTION	SIZE (KB)
185945_012400_Transmittal_Form.PDF	4	Reexam Miscellaneous Incoming Letter	285 KB
185945_012400_Reexam_Request_US7627781.PDF	116	Reexam Miscellaneous Incoming Letter	6816 KB
185945_012400_Ex_1001_US7627781.PDF	18	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	7602 KB
185945_012400_Ex_1003_Wolfe.PDF	70	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	3782 KB

185945_012400_Ex_1004_Safford.PDF	9	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	2571 KB
185945_012400_Ex_1005_Kennedy.PDF	15	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	7910 KB
185945_012400_Ex_1006_Fox.PDF	12	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	3942 KB
185945_012400_Ex_1007_Kalyandasundharam.PDF	18	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	6352 KB
185945_012400_Ex_1014_Intel_Itanium_Manual.PDF	7	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	425 KB
185945_012400_Ex_1011_Hobson.PDF	10	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	2843 KB
185945_012400_Ex_1015_Intel_MultiProcessor_Specification.PDF	7	Specification	4227 KB
185945_012400_Ex_1013_Cepulis.PDF	24	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	8074 KB
185945_012400_Ex_1008_Nguyen.PDF	17	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	5108 KB
185945_012400_Ex_1012_Levi.PDF	12	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	10453 KB

185945_012400_Ex_1010_Intel_EFI_Specification.PDF	9	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	13280 KB
185945_012400_Ex_1009_Bartlett.PDF	11	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	18639 KB
185945_012400_Exhibit_A_Wolfe_CV.PDF	9	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	186 KB
185945_012400_IDS.PDF	5	Information Disclosure Statement (IDS) Form (SB08)	598 KB
185945_012400_Exhibits_AA_DD_Claim_Charts.PDF	79	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	3105 KB
185945_012400_Certificate_of_Service_US7627781.PDF	3	Reexam Certificate of Service	180 KB
185945_012400_Ex_1002_File_History_2_of_3.pdf	100	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	16726 KB
185945_012400_Ex_1002_File_History_3_of_3.pdf	110	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	21222 KB
185945_012400_Ex_1002_File_History_1_of_3.pdf	100	Reexam - Affidavit/Declaration/Exhibit Filed by 3rd Party	16698 KB
185945_012400_Requesters_POA.PDF	1	Power of Attorney	105 KB

Digest

DOCUMENT	MESSAGE DIGEST(SHA-512)
185945_012400_Transmittal_Form.PDF	B4C712F7A5945E02B35F35DC35050D08A52B79D8A320026E01665CEC20B2E650180651E2E98C183AF44E874BA9050F8931292DB6BD69E9B8F3217D15A11DFE87
185945_012400_Reexam_Request_US7627781.PDF	957DE84C6AA9CB6F8246B3869C5C3DFDEFA2020849D8A88B389B8102924BFBDF728D251315A544BA96D30E9336822297E156A5B41064BEF97CC77C857927DAC3
185945_012400_Ex_1001_US7627781.PDF	EDCC818C7D9F26C1F873C5CD707F693205683B92972F6FD528E784509389DE4CC11592C729C0C2DA3E3964759C31C0BA9E7D15C9E295DEFC76C98C80B181AD55
185945_012400_Ex_1003_Wolfe.PDF	C3CA89850F1BD6D25EE1AA6F9C04B2A4BD34510F29AEA2B4A5489E81FA7760E9D4683A7CFC8B2959EC95794F830573EEE0BFD626CAFA364244DA5D7ADDB6C061
185945_012400_Ex_1004_Safford.PDF	B7B65205CF40D0D2023B622021AEA02EEED88868665116BA1433EF52ED07127C8ABE07AE01AC3A597C7411BC6EE626F2D1651B8116FDD89214287B78F3DA0EEB
185945_012400_Ex_1005_Kennedy.PDF	C933CED65DAD22D773A09BC6214E132F33FAA02972774EC15EB222DE3C5A78A95BA1A7FBF5A7F259C59C4045CDE10FB541D089105608942BAE5B491A87CB40B7
185945_012400_Ex_1006_Fox.PDF	FFD30EF6384EA055D9ABE4CD1E00B76AEBBB669BEB6EB58195430596AC271533DF26788B0626CD3A9C494418AF9D0C79F5CE67BA355FF87FBB5DE371A223086B
185945_012400_Ex_1007_Kalyandasundharam.PDF	4F9FAB7E2D71FFD8C15F982C8A896C0365846641C59C49E9C1F2BE9D12462AC716484128F3DC9498BE33EE9022CA0A55EA1A799EA24998D2710902F80AD62CB0
185945_012400_Ex_1014_Intel_Itanium_Manual.PDF	0E725FA257435F0DD74821F2CB8C3335047FBDE7343054B0AEEDF65ED42CB240677913A07E05D4535FAF25F7078BD5657C0648E5D6C68EE7D7F7EEA358E75C7E
185945_012400_Ex_1011_Hobson.PDF	BB8E732B7BED982216C4153A74C4F93239E0CEE1D77F9E9B28F15D5D10F3A941C2492815A3B446E31A28BCC0D76902BFA6

	C83CE2B75064468E01F545C5091E9A
185945_012400_Ex_1015_Intel_MultiProcessor_Specification.PDF	10D9AE33C988A248C8B4CAF76654F06C018C4BDB43C3878B7DA5D909C81CDA921ED0A845356324D67FF664FB70626DEC83E8F491F47E3807E5C1F94391407839
185945_012400_Ex_1013_Cepulis.PDF	5391C30632FFCFEF914190CA80B9FA862AF83AAA6C64DBB61750D1A1D0BD918A8DAECC9083C9A4157CFEC954F243235F7B01C51B755DCB353A7D8FFBCD3B67CC
185945_012400_Ex_1008_Nguyen.PDF	2EC5D0A655D0787941440919A92C13BB8290B44D839DEF19E5A2DA5C840AD3CE1BB452EC55969E51C96765ADFD798040CF67D32E7F51187F248E6AAB137D3573
185945_012400_Ex_1012_Levi.PDF	5ABBD36FA0CD9FDEC57AD179DCAED0D63D9DBB4FAF2CDA4B3E74CFE5EC1569D3E20D162A2F2857B991681F0EB4906D7F7E0F843E59073F52655C2E88014C82FC
185945_012400_Ex_1010_Intel_EFI_Specification.PDF	F9A6599EDFF9615D466FFDB7A0EC182A1ABFC798E851310C22A52E5725110AE74AF0A4C101639C0FD59909E2610E3C0845651CC8805EECF039287A4EE3FAC42E
185945_012400_Ex_1009_Bartlett.PDF	6C76A306EAFFB25135B0E04B7A15386E08F37F19D6E004C6FD4C498EE5CFBD0A1B54C099D0CABD61A6F89E9FFC92E9A56A5AEF3E265867FB8F3DBF58226752CA
185945_012400_Exhibit_A_Wolfe_CV.PDF	548374B45A680AF90170A20111430FBDFBABAFA5510D6F0AEB1E1A6E02EA51D6FCA95AA32C63EC6E475D87F3D0DEC2BE2700BCEA29C72AB0E6C0E39CC92975BE
185945_012400_IDS.PDF	7378BBDBD9D45D970467B0081884431CBDED8429F97603A2838BD5E8B0E379AA5B3CD7F60F6C5A0BA0C81AC761BADBCCDA9E057CF02A531D69E0AF13C6EFF3A5
185945_012400_Exhibits_AA_DD_Claim_Charts.PDF	8A3AFBDF4E1033047643F490C4A4B4CCAD85246A450A35D0B969800D22D6FB927E03FF59AE5658A3D7944FB0A416D30106742FB7D78907FC9113B9D49C7CE656
185945_012400_Certificate_of_Service_US7627781.PDF	58A5DDC56F731C45A511BFEFC96804D57EEC89B60EE89554D34FC1DB55895E3C6214A8AA035AD34BBEFFED141B5712E9

	3726A26375054307296C8BD97F529BB1
185945_012400_Ex_1002_File_History_2_of_3.pdf	2D411A33471A25A7A911BE89735B448459D0786C45F18478373B7CBDDE6D4D2781BE6291B464D03066903DA55F3019326F548195B28769CE7450373F41A6FC26
185945_012400_Ex_1002_File_History_3_of_3.pdf	5B92DDDB33E38E287F644B2B4217BF0E84C22958A248792C914B0405BD66DC8F487F258E38C4A56B883D29932CB0B34DA141472884EF0BCD6B38D54EE5FFFA8F
185945_012400_Ex_1002_File_History_1_of_3.pdf	EF25BA3224FD122DE6C014BE6D03886E8F1C40B182C82FF8A0B0D064507D5F26D3212A81842D65AC0F91859EB5E5C6C08497AD549F054CC3EDA3C98A25D1DF3C
185945_012400_Requesters_POA.PDF	0B6DF2B27D410DB42E53A4E363D6FD6599B876EE15D7274D220295B6790FC953110E5931732D644A2E93C0D78F72FF44FE6C6ADC8CAE8163B64D95DDC2520830

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Michaelis *et al.*	§	Attorney Docket No.: 185945.012400
	§	
U.S. Patent No. 7,627,781	§	Customer No.: 67395
	§	
Issue Date: December 1, 2009	§	
	§	
Filing Date: October 25, 2004	§	
	§	
For: SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR	§	

REQUEST FOR *EX PARTE* REEXAMINATION OF U.S. PATENT 7,627,781

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

Dear Commissioner:

Pursuant to the provisions of 35 U.S.C. §§ 301-307, Unified Patents, LLC ("Requester") hereby requests an *ex parte* reexamination of claims 1-30 (the "Challenged Claims") of U.S. Patent 7,627,781 (the "'781 Patent", Ex. 1001), which issued on December 1, 2009, based from U.S. Application 10/973,077 (the "'077 Application") filed on October 25, 2004. The '781 Patent was originally assigned to Hewlett-Packard Development Company, LP, as reflected in the assignment recorded in the U.S. Patent and Trademark Office ("USPTO") at reel/frame 015941/0381 and is currently assigned to Foras Technologies LTD., as reflected in the assignment recorded in the USPTO at reel and frame number 058992/0571.

This Request presents prior art references and analyses that are noncumulative of the prior

art that was before the Examiner during the original prosecution of the '781 Patent. As demonstrated herein, the Challenged Claims are invalid over these references. Requester therefore requests that an order for reexamination and an Office Action rejecting claims 1-30 be issued.

Ex Parte Patent Reexamination Filing Requirements

Pursuant to 37 C.F.R. § 1.510(b)(1), statements pointing out at least one substantial new question of patentability based on material, non-cumulative reference patents and printed publications for the Challenged Claims of the '781 Patent are provided in Section I of this Request.

Pursuant to 37 C.F.R. § 1.510(b)(2), reexamination of the Challenged Claims (1-30) of the '781 Patent is requested, and a detailed explanation of the pertinence and manner of applying the cited references to the Challenged Claims is provided in Section II of this Request.

Pursuant to 37 C.F.R. § 1.510(b)(3), copies of every patent or printed publication relied upon or referred to in the statement pointing out each substantial new question of patentability or in the detailed explanation of the pertinence and manner of applying the cited references are provided as Exhibits 1001-1015 (with Exhibits 1004-1008 and 1013 forming the basis for the substantial new questions of patentability presented herein) and Appendices A-D of this Request.

Pursuant to 37 C.F.R. §1.510(b)(4), a copy of the '781 Patent is provided as Exhibit 1001 of this Request, along with a copy of any disclaimer, certificate of correction, and reexamination certificate issued corresponding to the patent.

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

FORAS TECHNOLOGIES LTD.
c/o 56436 - Hewlett Packard Enterprise
3404 E. Harmony Road
Mail Stop 79
Fort Collins, CO 80528

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

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

TABLE OF CONTENTS

TABLE OF EXHIBITS ..... **iii**

I. SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY **1**

A. U.S. Patent 7,627,781 2

1. Technical Background 2

2. Summary 3

3. Discretionary Denial is not Warranted 5

4. Challenged Claims 5

5. Priority 9

6. Prosecution History 9

B. Level of Ordinary Skill in the Art 13

C. Claim Construction 13

1. Overview 13

2. **"boot processor"** 13

D. List of Prior Art Patents and Printed Publication 15

E. Ground 1: Kalyanasundharam in view of Kennedy Presents Substantial New Questions of Patentability for Claims 1-6 and 9-15 16

1. Overview of Kalyanasundharam 16

2. Overview of Kennedy 20

3. **Obviousness of the '781 Patent in view** Kalyanasundharam and Kennedy 22

(I) Overview of Motivation to Combine Kalyanasundharam and Kennedy as Proposed 23

(II) Reasonable Expectation of Success 23

(III) *KSR* Rationales 24

F. Ground 2: Safford in view of Kennedy Presents Substantial New Questions of Patentability for Claims 1-6 and 9-15 31

1. Overview of Safford 31

2. Overview of Kennedy 34

3. **Obviousness of the '781 Patent in view of Safford and Kennedy** 34

(I) Overview of Motivation to Combine Safford and Kennedy 34

(II) Reasonable Expectation of Success 35

(III) *KSR* Rationales 35

G. Ground 3: Kennedy in view of Nguyen and Fox Presents Substantial New Questions of Patentability for Claims 1-30 42

1. Overview of Kennedy 42

2. Overview of Nguyen 42
3. Overview of Fox 44
3. **Obviousness of the '781 Patent in view of Kennedy in view of Nguyen and Fox** 47
(I) Overview of Motivation to Combine Kennedy, Nguyen, and Fox as Proposed .. 47
(II) Reasonable Expectation of Success 49
(III) *KSR* Rationales 50
H. Ground 4: Safford Presents Substantial New Questions of Patentability for Claims 1 and 6 57
1. Overview of Safford 57
2. Overview of Cepulis 57
3. Anticipation by Safford 58
II. DETAILED APPLICATION OF THE PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED 58
A. Proposed Obviousness Rejections for the Challenged Claims 58
1. Ground 1 Kalyanasundharam in view of Kennedy 58
2. Ground 2 Safford in view of Kennedy 73
3. Ground 3 Kennedy in view of Nguyen and Fox 88
B. Proposed Anticipation Rejections for Claims 1 and 6 106
1. Ground 4 Safford in Light of Cepulis 106
C. Secondary Considerations 110
III. DISCLOSURE OF CONCURRENT LITIGATION, REEXAMINATION, AND RELATED PROCEEDINGS 110
IV. CONCLUSION 111

TABLE OF EXHIBITS

Exhibit	Description
Ex. 1001	U.S. Patent No. 7,627,781 to Michaelis et al. (the "'781 Patent")
Ex. 1002	U.S. File History of the '781 Patent ("U.S. File History")
Ex. 1003	Declaration of Andrew Wolfe ("Wolfe Decl.")
Ex. 1004	U.S. Patent No. 7,085,959 to Safford ("Safford")
Ex. 1005	U.S. Patent No. 5,450,576 to Kennedy et al. ("Kennedy")
Ex. 1006	U.S. Patent No. 7,251,746 to Fox ("Fox")
Ex. 1007	U.S. Patent No. 7,328,371 to Kalyanasundharam et al. ("Kalyanasundharam")
Ex. 1008	U.S. Patent Application Pub. No. 2004/0123201 to Nguyen et al. ("Nguyen")
Ex. 1009	Wendy Bartlett, et. al., *Commercial Fault Tolerance: A Tale of Two Systems, IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 1 (January-March 2004) ("Bartlett")
Ex. 1010	Intel, *Extensible Firmware Interface Specification*, Version 1.10 (Dec. 1, 2002) ("Intel EFI Specification")
Ex. 1011	U.S. Patent No. 7,191,328 to Hobson ("Hobson")
Ex. 1012	Bozidar Levi, *UNIX Administration, A Comprehensive Sourcebook for Effective Systems and Network Management* (2002) ("Levi")
Ex. 1013	U.S. Patent No. 5,491,788 to Cepulis et al. ("Cepulis")
Ex. 1014	*Intel IA-64 Architecture Software Developer's Manual*, Revision 1.1 (July 2000) ("Intel Itanium Manual")
Ex. 1015	*Intel Multiprocessor Specification*, Version 1.4 (May 1997) ("Intel Multiprocessor Specification")
Ex. AA	Appendix A: Claim chart comparing claims to Kalyanasundharam in view of Kennedy, as set forth in Ground 1.
Ex. BB	Appendix B: Claim chart comparing claims to Safford in view of Kennedy, as set forth in Ground 2.
Ex. CC	Appendix C: Claim chart comparing claims to Kennedy in view of Nguyen and Fox, as set forth in Ground 3.
Ex. DD	Appendix D: Claim chart comparing claims to Safford in light of Cepulis, as set forth in Ground 4.

I. SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

As shown below, Requester submits that the prior art references raise substantial new questions of patentability because "the teaching of the (prior art) patents and printed publications is such that a reasonable examiner would consider the teaching to be important in deciding whether or not the claim is patentable." *See* Manual of Patent Examining Procedure (M.P.E.P.) § 2242.

For example, the references discussed below and in further detail in the attached charts, when considered alone or in combination, teach or suggest each limitation of the Challenged Claims, including a plurality of processors operating in lockstep, a data storage that stores information that assigns the roles of boot processor and spare processor, logic that transfers the role of boot processor to the spare processor in response to a loss of lockstep, and an operating system that recognizes the spare processor as the new boot processor without shutting down. Two of the references relied upon in this Request, namely U.S. Patent No. 7,085,959 to Safford ("Safford") and U.S. Patent 7,251,746 to Fox et al. ("Fox"), were considered alone or in different combinations by the Examiner who handled the original examination of the '077 Application. As shown below, the ways in which these two references are applied in this Request differ from, and are non-cumulative to, all those decided by the Examiner in the original examination of the '077 Application. Accordingly, these two references, as applied in combination with U.S. Patent No. 5,450,576 to Kennedy ("Kennedy") in this Request, each raise substantial new questions of patentability, none of which has already been decided by the Office. In addition, this Request presents new, non-cumulative technological teachings of Cepulis that place Safford in "a new light or a different way that escaped review during earlier examination." *See* MPEP § 2242. In other words, the "same question of patentability as to the claim has not been decided by the Office in an earlier concluded examination or review of the patent" at least because Safford and Fox were never considered in view of Kennedy, in light of Cepulis, or the like during prosecution of the '077 Application or during a prior post-grant proceeding challenging any claim of the '781 Patent.

All the other references relied upon in this Request—namely Kennedy, U.S. Patent No. 7,328,371 to Kalyanasundharam et al. ("Kalyanasundharam"), U.S. Patent Publication No. 2004/0123201 to Nguyen et al. ("Nguyen"), and U.S. Patent No. 5,491,788 to Cepulis et al. ("Cepulis")—are new and were not previously considered at all by the Office. Thus, the "same question of patentability as to the claim[s] has not been decided by the Office in an earlier concluded examination or review of the patent" at least because none of these other references

were before the Office during prosecution of the '077 Application or during a prior post-grant proceeding challenging any claim of the '781 Patent. The original Examiner did not consider any of these other references prior to deciding to allow the '077 Application and issue the '781 Patent.

In view of these observations and the corresponding support that follows, reexamination should be ordered and the Challenged Claims should be canceled.

A. U.S. Patent 7,627,781

1. Technical Background

Since the 1960s and 1970s, computer system designers, such as IBM and Tandem, have been developing and releasing fault-tolerant systems with high availability. *Bartlett* (Ex. 1009) at 89. For example, in 1964, IBM shipped the S/360, which "was designed with robust error checking and was the first commercial computer to include Error Correction Code (ECC) in memory." *Id.* As another example, the Tandem NonStop system shipped in 1976 and included "processor pairs [that] are an effective design for recovering from transient problems." *Id.* at 91. Tandem and IBM both continually improved these systems, and by the early 1990s Tandem had moved to "tight lockstepping of a pair of MIPS chips" with a variant "being developed for future Intel Itanium(R)-based platforms." *Id.* at 91-92. For its part, in 2001 IBM introduced its "Autonomic Computing initiative," which focused on self-healing that included "dynamic CPU sparing, a process by which a failed CPU can be transparently replaced by a spare." *Id.* at 95. These systems were often operating-system centric, with "the operating system provid[ing] its own layer of fault detection and fail-fast response [and supplying] the infrastructure that allows applications to survive single processor failures." *Id.* at 89.

Around the same time that IBM and Tandem were iterating on their high-availability systems in which the OSes managed the hardware, Intel was working on an initiative that attempted to address some of challenges with OS management of hardware and released the Extensible Firmware Interface (EFI). *See Intel EFI Specification* (Ex. 1010) at 1-6. One of the primary goals of the EFI was "abstraction of the OS from the firmware" to allow the OS "to be constructed with far less knowledge of the platform and firmware that underlie those interfaces." *Id.* In other words, Intel proposed "a backward-compatible solution in which the EFI layer is configured to emulate certain aspects of legacy BIOS operation." *Hobson* (Ex. 1011) at 1:43-46.

Within this historical context, on October 25, 2004, Applicants for the '781 Patent filed a patent application claiming that they had invented a novel system that is "responsive to detecting

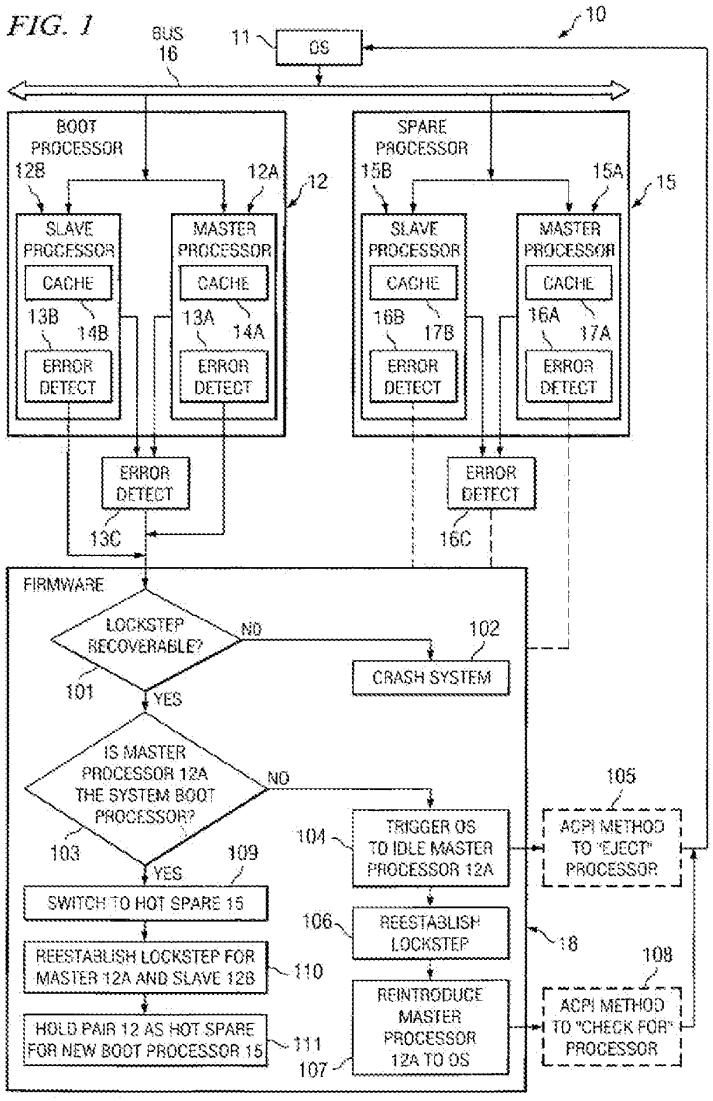
loss of lockstep" by transferring "the role of boot processor to [a] spare processor." *'781 Patent* (Ex. 1001), claim 1. However, as noted above and shown in greater detail below, detecting and recovering from loss-of-lockstep was not new, and abstracting and replacing a failed processor with a spare was not new either. Furthermore, as explained in detail herein, it would have been obvious to combine these limitations in the manner claimed in the '781 Patent.

2. Summary

The '781 Patent is generally directed to systems and methods for recovering from lockstep errors that occur in boot processors. Ex. 1001, Abstract. For example, the '781 Patent describes a multi-processor system that assigns the roles of boot processor and spare processor and then, in response to detecting a loss of lockstep in the boot processor, transfers the role of boot processor to the spare processor. *See id.* Specifically, the '781 Patent notes that a firmware component known as a processor abstraction layer (PAL) "copies all the needed information from memory into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor." *Id.* at 17:57-61.

According to the '781 Patent, a loss of lockstep "is used broadly…to refer to any error in the pair of lockstep processors." *Id.* at 2:38-39, 6:13-19. In the '781 Patent, the boot processor is responsible for booting the underlying system and communicating the output of processed instructions to the system bus. *See id.* at 11:60-12:26, 6:26-40. The '781 Patent distinguishes the boot processor from other processors (*e.g.,* the spare processor) by noting that the output of the boot processor is the only one communicated to the system bus. *See id.* at 6:26-40.

The configuration of the '781 Patent's system is illustrated in FIG. 1, which is reproduced below for convenience.



'781 Patent (Ex. 1001), Fig. 1

The '781 Patent includes a total of thirty claims, four of which are independent claims (*e.g.*, claims 1, 9, 16, and 25) whose limitations vary slightly relative to one another. For example, independent claim 1 of the '781 Patent specifies that the spare processor is a hot spare held in reserve until the loss of lockstep is detected. *See id.*, claim 1, 5:4-10. Independent claim 9 of the '781 Patent specifies that, after the transfer of the boot processor role to the spare processor, an operating system recognizes the spare processor as the new boot processor without shutting down. *See id.*,

claim 9, 18:33-36. Independent claims 16 and 25 of the '781 Patent specify that the boot processor and the spare processor are both communicatively coupled to the same communication bus and also specify that the boot processor and the spare processor both include a master-and-slave pair whose outputs are compared to one another for detecting a loss of lockstep. *See id.*, claims 16 and 25.

3. Discretionary Denial is not Warranted

Requester submits that, because a substantial new question is presented herein that has not been previously adjudicated by the U.S. Patent Office, 35 U.S.C. § 325(d) does not warrant denial of this Reexam Request. Indeed, 35 U.S.C. § 325(d) notes that:

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

Requester notes that the cited art combinations presented herein were not considered by the Examiner during initial prosecution. As such, the material differences between the prior art asserted in this request and the prior art involved during examination are substantial. Moreover, because none of the combinations presented herein were relied on by the Examiner as a primary basis for unpatentability, Requester submits that 35 U.S.C. § 325(d) does not warrant denial of this Reexam Request.

4. Challenged Claims

For convenience, the claims for which reexamination is requested are reproduced below. Claims 1, 9, 16, and 25 are the independent claims, while the other Challenged Claims depend directly or indirectly from the independent claims.

1. A system comprising:

a plurality of processors;

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

logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor, wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor.

2. The system of claim 1 wherein said data storage is non-volatile data storage.

3. The system of claim 1 further comprising:

a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor.

4. The system of claim 1 further comprising:

a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor.

5. The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor.

6. The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor.

7. The system of claim 1 further comprising:

an operating system, wherein, responsive to the logic transferring the role of boot processor to the spare processor, the operating system recognizes the spare processor as the boot processor.

8. The system of claim 1 wherein said plurality of processors are Itanium Processor Family (IPF) processors.

9. A method comprising:

assigning a role of boot processor to one of a plurality of processors in a system;

assigning a role of spare processor to another of said plurality of processors;

responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor; and

responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system.

10. The method of claim 9 wherein said assigning said role of boot processor comprises storing information to non-volatile data storage that identifies said one of said plurality of

processors as said boot processor.

11. The method of claim 9 wherein said assigning said role of spare processor comprises storing information to non-volatile data storage that identifies said another of said plurality of processors as said spare processor.

12. The method of claim 9 further comprising:

determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor; and

selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor.

13. The method of claim 9 further comprising:

during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare.

14. The method of claim 9 further comprising:

during system boot-up, system firmware booting the spare processor as an active spare.

15. The method of claim 9 wherein system firmware performs said changing operation.

16. A system comprising:

a plurality of lockstep processor modules that each includes a master processor and a slave processor that is in lockstep with said master processor;

a first communication bus with which at least a first two of said plurality of lockstep processor modules are communicatively coupled;

a second communication bus with which at least a second two of said plurality of lockstep processor modules are communicatively coupled; and

data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors, wherein said spare processor is communicatively coupled to the same one of said first and second communication buses as said boot processor.

17. The system of claim 16 further comprising:

system firmware operable, responsive to detecting loss of lockstep for said boot processor, changing the assignment of the role of boot processor to said spare processor.

18. The system of claim 17 wherein said system firmware performs said changing during system runtime without requiring an operating system of said system to be shutdown.

19. The system of claim 16 further comprising:

a user interface for receiving information selecting said one of said plurality of processors to be assigned the role of boot processor.

20. The system of claim 16 further comprising:

a user interface for receiving information selecting said another of said plurality of processors to be assigned the role of spare processor.

21. The system of claim 20 wherein said user interface presents for selection as said spare processor at least one processor that is communicatively to the same one of said first and second communication buses as said boot processor.

22. The system of claim 16 further comprising:

system firmware that, during system boot-up, identifies the spare processor and holds said spare processor and holds said spare processor in reserve as a hot spare.

23. The system of claim 16 further comprising:

system firmware that, during system boot-up, boots the spare processor as an active spare.

24. The system of claim 16 wherein said plurality of lockstep processor modules include Itanium Processor Family (IPF) processors.

25. A method comprising:

selecting a first one of a plurality of lockstep processor modules in a system as a boot processor, wherein each of said plurality of lockstep processor modules includes a master processor and a slave processor that is in lockstep with said master processor, and wherein said boot processor is communicatively coupled to a first of a plurality of communication buses;

selecting a second one of said plurality of lockstep processor modules as a spare processor, wherein said spare processor is selected as one that is communicatively coupled to the same communication bus as said boot processor; and

responsive to detecting loss of lockstep for said boot processor, said system causing said spare processor to dynamically assume the role of boot processor.

26. The method of claim 25 wherein said system causing said spare processor to dynamically assume the role of boot processor comprises:

said system causing said spare processor to dynamically assume the role of boot processor without shutting down an operating system of said system.

27. The method of claim 25 further comprising:

receiving input to said system specifying said first one of said plurality of lockstep processor modules to be said boot processor.

28. The method of claim 25 further comprising:

receiving input to said system specifying said second one of said plurality of lockstep processor modules to be said spare processor.

29. The method of claim 25 further comprising:

storing information identifying said selected first one of said plurality of lockstep processor modules as said boot processor and identifying said selected second one of said plurality of lockstep processor modules as said spare processor.

30. The method of claim 29 wherein said system causing said spare processor to dynamically assume the role of boot processor comprises:

system firmware determining said spare processor from said stored information; and

said system firmware causing an operating system to recognize said spare processor as said boot processor.

5. Priority

The '781 Patent has a priority date of October 25, 2004. Specifically, the '781 Patent resulted from the allowance of the '077 Application, which was filed on October 25, 2004. While the '781 Patent does include cross-references to applications related by subject matter, the '781 Patent has no family members related by priority.

6. Prosecution History

Prosecution of the '077 Application consisted of two Office Actions (*e.g.*, a Non-Final Office Action of Sept. 20, 2007, and a Non-Final Office Action of Mar. 19, 2008) and corresponding Responses, which were followed by an Examiner Interview and a corresponding Supplemental Amendment. *See File History* (Ex. 1002) at 189-190, 207-214, 233-247, 274-295. As will be described below, the Examiner who handled this prosecution made several errors that led to the allowance of the '077 Application. These errors included (1) the Examiner being persuaded by a faulty, subsequently retracted argument about Safford not being a valid prior-art reference under 35 U.S.C. § 103(a), (2) the Examiner believing that a hot spare processor held in reserve was missing from the cited art, and (3) the Examiner believing that an operating system recognizing a new role for the spare processor without requiring a shutdown was missing from the

cited art.

In the Non-Final Office Action of Sept. 20, 2007 (the "first OA"), the Examiner rejected claims 1, 3-4, 6-8, 10, and 13-17 under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent 7,085,959 to Safford ("Safford"). *See File History* (Ex. 1002) at 277-281. In the first OA, the Examiner also rejected claims 2, 5, 9, 11-12, and 18-32 under 35 U.S.C. § 103(a) as being rendered obvious by Safford in view of U.S. Patent 5,915,082 to Marshall ("Marshall"). *See id.* at 282-294.

In response to the first OA, the Applicant traversed the rejections raised against claims 1-32 without making any claim amendments. *See id.* at 260-271. For example, with respect to claims 1-17, the Applicant argued that Safford did not disclose a boot processor, much less transferring the role of boot processor to the spare processor. *See id.* at 268-269. With respect to claims 18-32, the Applicant argued that Safford was not a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a). *See id.* at 270. Specifically, the Applicant argued that "the disclosures of Safford are available only as 35 U.S.C. § 102(e)-type prior art. 35 U.S.C. § 103(c) specifies that Safford 'shall not preclude patentability' of the claimed invention." *Id.*

Based on the Non-Final Office Action of Mar. 19, 2008 (the "second OA"), the Applicant's arguments directed to the rejections raised under 35 U.S.C. § 102(e) were unsuccessful in persuading the Examiner that original claims 1-17 were allowable. *See id.* at 235-239. However, based on the second OA, the Applicant's argument directed to the rejections raised under 35 U.S.C. § 103(a) appeared to be successful in persuading the Examiner that original claims 18-32 were allowable. *See id.* at 246-247. Accordingly, the Examiner indicated that claims 18-32 were allowed over the cited art. *See id.* In the second OA, the Examiner maintained the anticipation rejections raised against claims 1, 3-4, 6-8, 10, and 13-17 under 35 U.S.C. § 102(e) and also raised new obviousness rejections against claims 1-17 under 35 U.S.C. § 103(a) in view of the combination of Fox and U.S. Patent 6,625,749 to Quach ("Quach"). *See id.* at 235, 240-246.

In response to the second OA, the Applicant essentially repeated the same arguments about Safford not disclosing a boot processor or transferring the role of boot processor to the spare processor. *See id.* at 227-228. In addition, the Applicant made similar arguments about Fox and Quach, asserting that neither Fox nor Quach teaches or suggests anything about a boot processor or transferring the role of boot processor to the spare processor. *See id.* at 229. Notably, even though the Examiner was apparently persuaded by the Applicant's previous argument about

Safford not being a valid prior-art reference under 35 U.S.C. § 103(a), the Applicant retracted that argument and admitted that Safford was a valid prior-art reference under 35 U.S.C. § 103(a). *See id.* This admission made by the Applicant is reproduced below for convenience.

> NOTE: In applicant's reply to the previous office action, applicant asserted that Safford was not proper prior art under 35 U.S.C. § 103. However, that assertion was not correct, and is hereby retracted, with apologies for the mistake. Safford was published on January 8, 2004, before the filing of the present application on October 25, 2004.

File History (Ex. 1002), p. 229

After the Applicant responded to the second OA, the Examiner and the Applicant held an Examiner Interview on Sept. 5, 2008, to discuss a proposal to place some of claims 1-17 in condition for allowance. *See id.* at 189-190, 207-214. The Applicant then filed a Supplemental Amendment that incorporated the subject matter of claim 6 (which was simultaneously cancelled) into independent claim 1 and the subject matter of claim 14 (which was simultaneously cancelled) into independent claim 10 [issued claim 9], thus **resulting in the allowance of the '077** Application and the issuance of the '781 Patent. *See id.* at 207-214. The subject matter incorporated into independent claim 1 clarified that the spare processor is a hot spare held in reserve. *See id.* The subject matter incorporated into independent claim 10 clarified that, in response to changing the assignment of the boot processor role to the spare processor, an operating system recognizes the spare processor as the boot processor without requiring a shutdown. *See id.*

As the Supplemental Amendment resulted in the allowance of the '077 Application and the issuance of the '781 Patent, it appears as though the Examiner believed that the subject matter incorporated into independent claims 1 and 10 was missing from the cited art. *See id.* at 189-190. Additionally or Alternatively, the Examiner may have been partially persuaded by the **Applicant's repeated arguments directed to** Safford not disclosing a boot processor or transferring the role of boot processor to the spare processor.

In view of the above observations, the allowance of the '077 Application and the issuance of the '781 Patent apparently resulted from (1) the Examiner being persuaded by the faulty, subsequently retracted argument about Safford not being a valid prior-art reference under 35 U.S.C. § 103(a), (2) the Examiner believing that a hot spare processor that is held in reserve

distinguished independent claim 1 from the cited art under 35 U.S.C. §§ 102 & 103, (3) the Examiner believing that an operating system recognizing the spare processor as the boot processor without requiring a shutdown distinguished independent claim 10 from the cited art under 35 U.S.C. §§ 102 & 103, and (4) possibly the Examiner being partially persuaded by the faulty, repeated argument about Safford failing to disclose a boot processor.

Accordingly, the Examiner missed a few significant issues that led to the allowance of the '077 Application and the issuance of the '781 Patent. First, as the Applicant subsequently and correctly admitted on the record, Safford was a valid prior-art reference under 35 U.S.C. § 103(a), so original claims 18-32 [issued claims 16-30] should not have been allowed as a result of the argument that Safford was not a valid prior-art reference under 35 U.S.C. § 103(a). Indeed, as the Applicant explicitly retracted that argument on the record, original claims 18-32 were allowed even though the Applicant never made any viable arguments against the Examiner's original obviousness rejections raised against claims 18-32 in the first OA.

Second, Safford explicitly disclosed a hot-standby processor pair that is held idly in reserve until the first processor pair experiences an error, so claims 1-5 and 7-9 [issued claims 1-8] should not have been allowed as a result of the amending claim 1 to clarify that the spare processor is a hot spare held in reserve. *See Safford* (Ex. 1004) at 4:25-30. Third, Fox explicitly disclosed failing over to a hot-spare processor without affecting the operating system, so claims 10-13 and 15-17 [issued claims 9-16] should not have been allowed as a result of amending claim 10 to clarify that an operating system recognizes the spare processor as the boot processor without requiring a shutdown. *See Fox* (Ex. 1006) at 4:16-25. Fourth, to the extent that the Examiner found persuasive the repeated argument about Safford failing to disclose a boot processor, the Examiner did not recognize Safford's disclosure of a boot processor in the form of "logical CPU 0". *See, e.g., Safford* (Ex. 1005) at 3:4-5, 4:52-54; *see also infra* §§ I.C.2, I.F.1, I.H.2, I.H.3, II.B.1; *see also Wolfe Decl.* (Ex. 1003) ¶¶ 47, 49. For example, Safford opted for the designator "logical CPU 0," which was commonly used to refer to a system's boot processor, instead of using the word "boot processor" itself. *See, e.g., Safford* (Ex. 1005) at 3:4-5, 4:52-54; *see also Wolfe Decl.* (Ex. 1003) ¶ 47, 49.

The '781 Patent was subject to a terminal disclaimer filed on September 10, 2008, which disclaimed any term beyond U.S. Patent Application No. 10/972,588 filed on October 25, 2004, and U.S. Patent No. 7,366,948 also filed on October 25, 2004.

B. Level of Ordinary Skill in the Art

A Person of Ordinary Skill in The Art ("POSITA") would have had, as of October 25, 2004: (1) at least a Bachelor's Degree in electrical engineering, electronics engineering, computer engineering, or a related scientific field; and (2) at least one to two years of experience in the computer science field, including chip design and software engineering, with additional education substituting for less experience and vice versa. *Wolfe Decl.* (Ex. 1003) ¶ 45.

C. Claim Construction

1. Overview

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

2. "boot processor"

Claims 1, 3, 7, 9, 10, 12-14, 16, 17, 19, 21-23, 25-27, 29, and 30 recite a "boot processor," which should be given the broadest reasonable interpretation that is both "consistent with the ordinary and customary meaning of the term" and "consistent with the use of the term in the specification and drawings." *See* M.P.E.P. § 2111. Under this standard, a boot processor is a common component that was found in running computer systems at the time of the alleged invention of the '781 Patent and would have been understood by a POSITA to include **any processor designated for handling bootstrap functions**. *See Wolfe Decl.* (Ex. 1003) ¶¶ 47-49. Furthermore, a boot processor can also be referred as a bootstrap processor (BSP) or by a computer-readable designator (*e.g.*, logical CPU 0). *Id.*

As noted, a boot processor was a critical device for any computing system and was a common component included in computing systems. Indeed, "[t]he execution of the bootstrap program is always automatically initiated when the system is powered-on or when a system hardware reset is applied. It is also initiated when the system is rebooted from the system console." *Levi* (Ex. 1012) § 4.2.1. Furthermore, "[t]he origin of the word boot (as in, 'to boot the system') is

bootstrapping, which is the process of bringing a computer system to life and ready to use. ('Bootstrapping' is actually the 'nerd word' for starting up a computer.)." *Id.*

As of the priority date, a POSITA would have understood that in multi-processor system architectures, a boot processor may either be selected by default or through a selection process during bootup. For example, "one default boot CPU is designated to boot the system." *Kennedy* (Ex. 1005) at 2:18-20. Alternatively, as noted in Intel's Itanium Manual, which the '781 Patent incorporates by reference, "firmware selects a Bootstrap processor (BSP) in multi-processor (MP) configurations early in the boot sequence." Ex. 1014 § 24.1.1. And in some systems (*e.g.,* fault-tolerant systems), the processor designated as the boot processor may change over time. For example, the Intel MultiProcessor Specification explains how an operating system can determine and track the ID of a designated BSP:

> While all processors in an MP-compliant system are functionally identical, one of the processors will be designated as the bootstrap processor (BSP) at system initialization by the system hardware or by the system hardware in conjunction with the BIOS. The rest of the processors are designated as the application processors (APs). The BSP is responsible for booting the operating system. Once the MP operating system is up and running, the BSP functions as an AP. Usually a processor is designated as the BSP because it is capable of controlling all system hardware, including AP startup and shutdown. The operating system must determine and remember the APIC ID of the designated BSP, so it can keep the BSP operating as the last running processor during system shutdown. The BSP is not necessarily the first processor, especially in fault-tolerant MP systems in which any available processor can be designated as the BSP.

Ex. 1015 § B.1.

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

> If CPU P0 is not functioning properly and cannot perform even the most basic functions, it is designated as inoperative. The hardware system then awakens CPU P1 and repeats the process of testing the CPU. If CPU P1 is operational, then it is designated as CPU L0, and it boots the remainder of the system. On the other hand, if CPU P1 also fails, it is also given an inoperative designation. The computer system then turns on CPU P2, and repeats the process. The process repeats until an operational microprocessor is found to perform the CPU L0 functions.

Cepulis (Ex. 1013) at 2:52-64.

The proposed interpretation of boot processor is consistent with the context provided by the '781 Patent, which references the boot process of an Itanium Processor Family (IPF) system: "The boot-up process of a traditional IPF system, for example, proceeds as follows: when the system is first powered on, there are some sanity checks (*e.g.,* power on self-test) that are performed by microprocessors included in the system platform, which are not the main system processors that run applications. After those checks have passed, power and clocks are given to a boot processor (which may, for example, be master processor 12A)." Ex. 1001 at 11:60-68. As another indicator that the '781 Patent's usage of boot processor is consistent with the ordinary and customary meaning, the '781 Patent discusses using the process outlined in the Intel Multiprocessor Specification to switch the role of a boot processor by "changing a system identifier, used by the system's operating system to recognize the boot processor…to a different value." *Id.*, claim 1.

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

D. List of Prior Art Patents and Printed Publication

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

- **Ex. 1004 ("Safford"):** Safford is a patent that issued on August 1, 2006. Safford derived from a patent application filed on July 3, 2002. The contents of Safford were published as U.S. Patent Publication No. 2004/0006722 on January 8, 2004. Safford qualifies as prior art under at least 35 U.S.C. §§ 102(a) and 102(e) and is a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a).
- **Ex. 1005 ("Kennedy"):** Kennedy is a patent that issued on September 12, 1995. Kennedy derived from a patent application filed on April 26, 1993. Kennedy qualifies as prior art under at least 35 U.S.C. §§ 102(a), 102(b), and 102(e) and is a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a).

- **Ex. 1006 ("Fox"):** Fox is a patent that issued on July 31, 2007. Fox derived from a patent application filed on January 4, 2004. The contents of Fox were published as U.S. Patent Publication No. 2005/0172164 on August 4, 2005, by International Business Machines Corporation. Fox qualifies as prior art under at least 35 U.S.C. §§ 102(e) and is a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a).
- **Ex. 1007 ("Kalyanasundharam"):** Kalyanasundharam is a patent that issued on February 5, 2007. Kalyanasundharam derived from a patent application filed on October 15, 2004, by Advanced Micro Devices, Inc. Kalyanasundharam qualifies as prior art under at least 35 U.S.C. §§ 102(e) and is a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a).
- **Ex. 1008 ("Nguyen"):** Nguyen is a patent publication that was published on June 24, 2004. Nguyen derived from a patent application filed on December 19, 2002. Nguyen qualifies as prior art under at least 35 U.S.C. §§ 102(a) and 102(e) and is a valid prior-art reference for the purpose of establishing an obviousness rejection under 35 U.S.C. § 103(a).
- **Ex. 1013 ("Cepulis")**: Cepulis is a patent that issued on February 13, 1996. Cepulis derived from a patent application filed on September 10, 1993. Cepulis qualifies as prior art under at least 35 U.S.C. §§ 102(a) and 102(b).

E. Ground 1: Kalyanasundharam in view of Kennedy Presents Substantial New Questions of Patentability for Claims 1-6 and 9-15

1. Overview of Kalyanasundharam

Kalyanasundharam generally described a redundant processing system that establishes multiple processing core pairs "in which at least a first processor core is redundantly executing code that a second processor core is also executing" and that "compare[s] communications from the first processor core to communications from the second processor core to verify correct execution of the code." *Kalyanasundharam* (Ex. 1007), Abstract.

Kalyanasundharam's redundant processing system implements processors that operate in lockstep. Id., 4:15-17. Kalyanasundharam indicates that ". . . the processor cores 18A-18B may execute in lockstep (that is, may execute the same instructions on the same clock cycle)." *Id.* Such is illustrated in FIG. 1, in which a pair of processor cores 18A and 18B are shown to

operate in lockstep as controlled by node controller 20A.

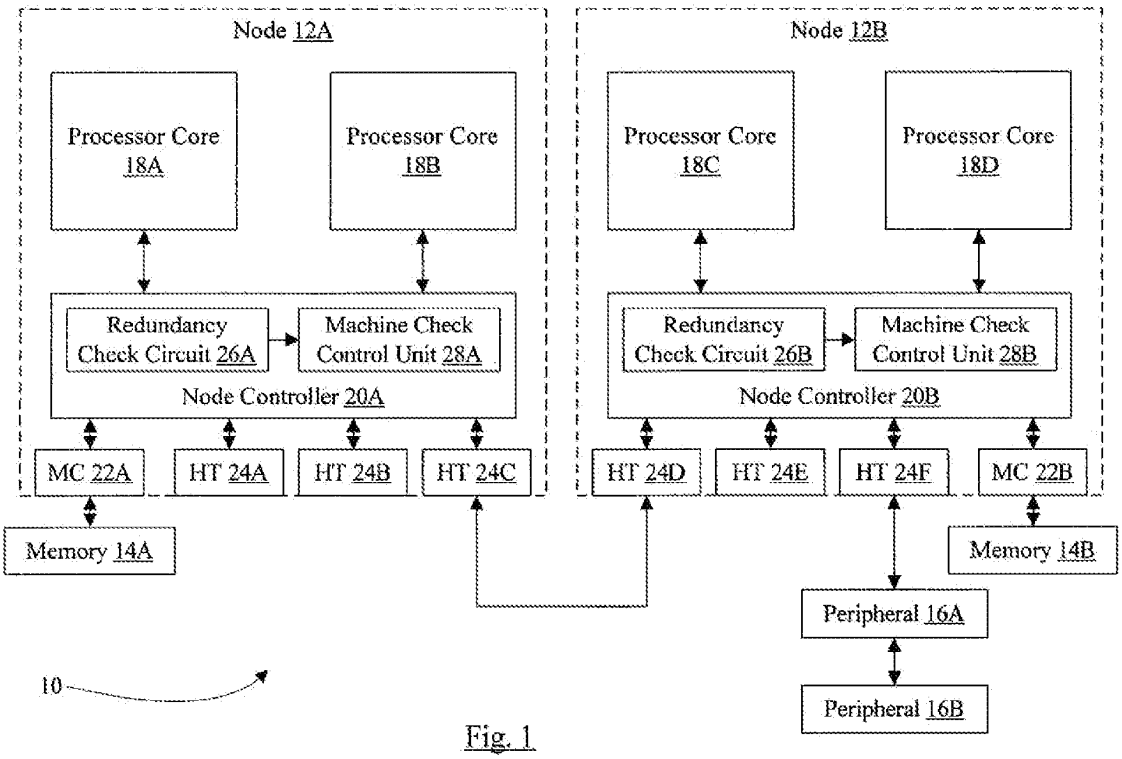


Fig. 1

Kalayanasundharam (Ex. 1008), FIG. 1.

Furthermore, as in the '781 Patent, which designates one processor pair as master and the remaining processor pairs as slaves, Kalyanasundharam's system designates one processor pair as a "primary processor core" and refers to other processor cores as "auxiliary processor cores." *Id.* at 4: 58-61. Kalyanasundharam further defines the primary and auxiliary cores, indicating that "[t]he primary processor core may be the processor core whose execution results will be used as the result of executing the redundantly-executed software code." *Id.* at 4:61-63. "The auxiliary processor core is used to verify correct execution by the primary processor core." *Id.* at 4:63-65.

Like the '781 Patent, however, Kalyanasundharam notes that problems can occur when processors lose lockstep with each other. Kalyanasundharam states that, "An execution error may be any indication that an error has occurred in the execution of the software code, detected via the redundant execution." *Id.* at 3:63-65. "In the embodiment of FIG. 1, it may not be clear which processor core 18A-18B experienced the error, but the error is detected." *Id.* at 3:65-67.

"Diagnostic software may be used to determine which processor core 18A-18B experienced the error and what corrective measures are to be taken." *Id.* at 3:67-4:3.

Once Kalyanasundharam's system identifies which processor core experienced the error, it will disable the primary processor core and select one of the auxiliary processor cores to become the new primary processor core. *Id.* at 13:16-22. Thus, just as the '781 Patent determines whether a boot processor experienced an error and then switches the boot processor to a spare processor, Kalyanasundharam's system will "disable the primary processor core and select one of the auxiliary processor cores to be the primary processor core (block 102)." *Id.* This is shown below in step 102 of Kalyanasundharam's FIG. 6. In this manner, Kalyanasundharam establishes primary and secondary processor cores and, upon detection of an error at the primary processor core, switches processing from the primary processor core to a secondary processor core. *Id. at* 4:15-25; *see also* 4:61-65.

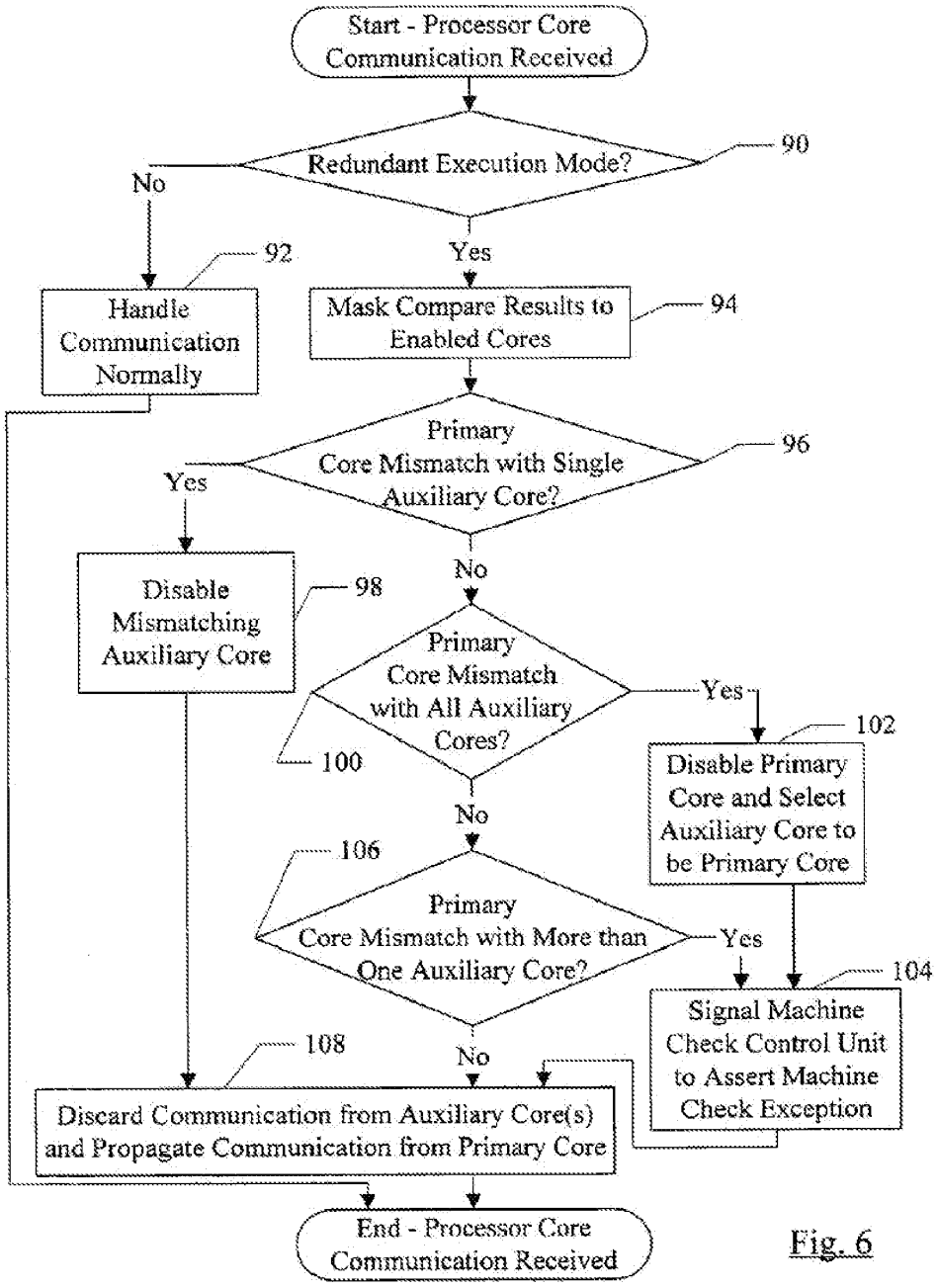


Fig. 6

Kalyanasundharam (Ex. 0007), Fig. 6

Kalyanasundharam is analogous art to the '**781** Patent because Kalyanasundharam is from the same field of endeavor (*e.g.*, providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '**781**

(*e.g.*, processor failures and high availability in multiprocessor systems). *Id.* at 1:20-35; *see also '781 Patent* 4:34-53; *Wolfe Decl.* (Ex. 1003) ¶ 54. For example, in focusing on "the increased desire for many systems to maintain high availability", the '781 Patent notes that the problem of "crashing the system each time LOL is detected is not an attractive proposition." Ex. 1001 at 3:16-18. Similarly, in focusing on "implement[ing] high reliability by including multiple processors", Kalyanasundharam notes that "controlling the processors to execute in lockstep…is complex and error prone." *Kalyanasundharam* (Ex. 0007) at 1:23, 1:31-34.

2. Overview of Kennedy

Kennedy generally described a high-availability multi-processor computing system for "controlling initialization and self-test operations." Ex. 1005, Abstract. Like the '781 Patent's high-availability processing system, the multi-processor system in Kennedy is configured "to automatically transfer default system initialization operations from a default processor to a first alternative processor if the default processor fails, and to automatically transfer initialization operations to a second alternative processor if the first alternative processor fails, and so forth depending upon how many alternative processors are installed in the multiprocessor system." *Id.*

Like the assignment of a boot processor role to a given processor in the '781 Patent, Kennedy determines which processor will boot the system and designates that processor as the "default boot processor." *Id.* at 7:7-10. Kennedy states that "one default boot CPU is designated to boot the system, and if this CPU fails, control transfers automatically to an alternate processor." *Id.* at 2:17-21. Accordingly, Kennedy similarly designates "boot" and "alternate" processors. In reference to FIG. 1 below, Kennedy states that, "slot 1 contains the default boot CPU." *Id.* at 10:64-65. Further, Kennedy taught that "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system." *Id.* at 7:9-13.

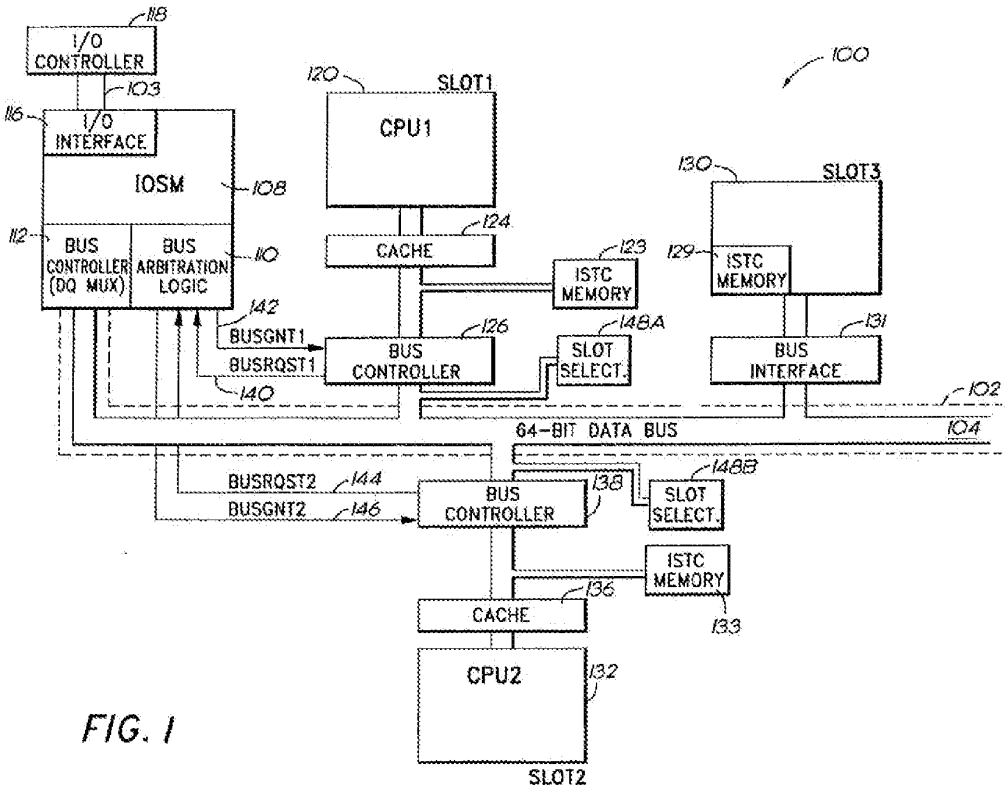


FIG. 1

Kennedy (Ex. 1005), Fig. 1

Kennedy specifically explains that "**the CPU2 132 takes over default boot operations as an alternative boot processor instead of CPU1 120.**" *Id.* at 13:38-42. Kennedy continues, "**If the CPU1** 120 does boot properly, then the CPU2 132 checks in as an additional processor rather than taking over **as a boot processor.**" *Id.* at 13:42-44. Thus, Kennedy designates a boot processor and, upon detecting a failure in the boot processor, switches to an alternative processor. *See id.* at 2:29-38, 2:50-56.

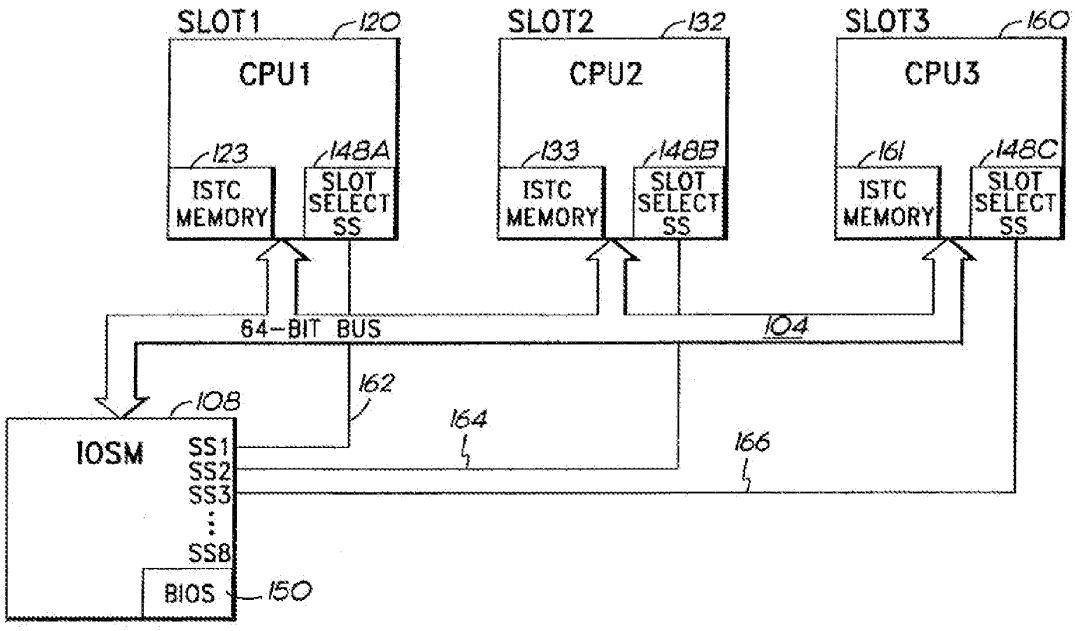


FIG. 3

Kennedy (Ex. 1005), Fig. 1

Kennedy is analogous art to the '781 Patent because Kennedy is from the same field of endeavor (*e.g.*, recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (*e.g.*, processor failures in multiprocessor systems). *Id.* at 2:15-21; ***see also '781 Patent*** (Ex. 1001) at 4:34-53; *see also Wolfe Decl.* (Ex. 1003) ¶ 56. For example, in focusing on "the increased desire for many systems to maintain high availability", the '781 Patent notes that the problem of "crashing the system each time LOL is detected is not an attractive proposition." Ex. 1001 at 3:16-18. Similarly, in focusing on "one critical determination", Kennedy notes that, "if one processor fails, provision for disabling the non-functional processor to boot the system enhances the availability of the system." Ex. 1005 at 10:4-7.

3. Obviousness of the '781 Patent in view Kalyanasundharam and Kennedy

Claims 1-6 and 9-15 of the '781 Patent would have been rendered obvious to a POSITA in view of Kalyanasundharam and Kennedy. As will be described below, this conclusion of

obviousness is supported by one or more motivations to combine Kalyanasundharam and Kennedy as proposed, a reasonable expectation of success for the proposed combination of Kalyanasundharam and Kennedy, and various *KSR* rationales detailed throughout M.P.E.P. § 2143.

(I) Overview of Motivation to Combine Kalyanasundharam and Kennedy as Proposed

As mentioned above, Kalyanasundharam described replacing a failed primary processor with an auxiliary processor in response to a detected lockstep mismatch. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Even if Kalyanasundharam did not explicitly identify the failed primary processor as having been the boot processor, Kalyanasundharam as modified by Kennedy would have rendered boot processor failover obvious. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy described failing over a boot processor to an alternate boot processor. *See id.* Accordingly, Kalyanasundharam as modified by Kennedy taught replacing a failed boot processor with an auxiliary processor in response to a detected lockstep mismatch.

A POSITA would have been motivated to combine the teachings of Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system (*e.g.*, in the event that the boot processor fails). *Wolfe Decl.* (Ex. 1003) ¶ 58. For example, a POSITA would have been motivated to modify Kalyanasundharam's system, which implements general processor failover, by Kennedy's teachings about boot processor failover. *Wolfe Decl.* (Ex. 1003) ¶ 59. A POSITA would have been motivated to do so because Kennedy's failover capabilities would provide Kalyanasundharam's system with additional fault tolerance and high availability (*e.g.*, in the event that the boot processor fails). *Wolfe Decl.* (Ex. 1003) ¶ 60. Indeed, a POSITA would have recognized the obvious benefits and advantages of enabling Kalyanasundharam's system to dynamically replace a failed boot processor, especially in view of Kennedy's express suggestion that doing so "provides fault tolerance and high availability." *Kennedy* (Ex. 1005) at 2:21-22; *see also Wolfe Decl.* (Ex. 1003) ¶ 61. Consistent with the Supreme Court's holding in *KSR*, the rationales provided below further support this conclusion of obviousness.

(II) Reasonable Expectation of Success

A POSITA would have had a reasonable expectation of success in combining Kalyanasundharam and Kennedy. *Wolfe Decl.* (Ex. 1003) ¶ 62. For example, Kalyanasundharam

described replacing a failed primary processor with an auxiliary processor in response to a detected lockstep mismatch. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Even if Kalyanasundharam did not explicitly identify the failed primary processor as having been a boot processor, Kalyanasundharam as modified by Kennedy would have rendered boot processor failover obvious. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Moreover, Kalyanasundharam's failed primary processor could have been the boot processor of Kalyanasundharam's fault-tolerant system even though that detail is not explicitly stated in Kalyanasundharam. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 63. Indeed, and as explained in detail above, a boot processor was a common component of computer systems, such as Kalyanasundharam's system. *See supra* § I.C.2. Because a boot processor was a common component in computer systems (*e.g.*, Kalyanasundharam's system) and Kennedy's specific teachings are merely directed to failing over the boot processor, the proposed combination of Kalyanasundharam and Kennedy would have been expected to succeed. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 64. It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in the manner proposed, and Kalyanasundharam did not teach away from or discourage applying such a modification. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 65. That is, Kalyanasundharam does not limit or otherwise discourage application of its techniques to a "boot processor." Accordingly, the proposed combination of Kalyanasundharam and Kennedy, in which Kalyanasundharam's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed. *Id.*

(III) *KSR* Rationales

Consistent with the Supreme Court's holding in *KSR* and M.P.E.P. § 2143, the following rationales further support this conclusion of obviousness in view of Kalyanasundharam and Kennedy.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

A POSITA would have combined the teachings of Kalyanasundharam and Kennedy according to known methods to yield predictable results. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 67. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed each element claimed, (2) a POSITA could have combined the prior-art elements by known methods, (3) each prior-art element performs the same function in isolation and in the

combination, and (4) a POSITA would have recognized the results of the proposed combination as being predictable. As shown below, the proposed combination of Kalyanasundharam and Kennedy satisfies all the prongs of this rationale.

First, the combination of Kalyanasundharam and Kennedy disclosed each element recited in the applicable claims. *See, e.g., Appendix A* (Ex. AA).

Second, a POSITA would have combined the prior-art elements by known methods. *Wolfe Decl.* (Ex. 1003) ¶¶ 67-70. For example, Kalyanasundharam described a fault-tolerant system that replaces a failed primary processor with an auxiliary processor in response to a detected lockstep mismatch. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Kennedy described failing over boot processors to spare processors. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13. Specifically, Kennedy disclosed a known method for transferring the role of boot processor to a spare processor. *See id.*; *see also Wolfe Decl.* (Ex. 1003) ¶ 69. A POSITA would have understood Kalyanasundharam as disclosing a boot processor and failing over the same to an auxiliary processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 69. However, to the extent that Kalyanasundharam is interpreted to not explicitly disclose a "boot processor", a POSITA would have combined Kalyanasundharam's fault-tolerant system with Kennedy's boot processor by the method disclosed in Kennedy to transfer the role of boot processor to Kalyanasundharam's auxiliary processor. *Wolfe Decl.* (Ex. 1003) ¶ 69.

Third, each prior-art element performs the same function in isolation and in the combination. *Wolfe Decl.* (Ex. 1003) ¶ 70. For example, in isolation and in the combination, Kalyanasundharam's fault-tolerant system replaces a failed primary processor with an auxiliary processor, and Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure. *Id.* Accordingly, the functions performed by the prior-art elements do not change from the individual Kalyanasundharam and Kennedy references to the proposed combination.

Fourth, a POSITA would have recognized the results of the proposed combination as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 70. For example, a POSITA would have recognized that, by replacing Kalyanasundharam's failed primary processor with Kennedy's failed boot processor, Kalyanasundharam's system would have been able to predictably provide boot processor failover, as opposed to just primary processor failover. *Wolfe Decl.* (Ex. 1003) ¶ 70.

Simple Substitution of One Known Element for Another to Obtain Predictable Results

A POSITA would have performed a simple substitution of Kalyanasundharam's processor with Kennedy's boot processor to obtain predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 71. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a device that differed from the claimed device by the substitution of some components with other components, (2) the substituted components and their functions were known in the art, (3) a POSITA could have substituted one known element for another, and (4) the results of the substitution would have been predictable. As shown below, the proposed combination of Kalyanasundharam and Kennedy satisfies all the prongs of this rationale.

First, to the extent that Kalyanasundharam is interpreted as differing from the claim limitations of the '781 Patent, Kalyanasundharam disclosed a device that differed from the claimed device by the substitution of some components with other components. For example, Kalyanasundharam disclosed a device that replaces a failed primary processor with an auxiliary processor. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Similarly, the claimed device transfers the role of boot processor to a spare processor in response to a detected lockstep mismatch. *'781 Patent* (Ex. 1001), claim 1. Accordingly, to the extent that Kalyanasundharam is not interpreted as explicitly disclosing a "boot processor," the substitution of Kalyanasundharam's failed primary processor with a failed boot processor would obviate the difference between Kalyanasundharam and the claimed device. *Wolfe Decl.* (Ex. 1003) ¶ 72.

Second, the substituted components and their functions were known in the art. *Wolfe Decl.* (Ex. 1003) ¶ 73. For example, the substituted boot processor from Kennedy and its functions were known in the art. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 73. As described in Kennedy, a "default boot CPU is designated to boot the system." *Kennedy* (Ex. 1005) at 2:18-19. Accordingly, the prior art shows that the substituted boot processor was known to boot the system.

Third, a POSITA would have substituted one known element for another. *Wolfe Decl.* (Ex. 1003) ¶ 74. For example, it was well within the skill of a POSITA to merely substitute one type of processor for another (*e.g.,* Kalyanasundharam's failed primary processor for Kennedy's failed boot processor). *Id.*

Fourth, a POSITA would have recognized the results of the substitution as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 75. For example, a POSITA would have recognized that, by substituting Kalyanasundharam's failed primary processor for Kennedy's failed boot

processor, Kalyanasundharam's device would have been able to predictably provide boot processor failover. *Id.*

Use of Known Technique to Improve Similar Devices in the Same Way

A known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA. *Wolfe Decl.* (Ex. 1003) ¶ 76. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention, (3) a POSITA could have applied the known improvement technique in the same way to the base device, and (4) the results would have been predictable to a POSITA. As shown below, the proposed combination of Kalyanasundharam and Kennedy satisfies all the prongs of this rationale.

First, Kalyanasundharam disclosed a base device upon which the claimed invention can be seen as an improvement. *Wolfe Decl.* (Ex. 1003) ¶ 77. For example, Kalyanasundharam disclosed a device that replaces a failed primary processor with an auxiliary processor in response to a detected lockstep mismatch. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Similarly, the claimed invention transfers the role of boot processor to a spare processor in response to in response to a detected lockstep mismatch. *'781 Patent* (Ex. 1001), claim 1. Accordingly, to the extent that Kalyanasundharam does not explicitly disclose applying its techniques specifically to a "boot processor," an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed primary processor. *Wolfe Decl.* (Ex. 1003) ¶ 77.

Second, the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 78. For example, Kennedy disclosed a comparable device that transfers control from a failed boot processor to an alternate processor. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 78. Because Kennedy's device transfers control from the failed boot processor to the alternate processor, Kennedy's device has been improved relative to Kalyanasundharam's device in the same way as the claimed invention. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 78.

Third, a POSITA would have applied the known improvement technique in the same way to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 79. For example, a POSITA would have improved

Kalyanasundharam's device by applying Kennedy's technique to fail over a boot processor, as opposed to failing over a primary processor. *Id.*

Fourth, a POSITA would have recognized the results of the applied improvement technique as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 80. For example, a POSITA would have recognized that, by applying Kennedy's known technique, Kalyanasundharam's device would have been able to predictably provide boot processor failover. *Id.*

Applying a Known Technique to a Known Device Ready for Improvement

A known technique that is recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 81. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a known technique that is applicable to the base device, (3) a POSITA would have recognized that applying the known technique would have yielded predictable results, and (4) the predictable results would have improved the base device. As shown below, the proposed combination of Kalyanasundharam and Kennedy satisfies all the prongs of this rationale.

First, Kalyanasundharam disclosed a base device upon which the claimed invention can be seen as an improvement (if Kalyanasundharam is interpreted to somehow limit its disclosure to processors other than "boot processors"). *Wolfe Decl.* (Ex. 1003) ¶ 82. For example, Kalyanasundharam disclosed a device that replaces a failed primary processor with an auxiliary processor in response to a detected lockstep mismatch. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Similarly, the claimed invention transfers the role of boot processor to a spare processor in response to a detected lockstep mismatch. *'781 Patent* (Ex. 1001), claim 1. Accordingly, an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed primary processor. *Wolfe Decl.* (Ex. 1003) ¶ 82.

Second, the prior art also disclosed a known technique that is applicable to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 83. For example, Kennedy disclosed a known technique for transferring control of a device from a failed boot processor to an alternate processor. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 83. Because Kennedy's known technique involves transferring control of the device from the failed boot processor to the alternate processor, and Kennedy's device is very similar to and includes the

same components and functionality as Kalyanasundharam's device, Kennedy's known technique is applicable to Kalyanasundharam's device. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 83.

Third, a POSITA would have recognized that applying the known technique to the base device would have yielded predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 84. For example, a POSITA would have recognized that applying Kennedy's known technique to Kalyanasundharam's device would have yielded predictable results. *Id.* More specifically, a POSITA would have recognized that, by applying Kennedy's known technique, Kalyanasundharam's device would have been able to predictably provide boot processor failover. *Id.* Moreover, because Kalyanasundharam already disclosed applying the same failover technique to processors in general, a POSITA would have expected the application of Kennedy's technique on Kalyanasundharam's device to be successful.

Fourth, the predictable results would have improved the base device. *Wolfe Decl.* (Ex. 1003) ¶ 85. For example, the predictable results would have improved the reliability of Kalyanasundharam's device (*e.g.*, in the event that the boot processor fails). *Id.*

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to Combine Prior-Art Teachings to Arrive at the Claimed Invention

One or more teachings, suggestions, or motivations in Kennedy would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy to arrive at the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 86. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) there was some teaching, suggestion, or motivation either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the references' teachings and (2) there was a reasonable expectation of success for the combination. As shown below, the proposed combination of Kalyanasundharam and Kennedy satisfies all the prongs of this rationale.

First, Kennedy emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy in this way. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 87. For example, Kennedy taught that its technique for failing over boot processors "provides fault tolerance and high availability for the system." *See, e.g., Kennedy* (Ex. 1005) at 2:21-22. Further, Kennedy taught that, "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system."

Kennedy (Ex. 1005) at 7:9-13. Kennedy also taught that, "if one processor fails, provision for disabling the non-functional processor and allowing another alternative processor to boot the system enhances the availability of the system." *Id.* at 10:4-7. Any or all of these teachings, suggestions, and/or motivations from Kennedy would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy in this way. *Wolfe Decl.* (Ex. 1003) ¶ 87.

Additionally or alternatively, a POSITA would have had the knowledge to combine the teachings of Kalyanasundharam and Kennedy so as to enable Kalyanasundharam's system to dynamically replace a failed boot processor, as opposed to dynamically replacing a failed primary processor. *Wolfe Decl.* (Ex. 1003) ¶ 88. Even disregarding the express teachings, suggestions, and/or motivations cited from Kennedy, a POSITA's knowledge alone constitutes a sufficient motivation to combine Kalyanasundharam and Kennedy in this way. *See, e.g.,* M.P.E.P. § 2143.I(G); *see also DyStar Textilfarben GmbH & Co. Deutschland KG v. C.H. Patrick Co.,* 464 F.3d 1356, 1360 (Fed. Cir. 2006). "Because the desire to enhance commercial opportunities by improving a product or process is universal—and even common-sensical—we have held that there exists in these situations a motivation to combine prior art references even absent any hint of suggestion in the references themselves." *Id.* "In such situations, the proper question is whether the ordinary artisan possesses knowledge and skills rendering him *capable* of combining the prior art references." *Id.* A POSITA would have possessed the knowledge and skills to combine the teachings of Kalyanasundharam and Kennedy in this way. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 88.

Second, there was a reasonable expectation of success for the proposed combination of Kalyanasundharam and Kennedy. *See Wolfe Decl.* (Ex. 1003) ¶ 88. For example, Kalyanasundharam described replacing a failed primary processor with an auxiliary processor. *See, e.g., Kalyanasundharam* (Ex. 1007) at 4:15-18, 13:15-29. Even if Kalyanasundharam did not explicitly identify the failed primary processor as having been a boot processor, Kalyanasundharam as modified by Kennedy would have rendered boot processor failover obvious. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Moreover, Kalyanasundharam's general disclosure of a failed primary processor would have been understood to include the boot processor of Kalyanasundharam's fault-tolerant system even though that detail is not explicitly stated in Kalyanasundharam. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 88. Kalyanasundharam does not discourage or otherwise limit its disclosure to processors other than "boot processors," and it was

well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in the manner proposed. *See id.* Accordingly, the proposed combination of Kalyanasundharam and Kennedy, in which Kalyanasundharam's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed. *Id.*

F. Ground 2: Safford in view of Kennedy Presents Substantial New Questions of Patentability for Claims 1-6 and 9-15

1. Overview of Safford

Safford generally described a multi-processor system capable of recovering from a loss of lockstep detected in connection with a certain processor pair. *See, e.g., Safford* (Ex. 1004), Abstract. As explained in Safford, "to improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a master/check pair." *Id.* at 2:55-57. "Each of two processors in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit located near external interfaces of the two processors." *Id.* at 2:57-60. "Any difference in the processor outputs indicates the existence of an error." *Id.* at 2:60-63. The configuration of Safford's system, whose processors operate in lockstep, is illustrated in FIG. 2, and is reproduced below for convenience.

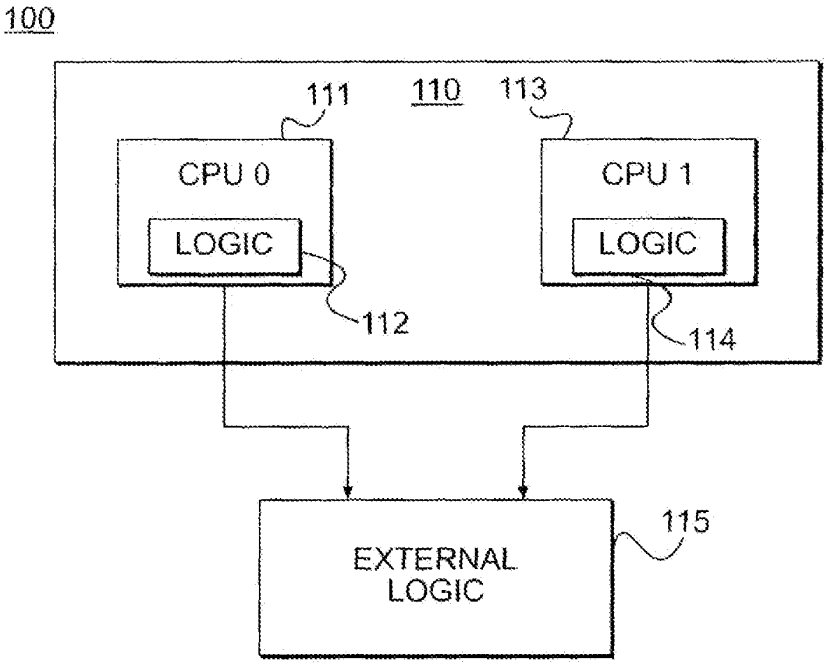


FIG. 2

Safford (Ex. 1004), FIG. 2

Safford's system includes data storage for storing information that assigns different roles to the processors. For example, in Safford's system, "the architected states of each of the microprocessors 22 and 24 is copied from the memory 25 into the microprocessors 22 and 24, respectively." *Id.* at 2:2-4. "[T]he node controller 130 may store the architected state of the processors 111, 113, 121, 123, 125, and 127, either internally in the node controller 130, or in another component of the computer system 100." *Id.* at 4:6-10. "The node controller 130 has available at all time [*sic*] a current architected state of the processors to which the node controller 130 is coupled." *Id.*

Further, Safford's system includes logic for changing the roles of the processors in response to a loss of lockstep. For example, "[u]sing the impending loss of lock step signal, the computer system 100 may continue operating (processing) using one of the processors 111 and 113 that did not experience an error." *Id.* at 3:27-30. In addition, Safford's system recovers from

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

To change the roles of the processors in this way, **Safford's system follows the process** illustrated in FIG. 4, which is reproduced below for convenience.

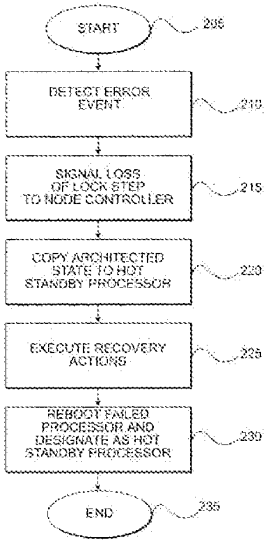


FIG. 4

Safford (Ex. 1004), Fig. 4

Safford is analogous art to the **'781** Patent because Safford is both from the same field of endeavor (*e.g.*, providing recovery from failure in a high-availability computing system) as the **'781 Patent** and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (*e.g.*, system crashes caused by loss of lockstep). *See Wolfe Decl.* (Ex. 1003) ¶ 91. For example, in **focusing on "the increased desire for many systems to maintain high availability", the '781 Patent notes the problem that "crashing the system each time LOL is detected is not an attractive**

proposition." Ex. 1001 at 3:16-18. Similarly, in focusing on "improv[ing] availability of the processor assets of the computer system", Safford notes that "loss of lock step, if not promptly corrected, may cause the computer system 18 to crash." Ex. 1004 at 3:21-22, 1:45-47.

2. Overview of Kennedy

An overview of Kennedy appears above. *See supra* § I.E.2. That overview is equally applicable to Ground 2 and will not be repeated here for the sake of brevity.

3. Obviousness of the '781 Patent in view of Safford and Kennedy

Claims 1-6 and 9-15 of the '781 Patent would have been rendered obvious to a POSITA in view of Safford and Kennedy. As will be described below, this conclusion of obviousness is supported by a reasonable expectation of success for the proposed combination of Safford and Kennedy and various *KSR* rationales detailed throughout M.P.E.P. § 2143.

(I) Overview of Motivation to Combine Safford and Kennedy

As mentioned above, Safford described replacing a failed processor with a hot spare processor in response to a loss of lockstep. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Even if Safford did not explicitly identify the failed processor as having been the boot processor[1], Safford as modified by Kennedy would have rendered obvious the application of Safford's methods to a boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13; *Wolfe Decl.* (Ex. 1003) ¶ 93. For example, Kennedy described failing over a boot processor to an alternate boot processor. *See id.* Accordingly, Safford as modified by Kennedy taught replacing a failed boot processor with a hot spare processor in response to a loss of lockstep.

A person of ordinary skill in the art ("POSITA") would have been motivated to combine the teachings of Safford and Kennedy to improve the reliability of Safford's system (*e.g.*, to provide high availability in the event that the boot processor fails). *Wolfe Decl.* (Ex. 1003) ¶ 93. For example, a POSITA would have been motivated to modify Safford's system, which implements general processor failover, by Kennedy's specific teachings about boot processor failover. *Wolfe Decl.* (Ex. 1003) ¶ 93. A POSITA would have been motivated to do so because

[1] Requester notes that Safford did disclose a boot processor in the form of "logical CPU 0". *See, e.g., Safford* (Ex. 1005) at 3:4-5, 4:52-54; *see also Wolfe Decl.* (Ex. 1003) ¶ XX. Safford opted for the designator "logical CPU 0," which was commonly used to refer to a system's boot processor, instead of using the word "boot processor" itself. *See, e.g., Safford* (Ex. 1005) at 3:4-5, 4:52-54; *see also Wolfe Decl.* (Ex. 1003) ¶ XX.

Kennedy's failover capabilities would provide Safford's system with additional fault tolerance and high availability (*e.g.*, in the event that the boot processor fails). *Wolfe Decl.* (Ex. 1003) ¶ 93. Indeed, a POSITA would have recognized the obvious benefits and advantages of enabling Safford's system to dynamically replace a failed boot processor, especially in view of Kennedy's express suggestion that doing so "provides fault tolerance and high availability." *Kennedy* (Ex. 1005) at 2:21-22; *see also Wolfe Decl.* (Ex. 1003) ¶ 93. Consistent with the Supreme Court's holding in *KSR*, the rationales provided below further support this conclusion of obviousness.

(II) Reasonable Expectation of Success

There would have been a reasonable expectation of success for the proposed combination of Safford and Kennedy. *Wolfe Decl.* (Ex. 1003) ¶ 94. For example, Safford described replacing a failed processor with a hot spare processor. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Even if Safford did not explicitly identify the failed processor as having been a boot processor, Safford as modified by Kennedy would have rendered obvious the application of Safford's methods to a boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Moreover, Safford's failed processor could have been the boot processor of Safford's system even though that detail is not explicitly stated in Safford. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 95. Indeed, and as explained in detail above, a boot processor was a common component of computer systems, such as Safford's system. *See supra* § I.C.2. Because a boot processor was a common component in computer systems (*e.g.*, Safford's system) and Kennedy's teachings are specifically directed to failing over the boot processor, the proposed combination of Safford and Kennedy would have been expected to succeed. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 95. It was well within the skill of a POSITA to modify Safford's system by Kennedy in the manner proposed, and Safford did not teach away from or discourage applying such a modification. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 95. That is, Safford does not limit or otherwise discourage application of its techniques to a "boot processor." Accordingly, the proposed combination of Safford and Kennedy, in which Safford's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed. *Id.*

(III) *KSR* Rationales

Consistent with the Supreme Court's holding in *KSR* and M.P.E.P. § 2143, the following rationales further support this conclusion of obviousness in view of Safford and Kennedy.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

A POSITA would have combined the teachings of Safford and Kennedy according to known methods to yield predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 97. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed each element claimed, (2) a POSITA would have combined the prior-art elements by known methods, (3) each prior-art element performs the same function in isolation and in the combination, and (4) a POSITA would have recognized the results of the proposed combination as being predictable. As shown below, the proposed combination of Safford and Kennedy satisfies all the prongs of this rationale.

First, the combination of Safford and Kennedy disclosed each element recited in the applicable claims. *See, e.g., Appendix B* (Ex. BB).

Second, a POSITA would have combined the prior-art elements by known methods. *Wolfe Decl.* (Ex. 1003) ¶ 97. For example, Safford described a system that replaces a failed processor with a hot spare processor. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Kennedy described failing over boot processors to spare processors. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, and 7:5-13. Specifically, Kennedy disclosed a known method for transferring the role of boot processor to a spare processor. *See id.* and *Wolfe Decl.* (Ex. 1003) ¶ XX. A POSITA would have understood Safford as disclosing a boot processor and failing over the same to a hot spare processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 97. However, to the extent that Safford is interpreted to not explicitly disclose a "boot processor", a POSITA would have combined Safford's system with Kennedy's boot processor by the method disclosed in Kennedy to transfer the role of boot processor to Safford's hot spare processor. *Wolfe Decl.* (Ex. 1003) ¶ 97.

Third, each prior-art element performs the same function in isolation and in the combination. *Wolfe Decl.* (Ex. 1003) ¶ 98. For example, in isolation Safford's system replaces a failed processor with a hot spare processor, and Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure. *Id.* And the combined Safford-Kennedy system would transfer the role of a failed boot processor to a hot spare processor, performing the same failover function that each element performed in isolation. Accordingly, the functions performed by the prior-art elements do not change from the individual Safford and Kennedy references to the proposed combination.

Fourth, a POSITA would have recognized the results of the proposed combination as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 99. For example, a POSITA would have recognized that, by **replacing Safford's failed processor with Kennedy's failed boot processor, Safford's** system would have been able to predictably provide boot processor failover, as opposed to just general processor failover. *Id.*

Simple Substitution of One Known Element for Another to Obtain Predictable Results

A POSITA **would have performed a simple substitution of Safford's processor with Kennedy's boot processor to obtain predictable results.** *Wolfe Decl.* (Ex. 1003) ¶ 100. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a device that differed from the claimed device by the substitution of some components with other components, (2) the substituted components and their functions were known in the art, (3) a POSITA could have substituted one known element for another, and (4) the results of the substitution would have been predictable. As shown below, the proposed combination of Safford and Kennedy satisfies all the prongs of this rationale.

First, to the extent that Safford is interpreted as differing from the claim limitations of the '781 Patent, Safford disclosed a device that differed from the claimed device by the substitution of some components with other components. For example, Safford disclosed a device that replaces a failed processor with a hot spare processor in response to a loss of lockstep. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Similarly, the claimed device transfers the role of boot processor to a spare processor in response to a loss of lockstep. ***See*** *'781 Patent* (Ex. 1001), claim 1. Accordingly, to the extent that Safford is not interpreted as explicitly disclosing **a "boot processor," the substitution of Safford's failed processor with a failed boot processor would** obviate the difference between Safford and the claimed device. *Wolfe Decl.* (Ex. 1003) ¶ 101.

Second, the substituted components and their functions were known in the art. *Wolfe Decl.* (Ex. 1003) ¶ 102. For example, the substituted boot processor from Kennedy and its functions were known in the art. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 102**. As described in Kennedy, a "default boot CPU is designated to boot the system."** *Kennedy* (Ex. 1005) at 2:18-19. Accordingly, the prior art shows that the substituted boot processor was known to boot the system.

Third, a POSITA would have substituted one known element for another. *Wolfe Decl.* (Ex. 1003) ¶ 103. For example, it was well within the skill of a POSITA to merely substitute one

type of processor for another (*e.g.,* **Safford's failed processor for Kennedy's failed boot** processor). *Id.*

Fourth, a POSITA would have recognized the results of the substitution as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 104. For example, a POSITA would have recognized that, **by substituting Safford's failed processor for Kennedy's failed boot processor, Safford's device** would have been able to predictably provide boot processor failover. *Id.*

Use of Known Technique to Improve Similar Devices in the Same Way

A known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA. *Wolfe Decl.* (Ex. 1003) ¶ 105. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a comparable device (method, or product that is not the same as the base device) that has been improved in the same way as the claimed invention, (3) a POSITA could have applied the known improvement technique in the same way to the base device, and (4) the results would have been predictable to a POSITA. As shown below, the proposed combination of Safford and Kennedy satisfies all the prongs of this rationale.

First, Safford disclosed a base device upon which the claimed invention can be seen as an improvement. *Wolfe Decl.* (Ex. 1003) ¶ 106. For example, Safford disclosed a device that replaces a failed processor with a hot spare processor in response to a loss of lockstep. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Similarly, the claimed invention transfers the role of boot processor to a spare processor in response to a loss of lockstep. ***See** '781 Patent* (Ex. 1001), claim 1. Accordingly, to the extent that Safford does not explicitly disclose applying its techniques **specifically to a "boot processor," an** improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed general processor, with a hot spare. *Wolfe Decl.* (Ex. 1003) ¶ 106.

Second, the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 107. For example, Kennedy disclosed a comparable device, method, or product that transfers control from a failed boot processor to an alternate processor. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, and 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 107. **Because Kennedy's device transfers control from the failed boot processor to the alternate processor, Kennedy**'s device has been improved relative to

Safford's device in the same way as the claimed invention. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 107.

Third, a POSITA would have applied the known improvement technique in the same way to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 108. For example, a POSITA would have improved Safford's device by applying Kennedy's technique to fail over a boot processor, as opposed to failing over a general processor. *Id.*

Fourth, a POSITA would have recognized the results of the applied improvement technique as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 109. For example, a POSITA would have recognized that, by applying Kennedy's known technique, Safford's device would have been able to predictably provide boot processor failover. *Id.*

Applying a Known Technique to a Known Device Ready for Improvement

A known technique that is recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 110. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a known technique that is applicable to the base device, (3) a POSITA would have recognized that applying the known technique would have yielded predictable results, and (4) the predictable results would have improved the base device. As shown below, the proposed combination of Safford and Kennedy satisfies all the prongs of this rationale.

First, Safford disclosed a base device upon which the claimed invention can be seen as an improvement (if Safford is interpreted to somehow limit its disclosure to processors other than "boot processors"). *Wolfe Decl.* (Ex. 1003) ¶ 111. For example, Safford disclosed a device that replaces a failed processor with a hot spare processor in response to a loss of lockstep. *See, e.g., Safford* (Ex. 1004) at 3:27-30; 4:37-54. Similarly, the claimed invention transfers the role of boot processor to a spare processor in response to a loss of lockstep. *'781 Patent* (Ex. 1001), claim 1. Accordingly, an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed general processor, with a hot spare. *Wolfe Decl.* (Ex. 1003) ¶ 111.

Second, the prior art also disclosed a known technique that is applicable to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 112. For example, Kennedy disclosed a known technique for

transferring control of a device from a failed boot processor to an alternate processor. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ XX. Because Kennedy's known technique involves transferring control of the device from the failed boot processor to the alternate processor and Kennedy's device is very similar to and includes the same components and functionality as Safford's device, Kennedy's known technique is applicable to Safford's device. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 112.

Third, a POSITA would have recognized that applying the known technique to the base device would have yielded predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 113. For example, a POSITA would have recognized that applying Kennedy's known technique to Safford's device would have yielded predictable results. *Id.* More specifically, a POSITA would have recognized that, by applying Kennedy's known technique, Safford's device would have been able to predictably provide boot processor failover. *Id.* Moreover, because Safford already disclosed applying the same failover technique to processors in general, a POSITA would have reasonably expected the application of Kennedy's technique on Safford's device to be successful.

Fourth, the predictable results would have improved the base device. *Wolfe Decl.* (Ex. 1003) ¶ 114. For example, the predictable results would have improved the reliability of Safford's device (*e.g.,* in the event that the boot processor fails). *Id.*

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to Combine Prior-Art Teachings to Arrive at the Claimed Invention

One or more teachings, suggestions, or motivations in Nguyen and Fox would have led a POSITA to combine the teachings of Safford and Kennedy to arrive at the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 115. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) there was some teaching, suggestion, or motivation either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the references' teachings and (2) there was a reasonable expectation of success for the combination. As shown below, the proposed combination of Safford and Kennedy satisfies all the prongs of this rationale.

First, Kennedy emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Safford and Kennedy in this way. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 116. For example, Kennedy taught that its technique for failing over boot processors

"provides fault tolerance and high availability for the system." *See, e.g., Kennedy* (Ex. 1005) at 2:21-22; *see also Wolfe Decl.* (Ex. 1003) ¶ XX. Further, Kennedy taught that, "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. Kennedy also taught that, "if one processor fails, provision for disabling the non-functional processor and allowing another alternative processor to boot the system enhances the availability of the system." *Id.* at 10:4-7. Any or all of these teachings, suggestions, and/or motivations from Kennedy would have led a POSITA to combine the teachings of Safford and Kennedy in this way. *Wolfe Decl.* (Ex. 1003) ¶ 116.

Additionally or alternatively, a POSITA would have had the knowledge to combine the teachings of Safford and Kennedy so as to enable Safford's system to dynamically replace a failed boot processor, as opposed to dynamically replacing a failed general processor. *Wolfe Decl.* (Ex. 1003) ¶ 117. Even disregarding the express teachings, suggestions, and/or motivations cited from Kennedy, a POSITA's knowledge alone constitutes a sufficient motivation to combine Safford and Kennedy in this way. *See, e.g.,* M.P.E.P. § 2143.I(G); *see also DyStar Textilfarben GmbH & Co. Deutschland KG v. C.H. Patrick Co.*, 464 F.3d 1356, 1360 (Fed. Cir. 2006). "Because the desire to enhance commercial opportunities by improving a product or process is universal—and even common-sensical—we have held that there exists in these situations a motivation to combine prior art references even absent any hint of suggestion in the references themselves." *Id.* "In such situations, the proper question is whether the ordinary artisan possesses knowledge and skills rendering him *capable* of combining the prior art references." *Id.* A POSITA would have possessed the knowledge and skills to combine the teachings of Safford and Kennedy in this way. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶¶ 117, 118.

Second, there was a reasonable expectation of success for the proposed combination of Safford and Kennedy. *Wolfe Decl.* (Ex. 1003) ¶ 118. For example, Safford described replacing a failed processor with a hot spare processor. *See, e.g., Safford* (Ex. 1004) at 3:27-30, 4:37-54. Even if Safford did not explicitly identify the failed processor as having been a boot processor, Safford as modified by Kennedy would have rendered obvious the application of Safford's methods to a boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Moreover, Safford's general disclosure of failed processors would have been understood to include the boot processor of Safford's system even though that detail is not explicitly stated in Safford. *See, e.g., Wolfe Decl.*

(Ex. 1003) ¶ 118. Safford does not discourage or otherwise limit its disclosure to processors other than "boot processors," and it was well within the skill of a POSITA to modify Safford's system by Kennedy in the manner proposed. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 118. Accordingly, the proposed combination of Safford and Kennedy, in which Safford's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed. *Id.*

G. Ground 3: Kennedy in view of Nguyen and Fox Presents Substantial New Questions of Patentability for Claims 1-30

1. Overview of Kennedy

An overview of Kennedy appears above. *See supra* § I.E.2. That overview is equally applicable to Ground 3 and will not be repeated here for the sake of brevity.

2. Overview of Nguyen

Nguyen generally described a processor that includes first and second execution cores that operate in functional redundancy checking (FRC) mode. Ex. 1008, Abstract. As explained in Nguyen, a "single processor enabled for FRC may include replicated instruction execution cores on which the same instruction code is run." *Id.* ¶ [0006]. "In FRC mode, one of execution cores 120 may be designated as the master. The master refers to the execution core that is responsible for updating the resources shared by execution cores 120." *Id.* ¶ [0027]. "The other execution core 120 may be designated as the slave. The slave is responsible for generating results from the same code sequence to be checked against those of the master." *Id.*

Nguyen's processor implements multiple "[e]xecution cores 120 [that] operate in lock step during normal FRC mode." *Id.* ¶ [0024]. The configuration of Nguyen's processor, whose execution cores 120 operate in lockstep, is illustrated in FIG. 1 and is reproduced below for convenience.

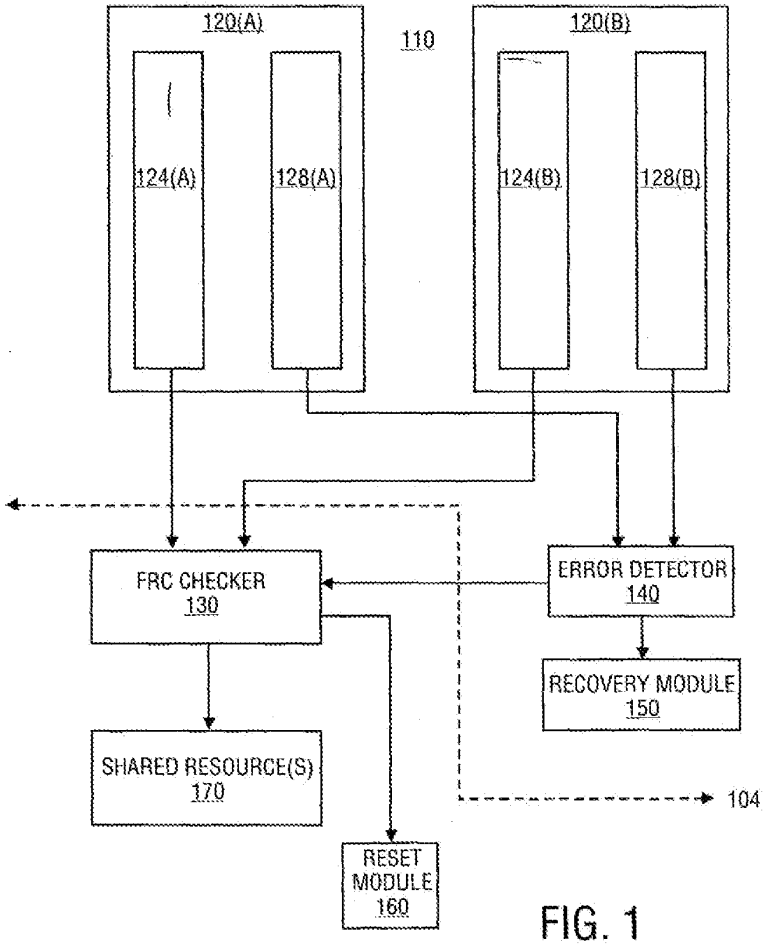


FIG. 1

Nguyen (Ex. 1008), FIG. 1

Also, Nguyen's processor includes data storage for storing information that assigns different roles to the execution cores. For example, Nguyen's processor maintains "a bit in a status register of each execution core 120…to indicate its status as master or slave. The bit may be set, for example, when the system is booted or reset." *Id.* ¶ [0047]. "[T]he master/slave status of an execution may be changed dynamically to allow recovery for an error in either core." *Id.*

Further, Nguyen's processor includes logic for changing the roles of the execution cores in response to a loss of lockstep. For example, "[i]f the error originates in the master core rather than the slave core, recovery may be handled by changing the master/slave designations of the execution cores." *Id.* ¶ [0054]. Nguyen notes that such errors result in a loss of lockstep.

Specifically, "for FRC-enabled processors, such errors are likely to be manifested as FRC errors, since they take the execution cores out of lock-step." *Id.* ¶ [0008].

Nguyen is analogous art to the '781 Patent. For example, Nguyen is from the same field of endeavor (*e.g.*, providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (*e.g.*, silent data corruption). *Wolfe Decl.* (Ex. 1003) ¶ 121. For example, as described in the '781 Patent, "Silent Data Corruption ('SDC') is a difficult problem in the computing industry." Ex. 1001 at 3:16-18. Similarly, as described in Nguyen, "[e]rrors that are not detected at the FRC boundary can lead to SDC." Ex. 1007 ¶ [0006]. Moreover, the '781 Patent elaborates on a common source of the SDC problem by noting that "a charged particle (*e.g.*, alpha particle or cosmic ray) strikes a region of a computing system and causes some bits to change value (*e.g.*, from a 0 to a 1 or from a 1 to a 0)." Ex. 1001 at 1:56-64. Like the '781 Patent, Nguyen also elaborated on the same source of the SDC problem by noting that soft errors, which pose a major challenge to both high reliability and high availability, "result from collisions between high-energy particles, e.g. alpha particles, and charge storing nodes." Ex. 1007 ¶ [0004].

3. Overview of Fox

Fox generally described a multiprocessor system capable of dynamically replacing a failed processor with a hot-spare processor that remains idle and/or off until the failed processor is detected. Ex. 1006, Abstract. As explained in Fox, a "multiprocessor computer system (MP) [is provided] with a spare processor that is initially held-off and is later brought into the group of executing processors whenever one of the main processors is detected as failing." *Id.* at 4:11-16. Fox's MP "provides a substantially seamless transfer of executing processes from the failing processor to the replacement processor, without halting the execution of the operating system (OS) and/or applications of the overall MP." *Id.* at 4:16-20. "That is, a failing processor is able to autonomically fail-over to a hot-spare processor without affecting the OS or executing applications." *Id.* at 4:20-23.

Fox's MP includes multiple main processors that operate in lockstep by processing the same application code and OS operations. For example, Fox's MP begins by setting up the BIOS and "installing the main processors, P0 and P1". *Id.* at col. 7, lines 53-55. "The primary processors commence processing application code and OS operations on the MP". *Id.* at 7:60-62. "[T]he application code and OS continue processing on the processors that are now functional (i.e., the

other main processor and the spare processor)". *Id.* at 8:15-20. "Notably, for this seamless transfer to be possible, all processors in the system must be identical **or process instructions in an identical manner.**" *Id.* at 8:20-22. The configuration of Fox's MP, whose processors operate in lockstep, is illustrated in FIG. 1 and is reproduced below for convenience.

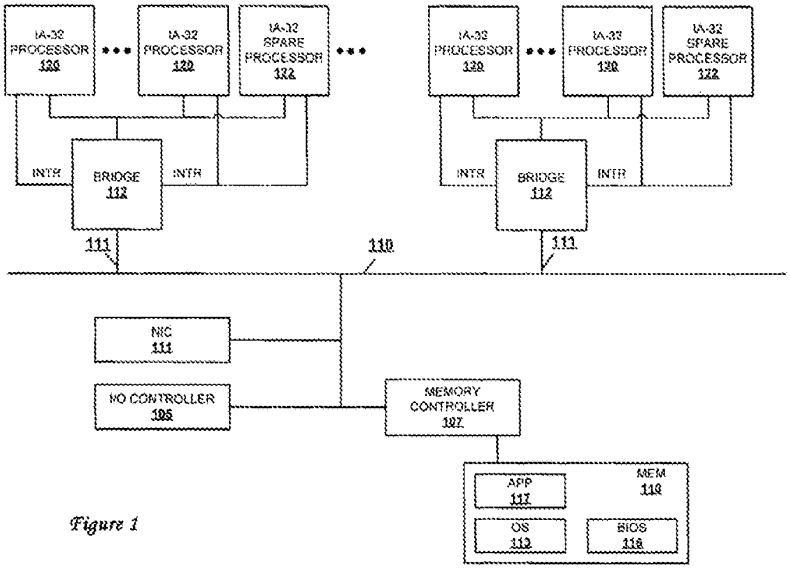


Figure 1

EX[1006], FIG. 1

Also, Fox's "system monitors the processors for a detection of failure." *Id.* at 7:62-63. "If failure is detected…the receipt of a processor fault SMI by the OS immediately activates a switch over response to replace the failing processor with the spare processor." *Id.* at 8:3-6. "The spare processor is brought out of the held-off state." *Id.* at 8: 6-7. Fox's MP then updates the configurations of the OS and bus controllers to "reflect a processor change from the failing processor to the spare processor." *Id.* at 8:11-15. To change the role of a processor from hot spare to primary in this way, Fox's system follows the process illustrated in FIG. 3, which is reproduced below for convenience.

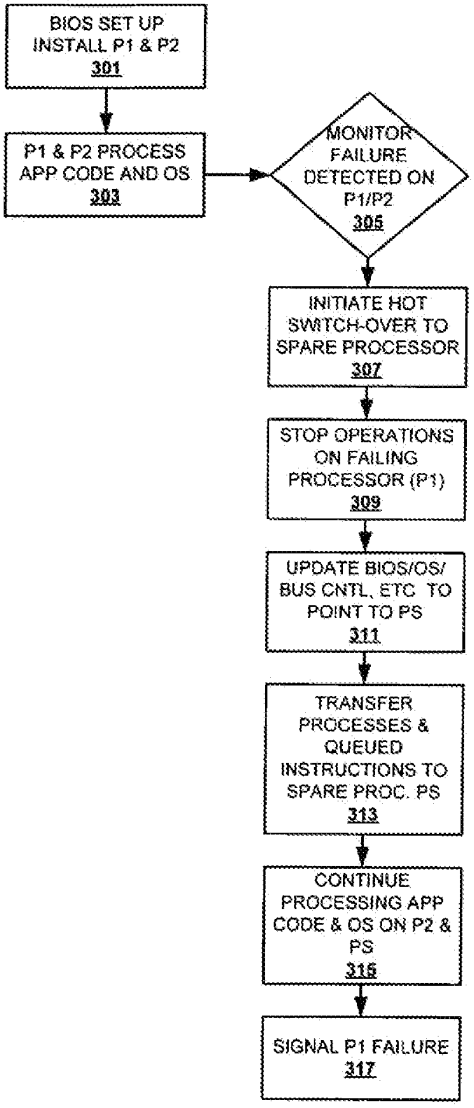


Figure 3

Fox (Ex. 1006), Fig. 3

Fox is analogous art to the '781 Patent because Fox is from the same field of endeavor (*e.g.*, providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (*e.g.*, suspension of processing during the replacement of a failed processor). *Wolfe Decl.* (Ex. 1003) ¶ 123. More specifically, the

'781 Patent identifies the problem of "requiring that the system be interrupted (*e.g.*, shutdown and restarted, or crashed)" when changing the role of a boot processor due to failure. *'781 Patent* (Ex. 1001) at 18:32-36. Similarly, Fox notes the problem of "the system having to suspend processing" when providing a replacement processor due to failure. *Fox* (Ex. 1006) at 2:60-62.

3. Obviousness of the '781 Patent in view of Kennedy in view of Nguyen and Fox

Claims 1-30 of the '781 Patent would have been obvious to a POSITA in view of Kennedy, Nguyen, and Fox. As will be described below, this conclusion of obviousness is supported by one or more motivations to combine Kennedy, Nguyen, and Fox as proposed, a reasonable expectation of success for the proposed combination of Kennedy, Nguyen, and Fox and various *KSR* rationales detailed throughout M.P.E.P. § 2143.

(I) Overview of Motivation to Combine Kennedy, Nguyen, and Fox as Proposed

As mentioned above, Kennedy described failing over a boot processor to an alternate boot processor in response to a failure of the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy states that "one default boot CPU is designated to boot the system, and if this CPU fails, control transfers automatically to an alternate processor." *Id.* at 2:17-21. Even if Kennedy did not explicitly describe failing over the boot processor in response to a loss of lockstep for the boot processor, Kennedy as modified by Nguyen would have rendered such a technique obvious. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]. For example, Nguyen described failing over a master execution core to a slave execution core in response to a functional redundancy checking (FRC) error (*e.g.*, loss of lockstep) detected in the master execution core. *See id.* In view of Nguyen's teachings regarding failing over from one core to another in response to a loss of lockstep, a POSITA would have been motivated to modify Kennedy to fail over from the boot processor to the alternate processor in response to a loss of lockstep in the boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 125. That Kennedy uses the term "processor" and Nguyen uses the term "core" does not change this since a POSITA would have recognized that, to the extent any difference exists, "processors" and "cores" are corresponding components whose differences do not implicate the combination. *See Wolfe Decl.* (Ex. 1003) ¶ 126. The '781 Patent itself even admits as much as it describes its processors as being cores. *See,*

e.g., '781 Patent (Ex. 1001) at 6:13-19. For example, the '781 Patent notes that "[s]ystem 10 includes OS 11, as well as master processor 12A and slave processor 12B (collectively referred to as a lockstep processor pair 12)." *See id* at 6:13-15. The '781 Patent further notes that "the lockstep processor pair 12 may be implemented on a single silicon chip, which is referred to as a 'dual core processor' in which master processor 12A is a first core and slave processor 12B is a second core." *See id.* at 6:15-19.

Even if neither Kennedy nor Nguyen explicitly described the alternate processor as being a hot spare processor that is held in reserve, Kennedy as modified by Nguyen and Fox would have rendered obvious such a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract. For example, Fox described failing over a processor to a hot spare processor. *See id.* Accordingly, Kennedy as modified by Nguyen and Fox taught replacing a failed boot processor with a hot spare processor in response to an FRC error detected in the boot processor.

A POSITA would have been motivated to combine the teachings of Kennedy, Nguyen, and Fox to improve the reliability of Kennedy's system (*e.g.,* by failing over the boot processor to a hot spare processor in response to a loss of lockstep). *Wolfe Decl.* (Ex. 1003) ¶ 127. For example, a POSITA would have been motivated to modify Kennedy's system, which implements boot processor failover, by Nguyen's teachings about detecting a loss of lockstep. *See id.* A POSITA would have been motivated to do so because Nguyen's capabilities of initiating failover in response to a loss of lockstep would provide Kennedy's system with additional fault tolerance and high availability (*e.g.,* in the event that the boot processor fails during runtime). *Wolfe Decl.* (Ex. 1003) ¶ 129. Indeed, a POSITA would have recognized the obvious benefits and advantages of enabling Kennedy's system to dynamically fail over boot processors in response to a loss of lockstep, especially in view of Kennedy's express objective of "provid[ing] fault tolerance and high availability." *Kennedy* (Ex. 1005) at 2:21-22; *see also Wolfe Decl.* (Ex. 1003) ¶ 130.

Moreover, a POSITA would have been motivated to further modify Kennedy's system by Fox's teachings about a hot spare processor that is held idly in reserve. *Wolfe Decl.* (Ex. 1003) ¶ 131. A POSITA would have been motivated to do so because Fox's hot spare processor would enable Kennedy's system to seamlessly transfer executing processes from the failing master execution core to a hot spare processor without affecting the operating system (OS), the applications running on Nguyen's system, or the high availability of Kennedy's system. *Wolfe Decl.* (Ex. 1003) ¶ 132. Indeed, a POSITA would have recognized the obvious benefits and

advantages of providing Kennedy's system with a hot spare processor, especially in view of Fox's express suggestion that "a failing processor is able to automatically fail-over to a hot-spare processor without affecting the OS or executing applications." *Fox* (Ex. 1006) at 4:20-23; *see also Wolfe Decl.* (Ex. 1003) ¶ 133. Consistent with the Supreme Court's holding in *KSR*, the rationales provided below further support this conclusion of obviousness.

(II) Reasonable Expectation of Success

A POSITA would have had a reasonable expectation of success in combining the teachings of Kennedy, Nguyen, and Fox. *Wolfe Decl.* (Ex. 1003) ¶ 134. For example, Kennedy described failing over a boot processor to an alternate boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Even if Kennedy did not explicitly describe detecting a loss of lockstep, Kennedy as modified by Nguyen would have rendered such a technique obvious. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]. For example, Nguyen described detecting a functional redundancy checking (FRC) error (*e.g.,* loss of lockstep) in a master execution core. *See id.* As mentioned above, the '781 Patent describes such cores as each being an example of a processor. *See, e.g., '781 Patent* (Ex. 1001) at 6:13-19. Because Kennedy's teachings are directed to failing over the boot processor and Nguyen describes a loss of lockstep as triggering such processor failover, the proposed combination of Kennedy and Nguyen would have been expected to succeed. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 135. It was well within the skill of a POSITA to modify Kennedy's system by Nguyen in the manner proposed, and Kennedy did not teach away from or discourage applying such a modification. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 136. That is, Kennedy does not limit or otherwise discourage application of its techniques from being triggered by a loss of lockstep.

Moreover, even if neither Kennedy nor Nguyen explicitly identified the alternate processor as being a hot spare processor that is held in reserve, Kennedy as modified by Nguyen and Fox would have rendered obvious such a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract. It was well within the skill of a POSITA to further modify Kennedy's system by Fox in the manner proposed, and Kennedy did not teach away from or discourage applying such a further modification. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 138. That is, Kennedy does not limit or otherwise discourage application of its techniques to a "hot spare processor." Accordingly, the proposed combination of Kennedy, Nguyen, and Fox—in which Kennedy's system is modified by Nguyen and Fox to achieve dynamically replacing a failed boot processor with a hot spare

processor in response to a loss of lockstep—would have been reasonably expected to succeed. *Id.*

(III) *KSR* Rationales

Consistent with the Supreme Court's holding in *KSR*, the following rationales further support this conclusion of obviousness in view of Kennedy, Nguyen, and Fox.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

A POSITA would have combined the teachings of Kennedy, Nguyen, and Fox according to known methods to yield predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 140. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed each element claimed, (2) a POSITA could have combined the prior-art elements by known methods, (3) each prior-art element performs the same function in isolation and in the combination, and (4) a POSITA would have recognized the results of the proposed combination as being predictable. As shown below, the proposed combination of Kennedy, Nguyen, and Fox satisfies all the prongs of this rationale.

First, the combination of Kennedy, Nguyen, and Fox disclosed each element recited in the applicable claims. *See, e.g., Appendix C* (Ex. CC).

Second, a POSITA would have combined the prior-art elements by known methods. *Wolfe Decl.* (Ex. 1003) ¶ 142. For example, Kennedy described failing over boot processors to spare processors. *See, e.g., Kennedy* (Ex. 1005) at 1:16-19, 2:15-22, 7:5-13; *see also Wolfe Decl.* (Ex. 1003) ¶ 142. Nguyen described detecting a loss of lockstep in a master execution core. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]; *see also Wolfe Decl.* (Ex. 1003) ¶ 142. A POSITA would have combined Kennedy's system with Nguyen's known methods to transfer the role of boot processor to the alternate processor in response to a loss of lockstep for the boot processor. *Wolfe Decl.* (Ex. 1003) ¶ 142. Moreover, Fox described a multiprocessor system that replaces a failed processor with a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract. Specifically, Fox disclosed a known method for failing over to a hot spare processor. *See id.*; *see also Wolfe Decl.* (Ex. 1003) ¶ 142. A POSITA would have further combined Kennedy's modified system with Fox's hot spare processor by the methods disclosed in Fox to perform the boot processor failover without affecting Kennedy's OS or corresponding applications. *Wolfe Decl.* (Ex. 1003) ¶ 142.

Third, each prior-art element performs the same function in isolation and in the combination. *Wolfe Decl.* (Ex. 1003) ¶ 143. For example, in isolation and in the combination, Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure, Nguyen's system detects a loss of lockstep, and Fox's system replaces a failed processor with a hot spare processor. *Id.* Accordingly, the functions performed by the prior-art elements do not change from the individual Kennedy, Nguyen, and Fox references to the proposed combination.

Fourth, a POSITA would have recognized the results of the proposed combination as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 144. For example, a POSITA would have recognized that, by failing over Kennedy's boot processor to Fox's hot spare processor in response to Nguyen's detection of a loss of lockstep, Kennedy's modified system would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep. *Wolfe Decl.* (Ex. 1003) ¶ 144.

Simple Substitution of One Known Element for Another to Obtain Predictable Results

A POSITA would have performed a simple substitution of Kennedy's detection of a boot processor failure with Nguyen's detection of a loss of lockstep and a simple substitution of Kennedy's alternate processor with Fox's hot spare processor to obtain predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 145. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a device that differed from the claimed device by the substitution of some components with other components, (2) the substituted components and their functions were known in the art, (3) a POSITA could have substituted one known element for another, and (4) the results of the substitution would have been predictable. As shown below, the proposed combination of Kennedy, Nguyen, and Fox satisfies all the prongs of this rationale.

First, Kennedy disclosed a device that replaces the boot processor with an alternate processor in response to a failure of the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Similarly, the claimed device transfers the role of boot processor to a hot spare processor in response to a detected error. *'781 Patent* (Ex. 1001), claim 1. Accordingly, the substitution of Kennedy's detection of a boot processor failure with a detection of a loss of lockstep and the substitution of Kennedy's alternate processor with a hot spare processor would obviate the difference between Kennedy and the claimed device. *Wolfe Decl.* (Ex. 1003) ¶ 146.

Second, the substituted components and their functions were known in the art. *Wolfe*

Decl. (Ex. 1003) ¶ 147. For example, the substituted detection of a loss of lockstep from Nguyen, the substituted hot spare processor from Fox, and their corresponding functions were known in the art. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]; *see also Fox* (Ex. 1006), Abstract; *see also Wolfe Decl.* (Ex. 1003) ¶ 147. As described in Nguyen, "[e]xecution cores 120 operate in lock step," and "a discrepancy is detected." *Nguyen* (Ex. 1008), pars. [0024], [0008]. As described in Fox, one processor "is initially provided as a reserve (or hot-spare) processor that remains in idle, off, or low-power mode." *Fox* (Ex. 1006) at 3:19-21. Accordingly, the prior art shows that the substituted detection of loss of lockstep, the substituted hot spare processor, and their functions were known.

Third, a POSITA would have substituted one known element for another. *Wolfe Decl.* (Ex. 1003) ¶ 148. For example, it was well within the skill of a POSITA to merely substitute one type of trigger for boot processor failover for another (*e.g.,* Kennedy's boot processor failure for Nguyen's loss of lockstep) and to merely substitute one type of spare processor for another (*e.g.,* Kennedy's alternate processor for Fox's hot spare processor). *Id.*

Fourth, a POSITA would have recognized the results of the substitution as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 149. For example, a POSITA would have recognized that, by substituting Kennedy's boot processor failure for Nguyen's loss of lockstep and Kennedy's alternate processor for Fox's hot spare processor, Kennedy's modified device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep to ensure that a boot processor is always selected in case of a reboot and to decrease delays in the failover process. *Wolfe Decl.* (Ex. 1003) ¶ 149.

Use of Known Technique to Improve Similar Devices in the Same Way

A known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA. *Wolfe Decl.* (Ex. 1003) ¶ 150. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention, (3) a POSITA could have applied the known improvement technique in the same way to the base device, and (4) the results would have been predictable to a POSITA. As shown below, the proposed combination of Kennedy, Nguyen, and Fox satisfies all the prongs of this rationale.

First, Kennedy disclosed a base device upon which the claimed invention can be seen as an improvement. *Wolfe Decl.* (Ex. 1003) ¶ 151. For example, Kennedy disclosed a device that replaces the boot processor with an alternate processor in response to a failure of the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Similarly, the claimed invention transfers the role of boot processor to a hot spare processor in response to a detected error. *'781 Patent* (Ex. 1001), claim 1. Accordingly, an improvement appears to be that the claimed invention fails over from the boot processor to the hot spare processor. *Wolfe Decl.* (Ex. 1003) ¶ 151.

Second, the prior art also disclosed comparable devices that have been improved in the same ways as the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 152. For example, Nguyen disclosed a comparable device that detects a loss of lockstep. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]; *see also Wolfe Decl.* (Ex. 1003) ¶ 152. Fox disclosed a comparable device that replaces a failed processor with a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract; *see also Wolfe Decl.* (Ex. 1003) ¶ 152. Because Nguyen's device detects a loss of lockstep and Fox's device replaces the failed processor with a hot spare processor, Nguyen's and Fox's devices have been improved relative to Kennedy's device in the same ways as the claimed invention. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 152.

Third, a POSITA would have applied the known improvement techniques in the same ways to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 153. For example, a POSITA would have improved Kennedy's device by applying Nguyen's technique to perform the failover in response to a loss of lockstep and by applying Fox's technique to fail over to a hot spare processor. *Id.*

Fourth, a POSITA would have recognized the results of the applied improvement techniques as being predictable. *Wolfe Decl.* (Ex. 1003) ¶ 154. For example, a POSITA would have recognized that, by applying Nguyen's and Fox's known techniques, Kennedy's device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep. *Id.*

Applying a Known Technique to a Known Device Ready for Improvement

A known technique that is recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 155. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) the prior art disclosed a base device upon which the claimed invention can be seen as an improvement, (2) the prior art also disclosed a known technique that is applicable to

the base device, (3) a POSITA would have recognized that applying the known technique would have yielded predictable results, and (4) the predictable results would have improved the base device. As shown below, the proposed combination of Kennedy, Nguyen, and Fox satisfies all the prongs of this rationale.

First, Kennedy disclosed a base device upon which the claimed invention can be seen as an improvement. *Wolfe Decl.* (Ex. 1003) ¶ 156. For example, Kennedy disclosed a device that replaces the boot processor to an alternate processor in response to a failure of the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Similarly, the claimed invention transfers the role of boot processor to a hot spare processor in response to a loss of lockstep. *'781 Patent* (Ex. 1001), claim 1. Accordingly, an improvement appears to be that the claimed invention fails over from the boot processor to the hot spare processor in response to a loss of lockstep. *Wolfe Decl.* (Ex. 1003) ¶ 156.

Second, the prior art also disclosed known techniques that are applicable to the base device. *Wolfe Decl.* (Ex. 1003) ¶ 157. For example, Nguyen described a known technique for detecting a functional redundancy checking (FRC) error (*e.g.,* loss of lockstep) in a master execution core. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]; *see also Wolfe Decl.* (Ex. 1003) ¶ 157. Fox disclosed a known technique for replacing a failed processor with a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract; *see also Wolfe Decl.* (Ex. 1003) ¶ 157. Because Nguyen's known technique involves detecting a loss of lockstep and Fox's known technique involves replacing the failed processor with the hot spare processor, Nguyen's and Fox's known techniques would have been applicable to Kennedy's device, which is very similar to and includes the same components and functionality as Nguyen's and Fox's devices. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 157.

Third, a POSITA would have recognized that applying the known techniques to the base device would have yielded predictable results. *Wolfe Decl.* (Ex. 1003) ¶ 158. For example, a POSITA would have recognized that applying Nguyen's and Fox's known techniques to Kennedy's device would have yielded predictable results. *Id.* More specifically, a POSITA would have recognized that, by applying Nguyen's and Fox's known techniques, Kennedy's device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep. *Id.*

Fourth, the predictable results would have improved the base device. *Id.* For example, the

predictable results would have improved the reliability of Kennedy's device (*e.g.*, by failing over the boot processor to a hot spare processor in response to a loss of lockstep). *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 159. More specifically, Kennedy's device would become more reliable by failing over the boot processor to a hot spare processor such that the hot spare processor immediately assumes responsibility for booting the underlying system in case a reboot is needed. *Id.* By doing so, Kennedy's device as modified by Nguyen and Fox would avoid a situation in which the underlying system attempts to reboot without a designated, functional boot processor after a loss of lockstep. *Id.*

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to Combine Prior-Art Teachings to Arrive at the Claimed Invention

One or more teachings, suggestions, or motivations would have led a POSITA to combine the teachings of Kennedy, Nguyen, and Fox to arrive at the claimed invention. *Wolfe Decl.* (Ex. 1003) ¶ 160. Pursuant to M.P.E.P. § 2143, this rationale is demonstrated by showing that (1) there was some teaching, suggestion, or motivation either in the references themselves or in the knowledge generally available to one of ordinary skill in the art to combine the references' teachings and (2) there was a reasonable expectation of success for the combination. As shown below, the proposed combination of Kennedy, Nguyen, and Fox satisfies all the prongs of this rationale.

First, Nguyen emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kennedy and Nguyen. *See, e.g., Nguyen* (Ex. 1007) ¶ [0006], [0008], [0054]; *see also Wolfe Decl.* (Ex. 1003) ¶ 161. For example, Nguyen taught that "[e]rrors that are not detected at the FRC boundary can lead to [Silent Data Corruption] SDC." *Id.* ¶ [0006]. Further, Nguyen taught that, "for FRC-enabled processors, such errors are likely to be manifested as FRC errors, since they take the execution cores out of lock-step." *Id.* ¶ [0008]. Nguyen also taught that soft errors, which pose a major challenge to both high reliability and high availability, "result from collisions between high-energy particles, e.g. alpha particles, and charge storing nodes." *Id.* ¶ [0004]. Any or all of these teachings, suggestions, and/or motivations from Kennedy would have led a POSITA to combine the teachings of Kennedy and Nguyen. *Wolfe Decl.* (Ex. 1003) ¶ 161.

Similarly, Fox emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kennedy and Fox. *See,*

e.g., Fox (Ex. 1006) at 3:32-42, 4:16-25, 7:38-40; *see also Wolfe Decl.* (Ex. 1003) ¶ 161. For example, Fox taught that its technique for failing over to hot spares "provides a substantially seamless transfer of executing processes…without halting the execution of the operating system (OS) and/or applications on the overall MP." *See, e.g., Fox* (Ex. 1006) at 4:16-20. Further, Fox taught that "a failing processor is able to autonomically fail-over to a hot-spare processor without affecting the OS or executing applications." *See, e.g., Fox* (Ex. 1006) at 4:20-23. Any or all of these teachings, suggestions, and/or motivations from Fox would have led a POSITA to combine the teachings of Kennedy and Fox. *Wolfe Decl.* (Ex. 1003) ¶ 161.

Additionally or alternatively, a POSITA would have had the knowledge to combine the teachings of Kennedy, Nguyen, and Fox so as to enable Kennedy's system to fail over the boot processor to a hot spare processor in response to a loss of lockstep. *Wolfe Decl.* (Ex. 1003) ¶ 162. Even disregarding the express teachings, suggestions, and/or motivations cited from Nguyen and Fox, a POSITA's knowledge alone constitutes a sufficient motivation to combine Kennedy, Nguyen, and Fox in this way. *See, e.g.,* M.P.E.P. § 2143.I(G); *see also DyStar Textilfarben GmbH & Co. Deutschland KG v. C.H. Patrick Co.,* 464 F.3d 1356, 1360 (Fed. Cir. 2006). "Because the desire to enhance commercial opportunities by improving a product or process is universal—and even common-sensical—we have held that there exists in these situations a motivation to combine prior art references even absent any hint of suggestion in the references themselves." *Id.* "In such situations, the proper question is whether the ordinary artisan possesses knowledge and skills rendering him *capable* of combining the prior art references." *Id.* A POSITA would have possessed the knowledge and skills to combine the teachings of Kennedy, Nguyen, and Fox. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 162.

Second, there was a reasonable expectation of success for the proposed combination of Kennedy, Nguyen, and Fox. *Wolfe Decl.* (Ex. 1003) ¶ 163. For example, Kennedy described failing over a boot processor to an alternate boot processor in response to a failure of the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Even if Kennedy did not explicitly describe detecting a loss of lockstep, Kennedy as modified by Nguyen would have rendered such a technique obvious. *See, e.g., Nguyen* (Ex. 1008), pars. [0008], [0054]-[0055]; *see also Wolfe Decl.* (Ex. 1003) ¶ 163. Even if neither Kennedy nor Nguyen explicitly identifies the alternate processor as being a hot spare processor that is held in reserve, Kennedy as modified by Nguyen and Fox would have rendered obvious such a hot spare processor. *See, e.g., Fox* (Ex. 1006),

Abstract; s*ee also Wolfe Decl.* (Ex. 1003) ¶ 163. Kennedy does not discourage or otherwise limit its disclosure to replacing the boot processor with an active spare processor in response to a non-lockstep error, and it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in the manner proposed. *See, e.g., Wolfe Decl.* (Ex. 1003) ¶ 163. Accordingly, the proposed combination of Kennedy, Nguyen, and Fox—in which Kennedy's system is modified by Nguyen and Fox to achieve dynamically replacing a failed boot processor with a hot spare processor in response to a loss of lockstep—would have been reasonably expected to succeed. *Id.*

H. Ground 4: Safford Presents Substantial New Questions of Patentability for Claims 1 and 6

1. Overview of Safford

An overview of Safford appears above. *See supra* § I.F.1. That overview is equally applicable to Ground 4 and will not be repeated here for the sake of brevity.

2. Overview of Cepulis

Cepulis generally described "a multiprocessor computer system [that] handles the failure of one or more of its processors without totally disabling the system." *Cepulis* (Ex. 1013), Abstract. Cepulis described a boot process for a multiprocessor computer system that includes designating one of the multiprocessor computer system's physical processors as a boot processor or "first logical CPU." *Id.* Like the boot processor described in the '781 Patent, the boot processor in Cepulis "retains its designation until power is cycled" unless it fails at which time its "designation is transferred to another active CPU." *Id.*

Like Safford, Cepulis used the "logical CPU 0" designator to identify its boot processor as explained in greater detail in § II.B.1 below.

Cepulis is analogous art to the '781 Patent. For example, Cepulis is from the same field of endeavor (*e.g.*, providing recovery in a high-availability multi-processor system) and is reasonably pertinent to the problem faced by the inventors of the '781 Patent. As noted above, the '781 Patent is directed to recovering from processor errors in multiprocessor systems in a manner that avoids crashing an operating system. *'781 Patent* (Ex. 1001) at 3:16-18. Similarly, Cepulis is directed to "a multiprocessor computer system [that] handles the failure of one or more of its processors

without totally disabling the system." *Cepulis* (Ex. 1013), Abstract.

3. Anticipation by Safford

Safford anticipates each and every limitation of claims 1 and 6, and therefore presents a substantial new question of patentability. Although Safford was considered by the Examiner, the analysis was without the benefit of Cepulis' clarifying explanation of "logical CPU 0." A POSITA would have considered the disclosure of Safford in light of Cepulis important in determining whether claims 1 and 6 are patentable because Safford in light of Cepulis' explanation of the meaning of the "logical CPU 0" boot-processor designator was never expressly considered or discussed during prosecution of the '077 Application or during a prior post-grant proceeding challenging any claim of the '781 Patent. Exhibit A shows in greater detail how Safford anticipates each and every element of claims 1 and 6 when considered in light of Cepulis' explanation of "logical CPU 0."

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

A. Proposed Obviousness Rejections for the Challenged Claims

1. Ground 1 Kalyanasundharam in view of Kennedy

Kalyanasundharam in view of Kennedy would have rendered claims 1-6 and 9-15 obvious to a POSITA. For ease of reference, Requester has provided some proposed obviousness rejections for claims 1-6 and 9-15 below. Requester has also provided a detailed mapping of Kalyanasundharam and Kennedy to claims 1-6 and 9-15 in Appendix A.

Claim 1: ***[1.pre] A system comprising:*** (Kalyanasundharam disclosed a computer system. For example, Kalyanasundharam taught that "the **computer system** 10 includes nodes 12A-12B, memories 14A-14B, and peripheral devices 16A-16B." *Kalyanasundharam* (Ex. 1007) at 2:26-29);

[1.1] a plurality of processors (Kalyanasundharam disclosed multiple processors executing in lockstep. For example, Kalyanasundharam taught that "the **processor cores 18A-18B** may execute in lockstep (that is, may execute the same instructions on the same clock cycle)." *Kalyanasundharam* (Ex. 1007) at 4:15-18. "For example, some fault-tolerant systems may implement high reliability by including **multiple processors** and having them redundantly execute the same software code." *Id.* at 1:23-25. Kalyanasundharam's description of processor cores is

consistent with the '781 Patent's use of the term "processor." *See, e.g., '781 Patent* (Ex. 1001) at 6:13-19. For example, the '781 Patent notes that "[s]ystem 10 includes OS 11, as well as master processor 12A and slave processor 12B (collectively referred to as a lockstep processor pair 12)." *See id* at 6:13-15. The '781 Patent further notes that "the lockstep processor pair 12 may be implemented on a single silicon chip, which is referred to as a 'dual core processor' in which master processor 12A is a first core and slave processor 12B is a second core." *See id.* at 6:15-19. Accordingly, Kalyanasundharam disclosed multiple processors executing in lockstep.);

[1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors (Kalyanasundharam disclosed, or at least rendered obvious, this limitation by disclosing a PC bit in a configuration register that identifies the role of a processor as either a primary processor or an alternate processor. For example, Kalyanasundharam taught that "the communications from the primary processor core may be propagated and the communications from the auxiliary processor core(s) may be discarded…In this embodiment, the **PC bit** from the **configuration register 42** may **identify the primary processor core.**" *Kalyanasundharam* (Ex. 1007) at 8:34-44. "The **configuration register 42** includes a **primary core (PC) field** encoded to **identify which of the processor cores 18A-18N is designated as the primary core.**" *Id.* at 8:34-44. "In such embodiments, one processor core may be the **primary core** and two or more of the remaining processor cores may be **auxiliary processor cores.**" *Id.* at 4:66 – 5:2. Accordingly, Kalyanasundharam described assigning the roles of primary processor core and auxiliary processor core to certain processors via the PC bit in the configuration register. *See id* at 8:34-44, 4:66 – 5:2. In view of these citations and the knowledge that a boot processor was a common component in computer systems (*e.g.,* Kalyanasundharam's system), a POSITA would have understood Kalyanasundharam as disclosing a boot processor and a spare processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 169. Even if Kalyanasundharam did not explicitly identify which processor was the boot processor, Kalyanasundharam as modified by Kennedy would have stored information about the assigned roles of boot processor and spare processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy taught that, "if this 'default' boot processor fails[,]" then an **alternate processor** "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. In view of the above citations, Kalyanasundharam taught data storage that stores role assignments (*e.g.,* primary core and auxiliary core) of the processors, and

Kennedy taught that such assignments could be the roles of boot processor and alternate processor. *See Wolfe Decl.* (Ex. 1003) ¶ 169. A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by storing information that assigns the roles of boot processor and spare processor in case of a reboot. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. *Id.*);

[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the replacement of a primary processor with an auxiliary processor in response to a lockstep mismatch. For example, Kalyanasundharam taught that "the processor cores 18A-18B may **execute in lockstep (that is, may execute the same instructions on the same clock cycle).**" *Kalyanasundharam* (Ex. 1007) at 4:15-18. "If the masked comparison results indicate that the communication from the primary processor core **mismatches** with all auxiliary processor cores (that are enabled)…the control circuit 84 may **disable the primary processor core** and **select one of the auxiliary processor cores to be the primary processor core** (block 102)." *Id.* at 13:15-22. "As with the discussion above for **disabling the auxiliary processor core**, the control circuit 83 may simply **disable the primary processor core from comparison** by clearing the enable bit in the configuration register 42, or may **disable the primary processor core from executing at all**." *Id.* at 13:22-27. "The control circuit 84 may also **update the PC field of the configuration register 42** to reflect the **newly selected primary processor core.**" *Id.* at 13:27-29. Accordingly, Kalyanasundharam described transferring the role of primary processor to an auxiliary processor in response to a mismatch involving the primary processor. *See id.* In view of these citations and the knowledge that a boot processor was a common component in computer systems (*e.g.*, Kalyanasundharam's system), a POSITA would have understood Kalyanasundharam as disclosing the failover of a boot processor to a spare processor in response to a loss of lockstep. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 170. Even if Kalyanasundharam did not explicitly identify

the mismatched processor as having been the boot processor, Kalyanasundharam as modified by Kennedy would have performed boot processor failover in response to a lockstep mismatch. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, **Kennedy taught that, "if this 'default' boot processor** fails[,]" then an **alternate processor** "**should take over boot operations to enhance availability of the system.**" *Id.* In view of the above citations, Kalyanasundharam taught logic that causes its system to replace the primary processor with an auxiliary processor in response to a mismatch involving the primary processor, and Kennedy taught logic that causes its system to fail over the boot processor to an alternate processor. *See Wolfe Decl.* (Ex. 1003) ¶ 170. A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam**'s system** by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam**'s system by Kennedy in this manner, and thus** this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*);

[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing processors that have the same functionality as a hot spare. For example, Kalyanasundharam **taught that "a fixed or programmable number of the processor cores (less than the total number in the node) may participate** in redundant **execution.**" *Kalyanasundharam* (Ex. 1007) at 11:35-**38.** "The configuration register 42 may further include a set of **core enables** which indicate **which cores are participating** in redundant execution mode." *Id.* at 12:12-15. "**The control circuit 84 may…optionally disable a processor core or cores from comparison in the redundant execution mode.**" *Id.* at 12:29-32; "**Additionally, the core** enables in the configuration register 42 may be used to **selectively enable various pairs** for **redundant execution.**" *Id.* at 15:7-10; "**If the primary processor core is programmably selected, a different processor core may be selected as the primary processor core** and **the previous primary processor core may be disabled.**" *Id.* at 11:45-**48.** "For example, servers are implemented on **fault-tolerant systems** so that the servers may be available as close to 24 hours

a day as possible. Any **mission-critical computing functions** may benefit from being assigned to **fault-tolerant computer systems.**" *Id.* at 1:14-19. Even if Kalyanasundharam did not explicitly use the term "hot spare" or the phrase "held in reserve" in connection with its processors, Kalyanasundharam described its processors as having the same functionality as the hot spare recited in claim 1. For example, in view of the above citations, Kalyanasundharam taught selectively enabling processors for participation, selectively disabling processors from participation, and maintaining configuration register 42 that tracks which processors are participating and which processors are not participating. *See Wolfe Decl.* (Ex. 1003) ¶ 171. For at least these reasons, a POSITA would have understood Kalyanasundharam as having taught that the auxiliary processor that assumes the role of primary processor would have been disabled prior to the primary processor's mismatch. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught that the processor assigned the role of spare would be held in reserve until a loss of lockstep is detected, thereby rendering this subject matter obvious to a POSITA. *Id.*).

Claim 2: ***The system of claim 1 wherein said data storage is non-volatile data storage*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a PC bit in a configuration register that identifies the role of a processor as either a primary processor or an alternate processor. For example, Kalyanasundharam taught that "the communications from the primary processor core may be propagated and the communications from the auxiliary processor core(s) may be discarded…In this embodiment, **the PC bit from the configuration register 42 may identify the primary processor core.**" *Kalyanasundharam* (Ex. 1007) at 8:34-44. "The **configuration register 42** includes **a primary core (PC) field** encoded to **identify which of the processor cores 18A-18N is designated as the primary core.**" *Id.* at 8:34-44. Even if Kalyanasundharam did not explicitly identify the configuration register as being non-volatile data storage, Kalyanasundharam as modified by Kennedy would have included non-volatile memory that contains the configuration information. For example, "The CPU1 120 has a corresponding **non-volatile memory 123** which contains configuration information and CPU specific initialize and self test code (ISTC) for the CPU1 120." *Kennedy* (Ex. 1005), 5:15-18. From these disclosures, a POSITA would have understood that the data storage that stores the information assigning the roles of boot processor and spare processor included non-volatile data storage. *See Wolfe Decl.* (Ex. 1003) ¶ 172. It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of

Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 2 obvious to a POSITA. *Id.*).

Claim 3: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the selection of a processor type to boot the system. In other words, even if Kalyanasundharam did not explicitly disclose a user interface that receives input specifying the processor assigned the role of boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* It was well known in the art that the act of selecting processor roles in computer systems (*e.g.,* master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 173. Based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure about selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Kalyanasundharam by Kennedy and this common knowledge in the art to enable a user to select which processor is to boot the system via a user interface. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA. *Id.*).

Claim 4: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by implicitly disclosing the selection of a processor type that does not boot the system. In other words, even if Kalyanasundharam did not explicitly disclose a user interface that receives input specifying the processor assigned the role of spare processor, Kalyanasundharam as

modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* at 4:33-34. It was well known in the art that the act of selecting processor roles in computer systems (*e.g.*, master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 174. Based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure about selecting the processor to boot the system as including a spare processor selection (*e.g.*, one of the alternate processors not selected as the boot processor) made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Kalyanasundharam by Kennedy and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA. *Id.*).

Claim 5: ***The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing nodes that each included a primary processor and an auxiliary processor. For example, Kalyanasundharam taught that "[t]he **nodes 12A-12B are coupled.**" *Kalyanasundharam* (Ex. 1007) at 2:30. "**The node 12A comprises processor cores 18A-18B coupled to a node controller 20A.**" *Id.* at 2:33-34. "**The node 12B similarly comprises processor cores 18C-18D coupled to a node controller 20B.**" *Id.* at 2:36-37. The node controller 20A may be configured to **propagate communications from one of the processor cores 18A-18B (e.g., from the primary processor core)** and to **discard communications from the other processor core 18A-18B (e.g., from the auxiliary processor core).**" *Id.* at 4:36-40. "**In one embodiment, each of the nodes 12A-12B may be a single integrated circuit chip** comprising the circuitry shown therein in FIG. 1. That is, **each node 12A-12B may be a chip multiprocessor (CMP).**" *Id.* at 2:46-49. In view of these citations, Kalyanasundharam taught nodes that each included a primary processor and an auxiliary

processor, but only the primary processor propagates communications because the auxiliary processor's communications are discarded. Therefore, under the broadest reasonable interpretation consistent with the '781 Patent, a POSITA would have understood Kalyanasundharam's nodes as each constituting a processor module, Kalyanasundharam's primary processor core as constituting a master processor, and Kalyanasundharam's auxiliary processor core as constituting a slave processor. *See Wolfe Decl.* (Ex. 1003) ¶ 175. Accordingly, Kalyanasundharam as modified by Kennedy taught processor modules that each include a master processor and a slave processor. *Id.*).

Claim 6: ***The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing that the spare to the boot processor may be an enabled (*e.g.*, active) processor core by control circuit. For example, Kalyanasundharam taught that "[t]he control circuit 84 may use any selection mechanism. For example, the processor cores may be numbered (*e.g.*, 0 to N-1), and the control circuit 84 may **increment** the number of the presently selected primary processor core **to select a new primary processor core**." *Kalyanasundharam* (Ex. 1007) at 13:29-33. "If one or more processor cores are disabled in the configuration register 42, the control circuit 84 may increment until **an enabled processor core is selected.**" *Id.* at 13:33-36. In view of these citations, Kalyanasundharam taught that the spare to the primary processor may be an enabled (*e.g.*, active) processor core selected by control circuit 84. *See Wolfe Decl.* (Ex. 1003) ¶ 176. Although Kalyanasundharam did not explicitly use the term "boot processor" when describing the primary processor, Kalyanasundharam as modified by Kennedy would have performed boot processor failover. Based on the supplementation of Kennedy's disclosures, a POSITA would have understood Kalyanasundharam's primary processor to have constituted a boot processor for at least the same reasons provided above in connection with claim 1. *See id.* Accordingly, Kalyanasundharam as modified by Kennedy taught that the spare processor is an active spare before being transferred the role of boot processor. *Id.*).

Claim 9: ***[9.pre] A method comprising:*** (Kalyanasundharam disclosed a method. "A similar **method** is also contemplated." *Kalyanasundharam* (Ex. 1007), Abstract);

[9.1] assigning a role of boot processor to one of a plurality of processors in a system (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation

by disclosing the assignment of a primary processor role to a certain processor core. For example, Kalyanasundharam **taught that** "the communications from **the primary processor core** may be propagated and the communications from the auxiliary processor core(s) may be discarded…In this embodiment, **the PC bit** from the configuration register 42 may **identify the primary processor core.**" *Kalyanasundharam* (Ex. 1007) at 8:34-44. "**The configuration register 42** includes a **primary core (PC) field** encoded to **identify which of the processor cores 18A-18N** is designated as **the primary core.**" *Id.* at 8:34-44. In view of these citations, Kalyanasundharam described assigning the role of primary processor to one of processor cores 18A-18N. In view of these citations and the knowledge that a boot processor was a common component in computer systems (*e.g.*, Kalyanasundharam's system), a POSITA would have understood Kalyanasundharam as disclosing a boot processor and a spare processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 177. Even if Kalyanasundharam did not explicitly identify which processor was the boot processor, Kalyanasundharam as modified by Kennedy would have assigned the role of boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy taught that, "if this 'default' **boot processor** fails[,]" then an **alternate processor** "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. Therefore, Kalyanasundharam taught assigning the role of primary processor to a certain processor core, and Kennedy taught assigning the role of boot processor to one of its processors. *See Wolfe Decl.* (Ex. 1003) ¶ 177. A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by assigning the roles of boot processor and spare processor in case of a reboot. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught assigning the role of boot processor to one of its processor cores. *Id.*);

[9.2] assigning a role spare processor to another of said plurality of processors (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the assignment of an auxiliary (*e.g.*, spare) processor role to at least one of its processor cores. For example, Kalyanasundharam **taught that** "the communications from the primary processor core may be propagated and the communications from the **auxiliary processor core(s)** may be discarded…In this embodiment, **the PC bit** from the configuration register 42 may

identify the primary processor core." *Kalyanasundharam* (Ex. 1007) at 8:34-44. "**The configuration register 42** includes a primary core (PC) field encoded to **identify which of the processor cores 18A-18N** is designated as **the primary core.**" *Id.* at 8:34-44. "**In such** embodiments, one processor core may be the primary core and two or more of the remaining processor cores may be **auxiliary processor cores.**" *Id.* at 4:66 – 5:2. "**If the masked comparison** results indicate that the communication from the primary processor core **mismatches** with all auxiliary processor cores (that are enabled)…the control circuit 84 may **disable the primary processor core** and **select one of the auxiliary processor cores to be the primary processor core** (block 102)." *Id.* at 13:15-22. In view of these citations, Kalyanasundharam described assigning the role of auxiliary (*e.g.*, spare) processor to at least one of its processor cores. Accordingly, Kalyanasundharam as modified by Kennedy taught assigning the role of spare processor to one of its processor cores. *Id.*);

[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the replacement of a primary processor with an auxiliary processor in response to a lockstep mismatch. For example, Kalyanasundharam **taught that** "the processor cores 18A-18B may **execute in lockstep (that is, may execute the same instructions on the same clock cycle).**" *Kalyanasundharam* (Ex. 1007) at 4:15-18. "**If the masked comparison results indicate that the** communication from the primary processor core **mismatches** with all auxiliary processor cores (that are enabled)…the control circuit 84 may **disable the primary processor core** and **select one of the auxiliary processor cores to be the primary processor core** (block 102)." *Id.*at 13:15-22. "As with the discussion above for **disabling the auxiliary processor core**, the control circuit 83 may simply **disable the primary processor core from comparison** by clearing the enable bit in the configuration register 42, or may **disable the primary processor core from executing at all**." *Id.* at 13:22-27. "The control circuit 84 may also **update the PC field of the configuration register 42** to reflect the **newly selected primary processor core.**" *Id.* at 13:27-29. Accordingly, Kalyanasundharam described changing the role of primary processor to an auxiliary processor in response to a mismatch (*e.g.*, loss of lockstep) involving the primary processor. *See id.* In view of these citations and the knowledge that a boot processor was a common component in computer systems (*e.g.*, Kalyanasundharam's **system), a** POSITA would have understood

Kalyanasundharam as disclosing the failover of a boot processor to a spare processor in response to a loss of lockstep. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 178. Even if Kalyanasundharam did not explicitly identify the mismatched processor as having been the boot processor, Kalyanasundharam as modified by Kennedy would have changed the boot processor assignment in response to a lockstep mismatch. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy taught that, "if this 'default' **boot processor** fails[,]" then an **alternate processor** "should take over boot operations to enhance availability of the system." *Id.* In view of the above citations, Kalyanasundharam taught changing the role of primary processor to the spare processor in response to a loss of lockstep involving the primary processor, and Kennedy taught changing the role of boot processor to an alternate processor. *See Wolfe Decl.* (Ex. 1003) ¶ 178. A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught changing the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*);

[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a central boot ROM that is used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system. In other words, even if Kalyanasundharam did not explicitly disclose an operating system that recognizes the spare processor as the boot processor after the role change, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "a **central system boot ROM 150** accessible by all processors…may be **used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he CPU1 120 controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "The alternate boot processors…can execute the **boot code** in the **central boot ROM 150**". *Id.* at 10:42-45. A POSITA would have understood that, because Kennedy's

central boot ROM 150 is used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system, the operating system would recognize the spare processor as the boot processor after failover. *See Wolfe Decl.* (Ex. 1003) ¶ 179. Or at the very least, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 9 obvious to a POSITA due at least in part to Kennedy's central boot ROM 150 being used by the operating system and storing the boot code executed in conjunction with the operating system to boot the system. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.*).

Claim 10: Claim 10 recites subject matter that is substantially similar to claim 2 and is therefore rendered obvious for the same reasons as claim 2 above.

Claim 11: Claim 11 recites subject matter that is substantially similar to claim 2 and is therefore rendered obvious for the same reasons as claim 2 above.

Claim 12: ***The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a determinization that the alternate processor is installed in one of the slots of a backplane bus and that the boot processor is installed in another one of the slots of the backplane bus. Even if Kalyanasundharam did not explicitly disclose determining a processor that is coupled to a common communication bus with the boot processor, Kalyanasundharam as modified by Kennedy would have determined that the alternate processor and the boot processor are coupled to the same backplane bus. For example, according to Kennedy, "the system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, Kennedy taught that the boot processor and the alternate boot processors are installed in specific slots along the same backplane bus. Based on this disclosure from Kennedy, a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 180. As a result, any form of identifying such an alternate processor would constitute a determination of an alternate

processor coupled to a common communication bus with the boot processor. *Id.* A POSITA would have been motivated to combine these disclosures from Kennedy with Kalyanasundharam to improve the communication capabilities of Kalyanasundharam's system for fault tolerance and high availability. *Id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught this claim limitation. *Id.*); ***and***

[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the selection of one or more alternate (*e.g.,* spare) boot processors by installing the alternate boot processor into a specific slot corresponding to the alternate boot role on the backplane bus. Even if Kalyanasundharam did not explicitly disclose selecting the processor coupled to the common communication bus as the spare processor, Kalyanasundharam as modified by Kennedy would have selected the alternate processor installed in the same backplane bus as the boot processor to be the spare. For example, according to Kennedy, "the system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, Kennedy taught selecting one or more alternate (*e.g.,* spare) boot processors by installing those alternate boot processors into specific slots corresponding to that alternate boot role on the backplane bus. A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *See Wolfe Decl.* (Ex. 1003) ¶ 181. It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy taught this claim limitation. *Id.*).

Claim 13: ***The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this

limitation by disclosing a BIOS responsible for identifying a spare processor and holding the same in reserve as a hot spare. For example, Kalyanasundharam taught identifying a spare processor and holding the spare processor in reserve as a hot spare for at least the same reasons provided above in connection with claim 1. Even if Kalyanasundharam did not explicitly disclose doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors…**may be used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "**[T]he CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors**…can **execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 182. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. *Id.* A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to achieve cost savings by performing these tasks with firmware instead of using additional hardware. *See id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA. *Id.*)

Claim 14: ***The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a BIOS responsible for booting the spare processor as an active spare. For example, Kalyanasundharam taught booting the spare processor as an active spare for at least the same reasons provided above in connection with claim 6. Even if Kalyanasundharam did not explicitly disclose doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-

memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors…**may be used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "**[T]he CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors**…can **execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 183. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. *Id.* A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to enable Kalyanasundharam's system to dynamically modify the activity of spare processors (*e.g.,* between active spare and hot spare) to increase efficiency and/or throughput. *See id.* It was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA. *Id.*).

Claim 15: ***The method of claim 9 wherein system firmware performs said changing operation*** (Kalyanasundharam as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a BIOS included in an IOSM responsible for changing the role of boot processor to the spare processor. Kalyanasundharam taught performing the changing operation for at least the same reasons provided above in connection with claim 9. Even if Kalyanasundharam did not explicitly disclose doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught that each CPU includes a slot selection (sometimes referred to as "SS") logic for selecting which CPU is designated as the boot processor. "In general, the slot select logic 148 on the default boot CPU allows this CPU to boot." *Kennedy* (Ex. 1005) at 10:29-31, FIG. 3. Kennedy's **input/output services module ("IOSM")** is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. *See id.* at 6:36-38, FIG. 3. For example, "[a]ny other slot selection circuits on CPUs installed in the system have a

corresponding slot selection line from the **IOSM 108."** *Id.* As demonstrated by FIG. 3 of Kennedy reproduced below, Kennedy's IOSM includes the **BIOS**.

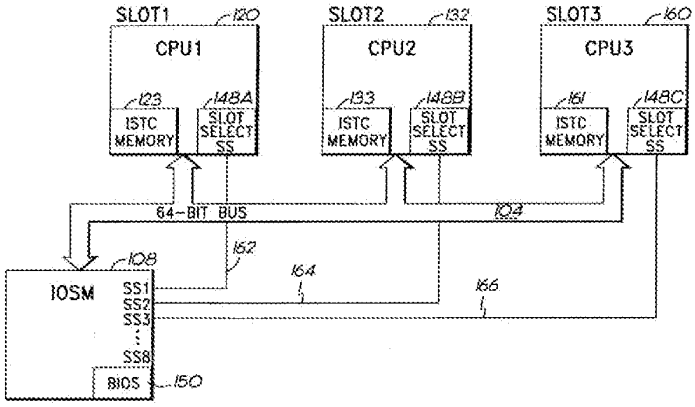


FIG. 3

The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 184. In view of the above citations and this common knowledge, a POSITA would have understood that **that Kennedy's BIOS, which constitutes firmware,** is responsible for changing the role of boot **processor to the spare processor in conjunction with the CPUs' slot** selection logic. *See id.* A POSITA would have been motivated to combine these disclosures from Kalyanasundharam and Kennedy to achieve cost savings by performing the change operation with firmware instead of using additional hardware. *See id.* It was well within the skill of a POSITA to modify Kalyanasundharam's **system by Kennedy in this manner, and thus this combination of** Kalyanasundharam and Kennedy would have been expected to succeed. *Id.* Accordingly, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA.).

2. Ground 2 Safford in view of Kennedy

Safford in view of Kennedy would have rendered claims 1-6 and 9-15 obvious to a POSITA under 35 U.S.C. § 103(a). For ease of reference, Requester has provided some proposed obviousness rejections for claims 1-6 and 9-15 below. Requester has also provided a detailed mapping of Safford and Kennedy to claims 1-6 and 9-15 in Appendix B.

Claim 1: ***[1.pre] A system comprising:*** (Safford disclosed a computer system. For example, Safford disclosed that "**a computer system** employs lock stepped processor cores that operate in a master/check pair." *Safford* (Ex. 1004) at 2:55-57.);

[1.1] a plurality of processors (Safford disclosed a plurality of processors included in the computer system. For example, Safford taught that "**a computer system employs** lock stepped processor cores that operate in a master/check pair. Each of **two processors** in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit located near external interfaces of the **two processors**." *Safford* (Ex. 1004) at 2:55-60. "**For** example, the node controller 130 may store the architected state of the **processors 111, 113, 121, 123, 125, and 127**, either internally in the node controller 130, or in another component **of the computer system 100.**" *Id.* at 4:6-10. Accordingly, Safford disclosed a plurality of processors included in the computer system.);

[1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors (Safford disclosed, or at least rendered obvious, this limitation by disclosing memory that stores the architected states of microprocessors, including the designation of a hot standby processor. For example, Safford taught that "**the architected states** of each of the microprocessors 22 and 24 is copied from **the memory 25 into the microprocessors 22 and 24, respectively.**" *Safford* (Ex. 1004) at 2:2-4. "**For example, the node controller 130 may store the architected state** of the processors 111, 113, 121, 123, 125, and 127, **either internally in the node controller 130**, or **in another component of the computer system 100.**" *Id.* at 4:6-10. "**To improve reliability of** processing assets, a computer system employs lock stepped processor cores that operate in a **master/check pair.**" *Id.* at 2:55-57. "**The processor 125 may be designated as a 'hot standby,'** and is sitting idle in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 suffer an error, the **hot standby processors 125, 127** may be used to speed recovery from the resulting loss of lock step**.**" *Id.* at 4:25-30. Accordingly, Safford described assigning the roles of master and checker to certain processors and the role of hot standby to another processor. *See Wolfe Decl.* (Ex. 1003) ¶ 187. Safford further described one pair of these **processors "operating in a lock step mode"** as initially "**appear**[ing] **to [a] computer system…to be…a logical CPU 0**" but later that **logical CPU 0** designation is reassigned to another processor. *See Safford* (Ex. 1007) at 3:4-5, 4:52-53 ("**The processor pair 125/127 then becomes the logical CPU 0** in the computer

system 100"). A POSITA would have recognized Safford's logical CPU 0 as being the designation used by computing system 100 to identify its boot processor at least because, in multi-processor system architectures like Safford's, "logical CPU 0" was commonly used to indicate that a processor had been selected as a boot processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 187. Even if Safford is somehow interpreted to lack an explicit disclosure of identifying which processor was the boot processor, Safford as modified by Kennedy would have stored information about the assigned roles of boot processor and spare processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy taught that, "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. In view of the above citations, Safford taught data storage that stores the architected states, such as role assignments (*e.g.,* master/checker), of paired processors, and Kennedy taught that such assignments could be the roles of boot processor and spare processor. *See Wolfe Decl.* (Ex. 1003) ¶ 187. A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to improve the reliability of Safford's system by storing information that assigns the roles of boot processor and spare processor in case of a reboot. *Id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. *Id.*);

[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the failover of a processor to a hot standby processor in response to a loss of lockstep. For example, Safford taught that, "[u]sing the impending **loss of lock step signal**, the computer system 100 may **continue operating (processing) using one of the processors 111 and 113 that did not experience an error.**" *Safford* (Ex. 1004) at 3:27-30; "The six processors 111, 113, 121, 123, 125, and 127 operate in lock step (i.e., are processing code sequences). For example, the processor 111 operates in lockstep with the processor 113, and the processor 121 operates in lock step with the processor 123 and the processor 125 operates in lock step with the processor 127." *Id.* at 4:19-24. "The processor 125 may be designated as a 'hot standby,' and is sitting idle in lock step mode

with the processor 127. Should one of the processors 111, 113, 121, and 123 **suffer an error**, the **hot standby processors 125, 127** may be **used to speed recovery from the resulting loss of lock step.**" *Id.* at 4:25-30. "In block 210, the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step** and that the first processor pair 111/113 should be **taken 'off-line.'**" *Id.* at 4:37-40. Accordingly, Safford described replacing a failed processor with a hot spare processor in response to a loss of lockstep. *See Wolfe Decl.* (Ex. 1003) ¶ 188. The '781 Patent notes that a firmware component known as a **processor abstraction layer (PAL) "copies all** the needed information from memory into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor." Ex. 1001 at 17:57-61. Safford disclosed this limitation through an identical process for transferring the role of boot processor to the spare processor. *See Safford* (Ex. 1004) at 32:29-32. For example, Safford taught **"moving an architected state of the processor unit experiencing the loss of lock step to the spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system."** *Id.* In view of these citations and the knowledge that a boot processor was a common component in computer systems (*e.g.*, **Safford's system), a** POSITA would have understood Safford as disclosing the failover of a boot processor to a hot spare processor in response to a loss of lockstep. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 188. However, even if Safford did not explicitly identify the failed processor as having been the boot processor, Safford as modified by Kennedy would have performed boot processor failover. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, Kennedy **taught that, "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system."** *Id.* In view of the above citations, Safford taught logic that causes its system to replace a failed processor with a hot spare processor in response to a loss of lockstep, and Kennedy taught logic that causes its system to fail over the boot processor to an alternate processor. *See Wolfe Decl.* (Ex. 1003) ¶ 188. A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to improve the reliability **of Safford's system** by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *Id.* It was well within the skill of a POSITA to modify **Safford's system by Kennedy in this** manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught

logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*);

[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the failover to a hot standby processor that is sitting idle until the loss of lockstep. For example, Safford taught that "[t]**he processor 125 may be designated as a 'hot standby,' and is sitting idle** in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 suffer an error, the **hot standby** processors 125, 127 may be used to speed recovery from the resulting loss of lock step." *Safford* (Ex. 1004). at 4:25-30. Accordingly, Safford as modified by Kennedy taught that the processor assigned the role of spare is held in reserve until a loss of lockstep is detected.).

Claim 2: ***The system of claim 1 wherein said data storage is non-volatile data storage*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing memory that stores the architected states of microprocessors. For example, as explained above in connection with claim 1, Safford as modified by Kennedy disclosed data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. Even if Safford did not explicitly identify the data storage as being non-volatile data storage, Safford as modified by Kennedy would have included non-volatile data storage that stores information about the assignments of the boot processor and the spare processor. For example, "[t]he CPU1 120 has a corresponding **non-volatile memory 123** which contains configuration information and CPU specific initialize and self test code (ISTC) for the CPU1 120[.]" *Kennedy* (Ex. 1005), 5:15-18. From these disclosures, a POSITA would have understood that the data storage that stores the information assigning the roles of boot processor and spare processor included non-volatile memory. *See Wolfe Decl.* (Ex. 1003) ¶ 189. Accordingly, Safford as modified by Kennedy would have rendered the subject matter of claim 2 obvious to a POSITA. *Id.*).

Claim 3: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the selection of a processor type to boot the system. In other words, even if Safford did

not explicitly disclose a user interface that receives input specifying the processor assigned the role of boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor types** may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* It was well known in the art that the act of selecting processor roles in computer systems (*e.g.,* master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 190. Based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure about selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Safford by Kennedy and this common knowledge in the art to enable a user to select which processor is to boot the system via a user interface. *Id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA. *Id.*).

Claim 4: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by implicitly disclosing the selection of a processor type that does not boot the system. In other words, even if Safford did not explicitly disclose a user interface that receives input specifying the processor assigned the role of spare processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor types** may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* at 4:33-34. It was well known in the art that the act of selecting processor roles in computer systems (*e.g.,* master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 191. Based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure of selecting the

processor to boot the system as including a spare processor selection (*e.g.*, one of the alternate processors not selected as the boot processor) made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Safford by Kennedy and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. *Id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA. *Id.*).

Claim 5: ***The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the selection of a processor type to boot the system. For example, Safford taught that **"a computer system employs lock stepped** processor cores that operate in a **master/check pair."** *Safford* (Ex. 1004) at 2:55-57. **"The processors** operate as pairs when in lock step (i.e., the processors 111 and 113 are a pair; the processors 121 and 123 are a second pair ; and the processors 125 and 127 are a third pair). From **the node controller's perspective, each pair of processors** appears as a **single (logical) processor."** *Id.* at 3:57-63. Accordingly, Safford as modified by Kennedy taught a plurality of processor modules that each include a master processor and a slave processor. *See Wolfe Decl.* (Ex. 1003) ¶ 192.).

Claim 6: ***The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a spare processor being active before a role transfer. For example, Safford taught that "the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken 'off-line.'"** *Safford* (Ex. 1004) at 4:37-40. **"In block 230, the node controller 130 'reboots' the processors 111 and 113**, and the processors 111 and 113 become the **new 'hot standby' processor pair** on the system 100." *Id.* at 4:57-60. Safford also **taught that "the spare processor unit becomes an active processor unit in the computer system"** *Id.*, claim 1. In view of the above citations, a POSITA would have understood Safford as disclosing that a spare processor can be active before a role transfer. *See Wolfe Decl.* (Ex. 1003) ¶ 193. Moreover, a POSITA would have understood that the

role at issue could be the boot processor assignment in view of Safford as modified by Kennedy for at least the same reasons provided above in connection with claim 1. *Id.* A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to improve the reliability of Safford's system by providing high availability through active and/or inactive spare processors. *Id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught that the spare processor is an active spare before being transferred the role of boot processor. *Id.*).

Claim 9: ***[9.pre] A method comprising:*** (Safford disclosed a method. "An apparatus, operating on an advanced multi-core processor architecture, and a corresponding **method**, are used to enhance recovery from loss of lock step in a multi-processor computer system." *Safford* (Ex. 1004), Abstract);

[9.1] assigning a role of boot processor to one of a plurality of processors in a system (Safford disclosed, or at least rendered obvious, this limitation by disclosing the architected states of microprocessors that operate as a master/checker pair as well as the designation of the logical CPU 0. For example, Safford taught that "the **architected states of each of the microprocessors 22 and 24** is copied from the memory 25 into the microprocessors 22 and 24, respectively." *Safford* (Ex. 1004) at 2:2-4. "**For example, the node controller 130 may store the architected state of the processors 111, 113, 121, 123, 125, and 127**, either internally in the node controller 130, or in another component of the computer system 100." *Id.* at 4:6-10. "To improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a **master/check pair.**" *Id.* at 2:55-57. Accordingly, Safford described assigning the roles of master and checker to certain processors. *See Wolfe Decl.* (Ex. 1003) ¶ 194. Safford further described one pair of these processors "operating in a lock step mode" as initially "appear[ing] to [a] computer system…to be…a **logical CPU 0**" but later that **logical CPU 0** designation is reassigned to another processor. *See Safford* (Ex. 1007) at 3:4-5, 4:52-53 ("The processor pair 125/127 then becomes the **logical CPU 0** in the computer system 100"). A POSITA would have recognized Safford's logical CPU 0 as being the designation used by computing system 100 to identify its boot processor at least because, in multi-processor system architectures like Safford's, "**logical CPU 0**" was commonly used to indicate that a processor had been selected as a boot processor. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 194. However, even if Safford is interpreted as not

explicitly identifying which processor was the boot processor, Safford as modified by Kennedy would have assigned the role of boot processor. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. For example, **Kennedy taught that, "if this 'default' boot processor fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system."** *Id.* at 7:9-13. In view of the above citations, Safford taught assigning roles (*e.g.,* master/checker) to paired processors, and Kennedy taught that one such assignment could be the role of boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 194. A POSITA would have been motivated to combine these disclosures from Safford **and Kennedy to improve the reliability of Safford's system** by assigning the role of boot processor to a certain processor in case of a reboot. *Id.* It was well within the skill of a POSITA to modify **Safford's system by Kennedy in this manner, and** thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught assigning a role of boot processor to one of a plurality of processors in a system. *Id.*);

[9.2] assigning a role spare processor to another of said plurality of processors (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the designation of a hot standby processor. For example, Safford taught that "[t]he processor 125 may **be designated as a 'hot standby,' and is** sitting idle in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 suffer an error, the **hot standby processors 125, 127 may be used to speed recovery** from the resulting loss of lock step." *Safford* (Ex. 1004) at 4:25-30. Accordingly, Safford as modified by Kennedy taught assigning a role of spare processor to another one of the plurality of processors in the system. *Id.*);

[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing failing over a processor to a hot standby processor in response to a loss of lockstep. For example, Safford taught that, "[u]sing the impending **loss of lock step signal**, the computer system 100 may **continue operating (processing) using one of the processors 111 and 113 that did not experience an error.**" *Safford* (Ex. 1004) at 3:27-30. **"The processor 125 may be designated as a 'hot standby,' and is sitting idle in lock step mode with the processor 127. Should one of the** processors 111, 113, 121, and 123 **suffer an error**, the **hot standby processors 125, 127** may be **used to speed recovery from the resulting loss of lock step**." *Id.* at 4:25-30. "In block 210, the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken**

lock step and that the first processor pair 111/113 should be **taken 'off-line.'"** *Id.* at 4:37-40. Accordingly, Safford described replacing a failed processor with a hot spare processor in response to a loss of lockstep. *See id.* at 3:27-30, 4:37-54. The '781 Patent notes that a firmware component known as a **processor abstraction layer (PAL) "copies all the needed information from memory** into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor." Ex. 1001 at 17:57-61. Safford disclosed this claim limitation through an identical process for transferring the role of boot processor to the spare processor. *Safford* (Ex. 1004) at 32:29-32. For example, Safford taught **"moving an architected state of the processor unit experiencing the loss of lock step to the spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system."** *Id.* In view of this disclosure and the knowledge that a boot processor was a common component in computer systems (*e.g.*, **Safford's system), a** POSITA would have understood Safford as disclosing the failover of a boot processor to a hot spare processor in response to a loss of lockstep. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 195. However, even if Safford is not interpreted as explicitly identifying the failed processor as having been the boot processor, Safford as modified by Kennedy would have performed boot processor failover. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In view of the above citations, Safford taught replacing a failed processor with a hot spare processor in response to a loss of lockstep, and Kennedy taught failing over the boot processor to an alternate processor. *See Wolfe Decl.* (Ex. 1003) ¶ 195. A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to **improve the reliability of Safford's system** by providing high availability in the event that the boot processor fails before a reboot occurs. *Id.* It was well within the skill of a POSITA to modify **Safford's system by Kennedy in** this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught changing, during system runtime, the assigned role of boot processor to the spare processor in response to loss of lockstep for the boot processor. *Id.*);

[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the improved availability of processor assets to avoid a computer system crash and by disclosing a central boot ROM that is

used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system. For example, in focusing on "**improv[ing] availability** of the processor assets of the computer system", Safford notes that "loss of lock step, if not promptly corrected, may **cause the computer system 18 to crash.**" *Safford* (Ex. 1004) at 3:21-22, 1:45-47. Even if Safford did not explicitly disclose an operating system that recognizes the spare processor as the boot processor without requiring shutdown, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "a **central system boot ROM 150** accessible by all processors…may be used by the **operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he CPU1 120 controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "The **alternate boot processors**…can execute the **boot code in the central boot ROM 150".** *Id.* at 10:42-45. A POSITA would have understood that, because Kennedy's central boot ROM 150 is used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system, the operating system would recognize the spare processor as the boot processor after failover without requiring shutdown of the operating system, especially if Kennedy is to achieve its objective of providing fault tolerance and high availability. *See Kennedy* (Ex. 1005) at 9:1-5, 9:53-55, 10:42-45; *see also Wolfe Decl.* (Ex. 1003) ¶ 196. This conclusion of a POSITA's understanding is also supported by Safford's objective of avoiding system crashes. *See Safford* (Ex. 1004) at 3:21-22, 1:45-47. A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to improve the reliability of Safford's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *See Wolfe Decl.* (Ex. 1003) ¶ 196. It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy would have rendered obvious to a POSITA an operating system that recognizes the spare processor as the boot processor without requiring shutdown. *Id.*).

Claim 10: Claim 10 recites subject matter that is substantially similar to claim 2 and is therefore rendered obvious for the same reasons as claim 2 above.

Claim 11: Claim 11 recites subject matter that is substantially similar to claim 2 and is therefore rendered obvious for the same reasons as claim 2 above.

Claim 12: ***The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality***

processors assigned to the role of boot processor (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a determinization that the alternate processor is installed in one of the slots of a backplane bus and that the boot processor is installed in another one of the slots of the backplane bus. Even if Safford did not explicitly disclose determining a processor that is coupled to a common communication bus with the boot processor, Safford as modified by Kennedy would have determined that the alternate processor and the boot processor are coupled to the same backplane bus. For example, according to Kennedy, "the system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, Kennedy taught that the boot processor and the alternate boot processors are installed in specific slots along the same backplane bus. Based on this disclosure from Kennedy, a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 197. As a result, any form of identifying such an alternate processor would constitute a determination of an alternate processor coupled to a common communication bus with the boot processor. *Id.* A POSITA would have been motivated to combine these disclosures from Kennedy with Safford to improve the communication capabilities of **Safford's system** for fault tolerance and high availability. *Id.* **It was well within the skill of a POSITA to modify Safford's** system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught this claim limitation. *Id.*); ***and***

[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing the selection of one or more alternate (*e.g.,* spare) boot processors by installing the alternate boot processor into a specific slot corresponding to the alternate boot role on the backplane bus. Even if Safford did not explicitly disclose selecting the processor coupled to the common communication bus as the spare processor, Safford as modified by Kennedy would have selected the alternate processor installed in the same backplane bus as the boot processor to be the spare**. For example, according to Kennedy,** "the system has a centrally accessible memory, **a default boot central processing unit** installed in a

first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. A POSITA would have been motivated to combine these disclosures from Kennedy with Safford to improve the reliability of Safford's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. *See Wolfe Decl.* (Ex. 1003) ¶ 198. It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy taught this claim limitation. *Id.*).

Claim 13: ***The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a BIOS responsible for identifying a spare processor and holding the same in reserve as a hot spare. Safford taught identifying a spare processor and holding the spare processor in reserve as a hot spare for at least the same reasons provided above in connection with claim 1. Even if Safford did not explicitly disclose doing so via system firmware, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors…may be used by the operating system." *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors**…can **execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 199. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. *Id.* A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to achieve cost savings by performing these tasks with firmware instead of using additional hardware. *See id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been

expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA. *Id.*).

Claim 14: ***The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a BIOS responsible for booting the spare processor as an active spare. Safford taught booting the spare processor as an active spare for at least the same reasons provided above in connection with claim 6. Even if Safford did not explicitly disclose doing so via system firmware, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors…may be used by the operating system." *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors**…can **execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 200. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. *Id.* A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to enable Safford's system to dynamically modify the activity of spare processors (*e.g.*, between active spare and hot spare) to increase efficiency and/or throughput. *See id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA. *Id.*).

Claim 15: ***The method of claim 9 wherein system firmware performs said changing operation*** (Safford as modified by Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a BIOS included in an IOSM responsible for changing the role of boot processor to the spare processor. Safford taught performing the changing operation for at least the same reasons provided above in connection with claim 9. Even if Safford did not explicitly disclose doing so via

system firmware, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught that each CPU includes a slot selection (sometimes referred to as "SS") logic for selecting which CPU is designated as the boot processor. "In general, the slot select logic 148 on the default boot CPU allows this CPU to boot." *Kennedy* (Ex. 1005) at 10:29-31, FIG. 3. Kennedy's **input/output services module ("IOSM")** is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. *See id.* at 6:36-38, FIG. 3. For example, "[a]ny other slot selection circuits on CPUs installed in the system have a corresponding slot selection line from the **IOSM** 108." *Id.* As demonstrated by FIG. 3 of Kennedy reproduced below, Kennedy's IOSM includes the **BIOS**.

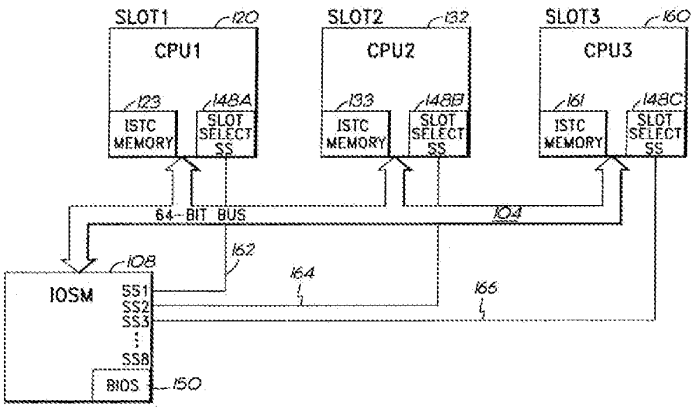


FIG. 3

The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 201. In view of the above citations and this common knowledge, a POSITA would have understood that that Kennedy's BIOS, which constitutes firmware, is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. *Id.* A POSITA would have been motivated to combine these disclosures from Safford and Kennedy to achieve cost savings by performing the change operation with firmware instead of using additional hardware. *See id.* It was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, Safford as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA.).

3. Ground 3 Kennedy in view of Nguyen and Fox

Kennedy in view of Nguyen and Fox would have rendered claims 1-30 obvious to a POSITA. For ease of reference, Requester has provided some proposed obviousness rejections for claims 1-30 below. Requester has also provided a detailed mapping of Kennedy in view of Nguyen and Fox to claims 1-30 in Appendix C.

Claim 1: ***[1.pre] A system comprising:*** (Kennedy disclosed a system. For example, Kennedy taught "multiprocessor computer **systems.**" *Kennedy* (Ex. 1005) at 1:14-15.);

[1.1] a plurality of processors (Kennedy disclosed this limitation by disclosing multiprocessor computer systems that enable a boot processor to fail over to an alternate processor. For example, Kennedy taught "**multiprocessor** computer systems." *Kennedy* (Ex. 1005) at 1:14-15. Kennedy's disclosure "relates to a method and apparatus to automatically control the determination of **which processor in the system will boot the system.**" *Id.* at 1:15-17. Kennedy further disclosed that, "[i]f **the boot processor** fails, control automatically shifts to an **alternate processor.**" *Id.* at 1:17-19. Accordingly, Kennedy taught a plurality of processors.);

[1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors (Kennedy disclosed, or at least rendered obvious, this limitation by disclosing memories that hold configuration information about processors that are assigned the roles of boot processor and spare processor. For example, Kennedy taught that "[t]he CPU ISTC **memories hold configuration information.**" *Kennedy* (Ex. 1005) at 7:33-35. Kennedy also taught that, "[a]s previously explained, the ISTC **memory** associated with each CPU **contains…configuration information about the CPU**. Thus, the **boot CPU** determines how many CPUs are present on the bus." *Id.* at 9:16-21. Kennedy further taught that, "if this 'default' **boot processor** fails[,]" then an **alternate processor** "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught that "**one default boot CPU** is designated to **boot the system**, and if this CPU fails, control transfers automatically to an **alternate processor.**" *Id.* at 2:17-21. In view of the above citations, Kennedy taught memories that store CPU configuration information that includes the roles of boot processor and alternate processor in the system. *See Wolfe Decl.* (Ex. 1003) ¶ 204. Accordingly, Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. *Id.*).

[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor (Kennedy as modified by Nguyen disclosed, or at least rendered obvious, this limitation by disclosing the transfer of the boot processor role to the alternate processor in response to a loss of lockstep. For example, Kennedy taught that, "if this 'default' **boot processor fails[,]**" then an **alternate processor** "should **take over boot operations** to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught that "**one default boot CPU is designated to boot the system**, and **if this CPU fails**, **control transfers automatically to an alternate processor.**" *Id.* at 2:17-21. Even if Kennedy did not explicitly identify the boot processor's failure as being a loss of lockstep, Kennedy as modified by Nguyen would have detected a loss of lockstep. *See, e.g., Nguyen* (Ex. 1008) ¶ [0008], [0024], [0054]-[0055]. For example, Nguyen taught that "[e]xecution cores 120 **operate in lock step.**" *Nguyen* (Ex. 1008) ¶ [0024]. "FRC-hardware **compares results generated by each core**, and **if a discrepancy is detected**, the FRC system passes control to an error-handling routine." *Id.* ¶ [0006]. "However, for FRC-enabled processors, such errors are likely to be manifested as **FRC errors**, since they **take the execution cores out of lock-step.**" *Id.* ¶ [0008]. "[E]ach execution core may include a status bit that is set if an error is detected. The **master execution core may** check this bit **to determine if it has also generated an error.**" *Id.* ¶ [0050]. Accordingly, Nguyen described detecting a loss of lockstep. In view of the above citations, Kennedy taught logic that causes its system to replace the boot processor with an alternate processor in response to a failure of the boot processor, and Nguyen taught detecting an error (*i.e.*, a loss of lockstep). *See Wolfe Decl.* (Ex. 1003) ¶ 205. A POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the reliability of Kennedy's system by providing fault tolerance and high availability in the event that the boot processor fails through a loss of lockstep. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen taught logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*).

[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered

obvious, this limitation by disclosing the transfer of the boot processor role to the spare processor in response to detecting loss of lockstep and by disclosing the automatic failover to a hot spare processor. For example, Kennedy as modified by Nguyen taught a spare processor that assumes the role of boot processor in response to a loss of lockstep for at least the reasons provided above. Even if Kennedy did not explicitly identify the spare processor as having been a hot spare processor that is held in reserve, Kennedy as modified by Nguyen and Fox would have transferred the role of boot processor to a hot spare processor. *See, e.g., Fox* (Ex. 1006), Abstract. For example, according to Fox, "[t]he invention provides a substantially seamless transfer of executing processes from the failing processor to the replacement processor, without halting the execution of the operating system (OS) and/or applications on the overall MP. That is, a failing processor is able to automatically fail-over to a **hot-spare processor** without affecting the OS or executing applications." *Fox* (Ex. 1006) at 4:16-23. In view of the above citations, Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, transferring the role of boot processor to a hot spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 206. A POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to decrease delays in failing over from the boot processor to the spare processor. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught that the processor assigned the role of spare is held in reserve as a hot spare processor until the loss of lockstep is detected for the boot processor. *Id.*).

Claim 2: ***The system of claim 1 wherein said data storage is non-volatile data storage***

(Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing non-volatile memory that stores the configuration information in the system. For example, as explained above in connection with claim 1, Kennedy as modified by Nguyen and Fox taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. Kennedy further taught that "[t]he CPU1 120 has a corresponding **non-volatile memory 123** which contains configuration information and CPU specific initialize and self test code (ISTC) for the CPU1 120." *Kennedy* (Ex. 1005), 5:15-18. From these disclosures, a POSITA would have understood that the data storage that stores the information assigning the roles of boot processor and spare processor

included non-volatile memory. *See Wolfe Decl.* (Ex. 1003) ¶ 207. Accordingly, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 2 obvious to a POSITA. *Id.*).

Claim 3: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor*** (Kennedy as modified by Nguyen, Fox, and common knowledge in the art rendered obvious this limitation by disclosing the selection of a processor type to boot the system. In other words, even if Kennedy did not explicitly disclose a user interface that receives input specifying the processor assigned the role of boot processor, Kennedy as modified by Nguyen, Fox, and common knowledge in the art would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with **a corresponding change in the boot ROM 150, as well known in the art.**" *Id.* It was well known in the art that the act of selecting processor roles in computer systems (*e.g.*, master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 208. Based on this common knowledge in the art, a POSITA would have **understood Kennedy's** disclosure of selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Kennedy by Nguyen, Fox, and this common knowledge in the art to enable a user to select which processor is to boot the system via a user interface. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen, Fox, and the common knowledge in this manner, and thus this combination of Kennedy, Nguyen, Fox, and the common knowledge would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen, Fox, and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA. *Id.*).

Claim 4: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor*** (Kennedy as modified by Nguyen, Fox, and common knowledge in the art rendered obvious this limitation by implicitly disclosing the selection of a processor type that does not boot the system. In other words, even if Kennedy did not explicitly disclose a user interface that receives input specifying the processor assigned the role of spare processor, Kennedy as modified by

Nguyen, Fox, and common knowledge in the art would have rendered this claim limitation obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* at 4:33-34. It was well known in the art that the act of selecting processor roles in computer systems (*e.g.,* master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. *See Wolfe Decl.* (Ex. 1003) ¶ 209. Based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure of selecting the processor to boot the system as including a spare processor selection (*e.g.,* one of the alternate processors not selected as the boot processor) made by input received via a user interface. *Id.* A POSITA would have been motivated to modify Kennedy by Nguyen, Fox, and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. *Id.* A POSITA would have been motivated to modify Kennedy by Nguyen, Fox, and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen, Fox, and the common knowledge in this manner, and thus this combination of Kennedy, Nguyen, Fox, and the common knowledge would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen, Fox, and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA. *Id.*).

Claim 5: ***The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing multicore processors that each include master and slave execution cores. Even if Kennedy did not explicitly disclose that each of its processors is a processor module that includes master and slave processors, Kennedy as modified by Nguyen and Fox would have incorporated processor modules that each include master and slave execution cores. For example, Nguyen "**relates to microprocessors** and, in particular, to mechanisms for handling errors in **FRC-enabled processors.**" *Nguyen* (Ex. 1008) ¶ [0002]. Nguyen also taught "a mechanism for handling recoverable and non-recoverable errors in **multi-core processors.**" *Id.* ¶ [0058]. "**Processor 110** includes first and second **execution cores 120(a), 120(b).**" *Id.* ¶ [0020]. "For one embodiment, **one of execution cores** may be designated

as the **master core** and **the other** as the **slave core** when **processor 10 is operating in FRC mode."** *Id.* ¶ [0047]. In view of the above citations, Kennedy as modified by Nguyen and Fox taught multicore processor modules that each include master and slave execution cores. *See Wolfe Decl.* (Ex. 1003) ¶ 210. A POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to improve the spatial efficiency of Kennedy's system by consolidating the amount of space occupied by the processors. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught that each processor is a processor module including a master processor and a slave processor.).

Claim 6: ***The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing the spare as active before the role of boot processor is transferred. Even if Kennedy did not explicitly disclose that the alternate processor is an active spare before being transferred the role of boot processor, Kennedy as modified by Nguyen and Fox would have included the alternate processor as an active spare before being transferred the role of boot processor. For example, Fox's BIOS **"brings the standby processor** out of the held-off state and into an **active state."** *Fox* (Ex. 1006) at 7:8-10. In view of this disclosure, Kennedy as modified by Nguyen and Fox taught that the alternate processor is an active spare before being transferred the role of boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 211. A POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to improve the usage and/or efficiency of the processors in the system. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught that the spare is active before being transferred the role of boot processor. *Id.*).

Claim 7: ***The system of claim 1 further comprising: an operating system, wherein, responsive to the logic transferring the role of boot processor to the spare processor, the operating system recognizes the spare processor as the boot processor*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing the failover

to a hot spare processor without affecting the OS or executing applications. Even if Kennedy did not explicitly disclose an operating system that recognizes the spare processor as the boot processor after the role transfer, Kennedy as modified by Nguyen and Fox would have rendered this claim limitation obvious to a POSITA. For example, Fox taught that "[t]he invention provides a substantially **seamless transfer of executing processes** from the failing processor to the replacement processor, **without halting the execution of the operating system (OS) and/or applications on the overall MP**. That is, a failing processor is able to **automatically fail-over to a hot-spare processor without affecting the OS or executing applications.**" *Fox* (Ex. 1006) at 4:16-25. Fox also taught that, **"once the spare processor has been successfully brought on line**, the BIOS (table) is updated and **the OS** [and] bus controllers…are messaged to **change their configurations to reflect a processor change from the failing processor to the spare processor.**" *Id.* at 8:10-14. **"Control is then returned to the OS, which updates the 'active' processor list and allocates the threads that were running on the failing processor to the hot-spare processor."** *Id.*, Abstract, 3:28-31, 6:64-67. In view of the above citations, a POSITA would have understood that, because Fox's failing processor automatically fails over to a hot spare processor without affecting the operating system and then control is returned to the operating system which updates the "active" processor list, the operating system would recognize the spare processor as having assumed the role of the failing processor after the role transfer. *See Wolfe Decl.* (Ex. 1003) ¶ 212. Or at the very least, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 7 obvious to a POSITA due at least in part to Fox's failing processor automatically being replaced by a hot spare processor without affecting the operating system and then having control returned to the operating system which updates the "active" processor list. *Id.* It was well within the skill of a POSITA to modify Nguyen's system by Kennedy and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.*).

Claim 8: ***The system of claim 1 wherein said plurality of processors are Itanium Processor Family (IPF) processors*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing IPF processors. Even if Kennedy did not explicitly disclose that its processors are IPF processors, Kennedy as modified by Nguyen and Fox would have included IPF processors. For example, Nguyen taught that "[o]ne embodiment of the recovery mechanism employs code that is closely associated with the processor. For example, the

Itanium Processor Family of Intel Corporation employs a layer of firmware called processor abstraction layer (PAL), which provides an abstraction of the processor to the rest of the computing system." *Nguyen* (Ex. 1008) ¶ [0045]. In view of the above citations, Kennedy as modified by Nguyen and Fox taught that the processors would have been IPF processors. *See Wolfe Decl.* (Ex. 1003) ¶ 213. A POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to improve the spatial efficiency of the system by consolidating the processors into the multicore processor package used by the IPF. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught this claim limitation.).

Claim 9: ***[9.pre] A method comprising:*** (Kennedy disclosed a method. For example, Kennedy taught "**a method** and apparatus to automatically control the determination of which processor in the system **will boot the system.**" *Kennedy* (Ex. 1005) at 1:15-17.);

[9.1] assigning a role of boot processor to one of a plurality of processors in a system (Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a processor that is assigned the role of boot processor in a multiprocessor system. For example, Kennedy taught that, "if this 'default' **boot processor** fails[,]" **then an** alternate processor "**should take over boot** operations to enhance availability of the **system.**" *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught that "**one default boot CPU** is designated to **boot the system**, and if this CPU fails, control transfers automatically to an alternate processor." *Id.* at 2:17-21. Kennedy also taught "**multiprocessor** computer systems." *Id.* at 1:14-15. In view of the above citations, Kennedy taught assigning the role of boot processor to one of the processors in the system. *See Wolfe Decl.* (Ex. 1003) ¶ 214. Accordingly, Kennedy taught assigning the role of boot processor to one of the processors in the system. *Id.*);

[9.2] assigning a role spare processor to another of said plurality of processors (Kennedy disclosed, or at least rendered obvious, this limitation by disclosing a processor that is assigned the role of alternate processor in a multiprocessor system. For example, Kennedy taught **that, "if this 'default'** boot processor **fails[,]" then an alternate processor "should take over boot operations to enhance availability of the system."** *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught that "**one default** boot CPU is designated to boot the system, and if this CPU fails, control transfers automatically to an **alternate processor.**" *Id.* at 2:17-21. Kennedy also taught

"**multiprocessor** computer systems." *Id.* at 1:14-15. In view of the above citations, Kennedy taught assigning the role of alternate processor to one of the processors in the system. *See Wolfe Decl.* (Ex. 1003) ¶ 215. Accordingly, Kennedy taught assigning the role of spare processor to one of the processors in the system. *Id.*);

[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor (Kennedy as modified by Nguyen disclosed, or at least rendered obvious, this limitation by disclosing the replacement of the boot processor with the alternate processor in response to a loss of lockstep for the boot processor. For example, **Kennedy taught that, "if this 'default' boot processor fails[,]"** then an **alternate processor "should take over boot operations** to enhance availability of the **system."** *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught **that "one default boot CPU is designated to boot the system**, and **if this CPU fails**, **control transfers automatically to an alternate processor."** *Id.* at 2:17-21. Even if Kennedy did not explicitly identify the boot **processor's failure as being a loss of** lockstep, Kennedy as modified by Nguyen would have detected a loss of lockstep. *See, e.g., Nguyen* (Ex. 1008) ¶ [0008], [0024], [0054]-[0055]. For example, Nguyen taught that "[e]xecution cores 120 **operate in lock step."** *Nguyen* (Ex. 1008) ¶ [0024]. "FRC-hardware **compares results generated by each core**, and **if a discrepancy is detected**, the FRC system passes control to an error-**handling routine."** *Id.* ¶ [0006]. "**However,** for FRC-enabled processors, such errors are likely to be manifested as **FRC errors**, since they **take the execution cores out of lock-step."** *Id.* ¶ [0008]. "[E]ach execution core may include a status bit that is set if an error is detected. The **master execution core may** check this bit **to determine if it has also generated an error."** *Id.* ¶ [0050]. Accordingly, Nguyen described detecting a loss of lockstep. In view of the above citations, Kennedy taught replacing the boot processor with an alternate processor in response to a failure of the boot processor, and Nguyen taught detecting an error (*i.e.,* a loss of lockstep). *See Wolfe Decl.* (Ex. 1003) ¶ 216. A POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the reliability of Kennedy's system by providing fault tolerance and high availability in the event that the boot processor fails through a loss of lockstep. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by

Nguyen taught changing the assignment of the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*).

[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing the failover to a hot spare processor without affecting the OS or executing applications. Even if Kennedy did not explicitly disclose an operating system that recognizes the spare processor as the boot processor after the role transfer, Kennedy as modified by Nguyen and Fox would have rendered this claim limitation obvious to a POSITA. For example, Fox taught that "[t]he invention provides a substantially **seamless transfer of executing processes** from the failing processor to the replacement processor, **without halting the execution of the operating system (OS) and/or applications on the overall MP**. That is, a failing processor is able to **automatically fail-over to a hot-spare processor without affecting the OS or executing applications.**" *Fox* (Ex. 1006), 4:16-25. Fox also taught that**, "once the spare processor has been successfully brought on line**, the BIOS (table) is updated and **the OS** [and] bus controllers…are messaged to **change their configurations to reflect a processor change from the failing processor to the spare processor.**" *Id.* 8:10-14. **"Control is then returned to the OS, which updates the 'active' processor list and allocates the threads that were running on the failing processor to the hot-spare processor."** *Id.*, Abstract, 3:28-31, 6:64-67. In view of these citations, a POSITA would have understood that, because Fox's failing processor automatically fails over to a hot spare processor without affecting the operating system and then control is returned to the operating system which updates the "active" processor list, the operating system would recognize the spare processor as having assumed the role of the failing processor after failover. *See Wolfe Decl.* (Ex. 1003) ¶ 217. Or at the very least, Kennedy as modified by Nguyen and Fox would have rendered this subject matter obvious to a POSITA due at least in part to Fox's failing processor automatically failing over to a hot spare processor without affecting the operating system or applications. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.*).

Claim 10: Claim 10 recites subject matter that is substantially similar to Claim 2 and is therefore rendered obvious for the same reasons as Claim 2 above.

Claim 11: Claim 11 recites subject matter that is substantially similar to Claim 2 and is therefore rendered obvious for the same reasons as Claim 2 above.

Claim 12: ***The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing the processors assigned the roles of boot processor and spare processor as being coupled to a common communication bus. For example, Kennedy taught that "the system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. Based on this disclosure from Kennedy, a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. *See Wolfe Decl.* (Ex. 1003) ¶ 218. As a result, any form of identifying such an alternate processor would constitute a determination of an alternate processor coupled to a common communication bus with the boot processor. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught this claim limitation. *Id.*); ***and***

[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor (Kennedy as modified by Nguyen and Fox taught assigning the role of spare to a certain processor for at least the reasons provided above in connection with claim 1. Moreover, Kennedy as modified by Nguyen and Fox taught determining that the spare processor is coupled to the common communication bus for at least the same reasons provided above in connection with this claim. Accordingly, Kennedy as modified by Nguyen and Fox taught selecting a processor coupled to the common communication bus to be assigned the role of spare. *See Wolfe Decl.* (Ex. 1003) ¶ 219.).

Claim 13: ***The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing a BIOS responsible for identifying a spare processor and holding the same

in reserve as a hot spare. As explained above in connection with claim 1, Kennedy as modified by Nguyen and Fox taught that the processor assigned the role of spare is held in reserve as a hot spare processor until the loss of lockstep is detected for the boot processor. Moreover, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150 accessible by all processors…may be used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors…can execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 220. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 13 obvious to a POSITA. *Id.*).

Claim 14: ***The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing a BIOS responsible for booting the spare processor as an active spare. As explained above in connection with claim 6, Kennedy as modified by Nguyen and Fox taught that that the spare is active before being transferred the role of boot processor. Moreover, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors…may be used by the operating system." *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "The **alternate boot processors…can execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 221. In view of the above citations and this common knowledge, a POSITA would have understood

that Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 14 obvious to a POSITA. *Id.*).

Claim 15: ***The method of claim 9 wherein system firmware performs said changing operation*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing a BIOS included in an IOSM responsible for changing the role of boot processor to the spare processor. As explained above in connection with claim 9, Kennedy as modified by Nguyen and Fox taught performing the changing operation. Moreover, Kennedy taught that each CPU includes a slot selection (sometimes referred to as "SS") logic for selecting which CPU is designated as the boot processor. "In general, the slot select logic 148 on the default boot CPU allows this CPU to boot." *Kennedy* (Ex. 1005) at 10:29-31, FIG. 3. Kennedy's input/output services module ("IOSM") is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. *See id.* at 6:36-38, FIG. 3. For example, "[a]ny other slot selection circuits on CPUs installed in the system have a corresponding slot selection line from the IOSM 108." *Id.* As demonstrated by FIG. 3 of Kennedy reproduced below, Kennedy's IOSM includes the BIOS.

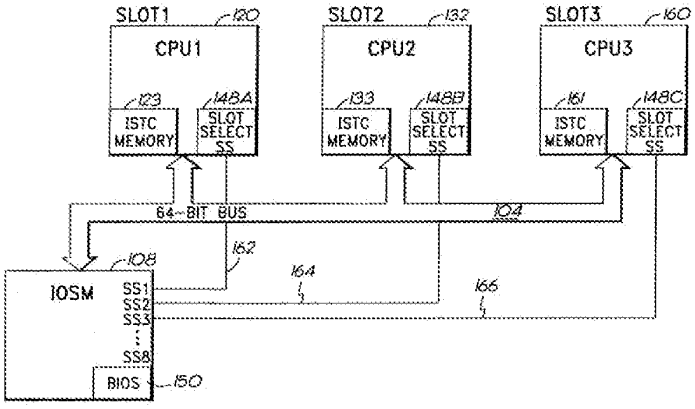


FIG. 3

The BIOS was a very well-known example of firmware. *Wolfe Decl.* (Ex. 1003) ¶ 222. In view of the above citations and this common knowledge, a POSITA would have understood that that Kennedy's BIOS, which constitutes firmware, is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. *Id.*

Accordingly, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 15 obvious to a POSITA. *Id.*)

Claim 16: ***[16.pre] A system comprising:*** (Kennedy disclosed a system. For example, Kennedy taught "multiprocessor computer **systems.**" *Kennedy* (Ex. 1005) at 1:14-15.);

[16.1] a plurality of lockstep processor modules that each includes a master processor and a slave processor that is in lockstep with said master processor (Kennedy as modified by Nguyen disclosed, or at least rendered obvious, this limitation by disclosing multicore processors that each include master and slave execution cores in lockstep. For example, Kennedy taught "**multiprocessor** computer systems." *Kennedy* (Ex. 1005) at 1:14-**15. Kennedy's disclosure** "relates to a method and apparatus to automatically control the determination of **which processor in the system** will boot the system." *Id.* at 1:15-17. Kennedy further disclosed **that, "[i]f the boot processor** fails, control automatically shifts to an **alternate processor.**" *Id.* at 1:17-19. Even if Kennedy did not explicitly disclose that each of its processors is a processor module that includes master and slave processors in lockstep, Kennedy as modified by Nguyen would have incorporated processor modules that each include master and slave execution cores in lockstep. For example, Nguyen "**relates to microprocessors** and, in particular, to mechanisms for handling errors in **FRC-enabled processors.**" *Nguyen* (Ex. 1008) ¶ [0002]. **Nguyen also taught "a mechanism for handling** recoverable and non-recoverable errors in **multi-core processors.**" *Id.* ¶ [0058]. **"Processor 110** includes first and second **execution cores 120(a), 120(b).**" *Id.* ¶ [0020]. **"For one embodiment, one of execution cores** may be designated as the **master core** and **the other** as the **slave core** when **processor 10 is operating in FRC mode.**" *Id.* ¶ [0047]. "Execution cores 120 **operate in lock step.**" *Id.* at ¶ [0024]. "FRC-hardware **compares results generated by each core**, and **if a discrepancy is detected**, the FRC system passes control to an error-**handling routine.**" *Id.* ¶ [0006]. **"However, for FRC**-enabled processors, such errors are likely to be manifested as **FRC errors**, since they **take the execution cores out of lock-step.**" *Id.* ¶ [0008]. In view of the above citations, Kennedy as modified by Nguyen taught multicore processor modules that each include master and slave execution cores in lockstep. *See Wolfe Decl.* (Ex. 1003) ¶ 223. A POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the spatial efficiency of Kennedy's system by consolidating the amount of space occupied by the processors. *Id.* It was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed.

Id. Accordingly, Kennedy as modified by Nguyen taught a plurality of lockstep processor modules that each include a master processor and a slave processor in lockstep with each other. *Id.*);

[16.2] a first communication bus with which at least a first two of said plurality of lockstep processor modules are communicatively coupled (Kennedy as modified by Nguyen disclosed, or at least rendered obvious, this limitation by disclosing the processors assigned the roles of boot processor and spare processor as being coupled to a common communication bus. For example, Kennedy taught that "the system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, Kennedy as modified by Nguyen taught that the boot processor and the alternate processor are installed in slots on the backplane bus. *See Wolfe Decl.* (Ex. 1003) ¶ 224. In other words, Kennedy's backplane bus is communicatively coupled to the boot processor and alternate processor. *See id.* Accordingly, Kennedy as modified by Nguyen taught a first communication bus with which at least a first two of the plurality of lockstep processor modules are communicatively coupled. *Id.*);

[16.3] a second communication bus with which at least a second two of said plurality of lockstep processor modules are communicatively coupled (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing the processors as being interconnected with each other via system buses, processor-to-processor buses, and control buses. As explained above in connection with this claim, Kennedy taught a backplane bus that is communicatively coupled to the boot processor and alternate processor. Even if Kennedy did not explicitly disclose a second communication bus with which at least a second two of the lockstep processor modules are communicatively coupled, Kennedy as modified by Nguyen and Fox would have rendered this claim limitation obvious to a POSITA. For example, Fox taught that "**[a]** system **bus 110** connects the memory controller 107 and other components to **IA-32 processors 120, 122, via bridges 112, and processor buses 114**. The IA-32 processors 120, 122 and the bridges 112 are grouped at nodes 111." *Fox* (Ex. 1006) at 5:9-12, FIG. 1. "**In one alternate implementation, processors 120, 122 are interconnected to each other via processor-to-processor buses 203.**" *Id.* at 5:41-43. "**Processors 120, 122** are **also connected to bridge (bus controller) 207 via individual control buses 205.**" *Id.* at 5:43-45. In view of these citations, Kennedy as modified by Nguyen and Fox taught a plurality of processors that are interconnected to each over via two or

more different buses (*e.g.*, system bus 110, processor-to-processor buses 203, and individual control buses 205). *See Wolfe Decl.* (Ex. 1003) ¶ 225. As further demonstrated by FIG. 1 of Fox reproduced below, Fox's system includes various processors that are communicatively coupled to one another via multiple communication buses.

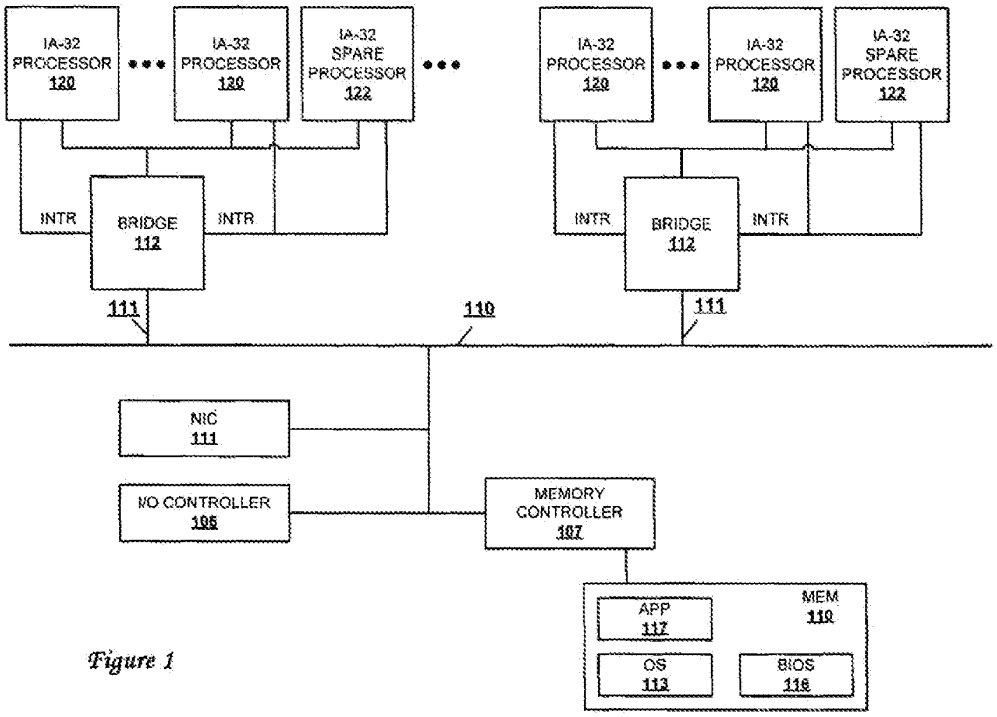


Figure 1

A POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen and Fox to provide multiple communication channels and/or mechanisms for communicating with the processors. *See Wolfe Decl.* (Ex. 1003) ¶ 225. It was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. *Id.* Accordingly, Kennedy as modified by Nguyen and Fox taught a second communication bus with which at least a second two of the plurality of lockstep processor modules are communicatively coupled. *Id.*);

[16.4] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors, wherein said spare processor is communicatively coupled to the same one of said

first and second communication buses as said boot processor (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by disclosing memories that hold configuration information about processors that are assigned the roles of boot processor and spare processor and communicatively coupled to the same communication bus. For example, Kennedy taught that "[t]he CPU ISTC **memories hold configuration information.**" *Kennedy* (Ex. 1005) at 7:33-35. Kennedy also taught that, "[a]s previously explained, the ISTC **memory** associated with each CPU **contains…configuration information about the CPU**. Thus, the **boot CPU** determines how many CPUs are present on the bus." *Id.* at 9:16-21. Kennedy further taught that, "if this 'default' **boot processor** fails[,]" then an **alternate processor** "should take over boot operations to enhance availability of the system." *Kennedy* (Ex. 1005) at 7:9-13. In addition, Kennedy taught that "**one default boot CPU** is designated to **boot the system**, and if this CPU fails, control transfers automatically to an **alternate processor.**" *Id.* at 2:17-21. Kennedy's "system has a centrally accessible memory, **a default boot central processing unit** installed in a first slot on the **backplane bus**, and may also have **one or more alternate boot central processing units** installed in a second slot on the **backplane bus.**" *Kennedy* (Ex. 1005) at 2:34-38. In view of the above citations, Kennedy as modified by Nguyen and Fox taught memories that store CPU configuration information that includes the roles of boot processor and alternate processor, which are assigned to processors coupled to the same communication bus. *See Wolfe Decl.* (Ex. 1003) ¶ 226. Accordingly, Kennedy as modified by Nguyen and Fox taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors, wherein the boot processor and the spare processor are coupled to the same communication bus. *Id.*).

Claim 17: Claim 17 recites subject matter that is substantially similar to claim 15, which depends from claim 9, and is therefore rendered obvious for the same reasons as claim 15 in view of claim 9 above.

Claim 18: Claim 18 recites subject matter that is substantially similar to claim 15, which depends from claim 9, and is therefore rendered obvious for the same reasons as claim 15 in view of claim 9 above.

Claim 19: Claim 19 recites subject matter that is substantially similar to claim 3 and is therefore rendered obvious for the same reasons as claim 3 above.

Claim 20: Claim 20 recites subject matter that is substantially similar to claim 4 and is

therefore rendered obvious for the same reasons as claim 4 above.

Claim 21: ***The system of claim 20 wherein said user interface presents for selection as said spare processor at least one processor that is communicatively to the same one of said first and second communication buses as said boot processor*** (Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, this limitation by implicitly disclosing the selection of a processor type that does not boot the system. For example, Kennedy as modified by Nguyen and Fox taught a spare processor that is communicatively coupled to the same communication bus as the boot processor for at least the same reasons provided above in connection with claim 16. Kennedy as modified by Nguyen and Fox also rendered obvious user interfaces that facilitate selecting the spare processor for at least the same reasons provided above in connection with claim 4. Accordingly, for at least the same reasons provided above in connection with claims 4 and 16, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 21 obvious to a POSITA. *See Wolfe Decl.* (Ex. 1003) ¶ 227.).

Claim 22: Claim 22 recites subject matter that is substantially similar to claim 13 and is therefore rendered obvious for the same reasons as claim 13 above.

Claim 23: Claim 23 recites subject matter that is substantially similar to claim 14 and is therefore rendered obvious for the same reasons as claim 14 above.

Claim 24: Claim 24 recites subject matter that is substantially similar to claim 8 and is therefore rendered obvious for the same reasons as claim 8 above.

Claim 25: Claim 25 recites subject matter that is substantially similar to claim 16 and is therefore rendered obvious for the same reasons as claim 16 above.

Claim 26: Claim 26 recites subject matter that is substantially similar to claim 18 and is therefore rendered obvious for the same reasons as claim 18 above.

Claim 27: Claim 27 recites subject matter that is substantially similar to claims 3 and 19 and is therefore rendered obvious for the same reasons as claims 3 and 19 above.

Claim 28: Claim 28 recites subject matter that is substantially similar to claims 3 and 19 and is therefore rendered obvious for the same reasons as claims 3 and 19 above.

Claim 29: Claim 29 recites subject matter that is substantially similar to claims 3 and 19 and is therefore rendered obvious for the same reasons as claims 3 and 19 above.

Claim 30: Claim 30 recites subject matter that is substantially similar to the combinations of claims 13 and 15 and is therefore rendered obvious for the same reasons as claims 13 and 15 above.

B. Proposed Anticipation Rejections for Claims 1 and 6

1. Ground 4 Safford in Light of Cepulis

Safford anticipates claims 1 and 6 under 35 U.S.C. § 102. In particular, when considered in light of the common technical knowledge that "CPU 0" is a reference for a boot processor as demonstrated in Cepulis, Safford discloses each and every limitation of claims 1 and 6. *See supra* §§ I.C.2, I.F.1, I.H.2, I.H.3. For ease of reference, Requester has provided some proposed obviousness rejections for claims and 6 below. Requester has also provided a detailed mapping of Safford in light of Cepulis to claims 1 and 6 in Appendix D.

Claim 1: ***[1.pre] A system comprising:*** (Safford disclosed a computer system. For example, Safford disclosed that "**a computer system** employs lock stepped processor cores that operate in a master/check pair." *Safford* (Ex. 1004) at 2:55-57.);

[1.1] a plurality of processors (Safford disclosed a plurality of processors included in the computer system. For example, Safford disclosed "**a computer system employs** lock stepped processor cores that operate in a master/check pair. Each of **two processors** in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit located near external interfaces of the **two processors**." *Safford* (Ex. 1004) at 2:55-60. "**For** example, the node controller 130 may store the architected state of the **processors 111, 113, 121, 123, 125, and 127**, either internally in the node controller 130, or in another component **of the computer system 100.**" *Id.* at 4:6-10. Accordingly, Safford disclosed a plurality of processors included in the computer system.);

[1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors (Safford disclosed this limitation, whether expressly or inherently, by disclosing memory that stores the architected states of microprocessors, including the designation of a hot standby processor, and by disclosing the logical CPU 0 designation. For example, Safford disclosed "**the architected states** of each of the microprocessors 22 and 24 is copied from **the memory 25 into the microprocessors 22 and 24, respectively.**" *Safford* (Ex. 1004) at 2:2-4. "**For** example, the node controller 130 may **store the architected state** of the processors 111, 113, 121,

123, 125, and 127, **either internally in the node controller 130**, or **in another component of the computer system 100.**" *Id.* at 4:6-10. "To improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a **master/check pair.**" *Id.* at 2:55-57. "The processor 125 may be **designated as a 'hot standby,'** and is sitting idle in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 suffer an error, the **hot standby processors 125, 127** may be used to speed recovery from the resulting loss of lock step." *Id.* at 4:25-30. Accordingly, Safford described assigning the roles of master and checker to certain processors and the role of hot standby to another processor. *See Wolfe Decl.* (Ex. 1003) ¶ 230. Safford further described one pair of these **processors "operating in a lock step mode"** as initially "appear[ing] to [a] computer system…to be…**a logical CPU 0**" but later that **logical CPU 0** designation is reassigned to another processor. *See Safford* (Ex. 1007) at 3:4-5, 4:52-53 ("The processor pair 125/127 then becomes the **logical CPU 0 in the computer system 100**"). A POSITA would have recognized Safford's logical CPU 0 as being the designation used by computing system 100 to identify its boot processor at least because, in multi-processor system architectures like Safford's, **"logical CPU 0" was commonly used to indicate that a processor had been selected** as a boot processor. *See, e.g., Cepulis* (Ex. 1013) at 2:52-64, *see also Wolfe Decl.* (Ex. 1003) ¶ 230. For example, during boot-processor selection,

> If CPU P0 is not functioning properly and cannot perform even the most basic functions, it is designated as inoperative. The hardware system then awakens CPU P1 and repeats the process of testing the CPU. If CPU P1 is operational, then it is designated as CPU L0, and it boots the remainder of the system. On the other hand, if CPU P1 also fails, it is also given an inoperative designation. The computer system then turns on CPU P2, and repeats the process. The process repeats until an operational microprocessor is found to perform the CPU L0 functions.[2]

Cepulis (Ex. 1013) at 2:52-64.

[2] Cepulis is provided herein to explain the meaning of the term "boot processor" used in Safford, the primary reference. *See* M.P.E.P. § 2131.01(B).

A POSITA would have also recognized "CPU 0" and "CPU 1" as being physical designations used by computing system 100 to identify processors 111 and 113. The use of both logical and physical designators by a computing system is generally necessary when the role of boot processor can be assigned and reassigned to a processor regardless of its physical position within a system. *See Wolfe Decl.* (Ex. 1003) ¶ 230; *see also Safford* (Ex. 1007) at 4:52-53 ("The processor pair 125/127 then becomes the **logical CPU 0** in the computer system 100"). Thus, a POSITA would have understood "**logical CPU 0**" to be the boot-processor designation used by an operating system to determine which of its processors is the boot processor and/or to distinguish its boot processor from its non-boot processors. *See Wolfe Decl.* (Ex. 1003) ¶ 230. Furthermore, because the architected states of Safford's processors are copied from memory, a POSITA would have understood the memory to be storing the boot-processor designation (*e.g.*, logical CPU 0) and the hot-standby designation for the processors so assigned by Safford's system. *See id.* In view of these citations and the knowledge that a boot processor was a common component identified as "logical CPU 0" in computer systems (*e.g.*, Safford's system), a POSITA would have understood Safford as disclosing data storage that stores information assigning the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 230.);

[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor (Safford disclosed this limitation by disclosing the failover of a processor to a hot standby processor in response to a loss of lockstep. For example, Safford disclosed "[u]sing the impending **loss of lock step signal**, the computer system 100 may **continue operating (processing) using one of the processors 111 and 113 that did not experience an error.**" *Safford* (Ex. 1004) at 3:27-30; "The six processors 111, 113, 121, 123, 125, and 127 operate in lock step (i.e., are processing code sequences). For example, the processor 111 operates in lockstep with the processor 113, and the processor 121 operates in lock step with the processor 123 and the processor 125 operates in lock step with the processor 127." *Id.* at 4:19-24. "The processor 125 may be designated as a '**hot standby**,' and is sitting idle in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 **suffer an error**, the **hot standby processors 125, 127** may be **used to speed recovery from the resulting loss of lock step.**" *Id.* at 4:25-30. "In block 210, the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step**

and that the first processor pair 111/113 should be **taken 'off-line.'"** *Id.* at 4:37-40. Accordingly, Safford described replacing a failed processor with a hot spare processor in response to a loss of lockstep. *See Wolfe Decl.* (Ex. 1003) ¶ 231. **The '781 Patent notes that a firmware component known as a processor abstraction layer (PAL) "copies all the needed information from memory** into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor**, transforming the role of the hot spare to boot processor."** Ex. 1001 at 17:57-61. Safford disclosed this limitation through an identical process for transferring the role of boot processor to the spare processor. *See Safford* (Ex. 1004) at 32:29-32. For example, Safford disclosed **"moving an architected state of the processor unit experiencing the loss of lock step to the spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system."** *Id.* Safford further described one pair of these processors "operating in a lock step mode" as initially "appear[ing] to [a] computer system…to be…**a logical CPU 0**" but later that **logical CPU 0** designation is reassigned to another processor. *See Safford* (Ex. 1007) at 3:4-5, 4:52-53 ("The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100"). As explained above, a POSITA would have understood "logical CPU 0" to be the boot-processor designation used by an operating system to determine which of its processors is the boot processor and/or to distinguish its boot processor from its non-boot processors. *See supra* § I.C.2; *see also Wolfe Decl.* (Ex. 1003) ¶ 231. In view of these citations and the knowledge that a boot processor was a common component **identified as "logical CPU 0"** in computer systems (*e.g.*, **Safford's system), a** POSITA would have understood Safford as disclosing failover of the boot processor (*e.g.*, logical CPU 0) to a spare processor in response to a loss of lockstep for the boot processor, especially in light of **Cepulis' explanation of the meaning of the "logical CPU 0" boot**-processor designator. *See Wolfe Decl.* (Ex. 1003) ¶ 231. Accordingly, Safford disclosed logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*);

[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor (Safford disclosed replacing the failed processor with a hot standby processor that is sitting idle in response to an error. For example, Safford disclosed that "[t]he **processor 125 may be designated as a 'hot standby,' and is sitting idle in lock step mode** with the processor 127. Should one of the processors 111, 113, 121, and 123 **suffer an error**, the **hot**

standby processors 125, 127 may be used to speed recovery from the resulting loss of lock step." *Safford* (Ex. 1004). at 4:25-30. Accordingly, Safford disclosed that the processor assigned the role of spare is a hot standby held in reserve until a loss of lockstep is detected.).

Claim 6: ***The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor*** (Safford disclosed a spare processor being active before a role transfer. For example, Safford disclosed that "the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step** and that the first processor pair 111/113 should be taken 'off-line.'" *Safford* (Ex. 1004) at 4:37-**40. "In block 230, the node controller 130 'reboots' the processors 111 and 113**, and the processors 111 and 113 become the **new 'hot standby' processor pair** on the system 100." *Id.* at 4:57-60. Safford also disclosed **that "the spare processor** unit becomes **an active processor unit in the computer system"** *Id.*, claim 1. In view of the above citations, a POSITA would have understood Safford as disclosing that a spare processor can be active before a role transfer. *See Wolfe Decl.* (Ex. 1003) ¶ 232. Moreover, a POSITA would have understood that the role at issue could be the boot processor assignment in view of Safford for at least the same reasons provided above in connection with claim 1. *Id.* Accordingly, Safford disclosed that the spare processor is an active spare before being transferred the role of boot processor. *Id.*).

C. Secondary Considerations

This Request demonstrates that the Challenged Claims of the '781 Patent are unpatentable as obvious in view of the prior art references. The Applicant did not identify any evidence of secondary considerations during prosecution. Further, the clear teachings in the prior art outweigh **any supposed "secondary considerations."** *Graham v. John Deere Co. of Kansas City*, 383 U.S. 1, 36 (1966).

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

Based on information available to Requester, the '781 Patent is the subject of three District Court litigations, as listed below. Requester is unaware of any prior reexamination or other post-grant proceeding in which the '781 Patent is or has been involved.

- *Foras Technologies, Ltd. vs Nissan Motor Co., Ltd.*, Case No. 1:23-cv-00640 (W.D. Tex., June 6, 2023).

- *Foras Technologies Ltd. v. Toyota Motor North America, Inc.*, Case No. 2:23-cv-00321 (E.D. Tex., July 12, 2023).
- *Foras Technologies Ltd. v. Toyota Motor North America, Inc., et al.*, Case No. 2:23-cv-00150 (E.D. Tex., April 5, 2023).

IV. CONCLUSION

Reexamination and cancellation of claims 1-30 of the '781 Patent is respectfully requested. The Commissioner is hereby authorized to charge Deposit Account 50-6990 under Docket No. 185945.012400 the Ex Parte Reexamination fee of $12,600 under 37 C.F.R. § 1.20(c)(1). Requester believes no other fee is due with this submission, however, the Commissioner is hereby authorized to charge any fee deficiency or credit any over-payment to Deposit Account 50-6990.

Please direct all correspondence in this matter to the undersigned.

Dated: August 31, 2023

Respectfully Submitted,

By: /Jon Bills/
Reg. No. 71,895

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

Counsel for Requester Unified Patents, LLC

EXHIBIT
1011

US007191328B2

(12) **United States Patent**
Hobson

(10) **Patent No.: US 7,191,328 B2**
(45) **Date of Patent: Mar. 13, 2007**

(54) **SYSTEM AND METHOD FOR USING AN EXTENSIBLE FIRMWARE INTERFACE (EFI) UTILITY TO BUILD AN EFI LAYER BETWEEN AN OPERATING SYSTEM AND A LEGACY BASIC INPUT/OUTPUT SYSTEM DURING A BOOT PROCESS**

(75) Inventor: **Louis B. Hobson**, Tomball, TX (US)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 483 days.

(21) Appl. No.: **10/894,808**

(22) Filed: **Jul. 20, 2004**

(65) **Prior Publication Data**

US 2006/0020780 A1 Jan. 26, 2006

(51) **Int. Cl.**
G06F 15/177 (2006.01)
G06F 9/00 (2006.01)

(52) **U.S. Cl.** **713/2**; 713/1; 710/1

(58) **Field of Classification Search** 713/1, 713/2; 710/1–74
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,080,243	B2 *	7/2006	Ramiz et al.	713/1
2005/0071618	A1 *	3/2005	Natu	713/1
2005/0204357	A1 *	9/2005	Garg et al.	718/1
2006/0047941	A1 *	3/2006	Lai	713/2

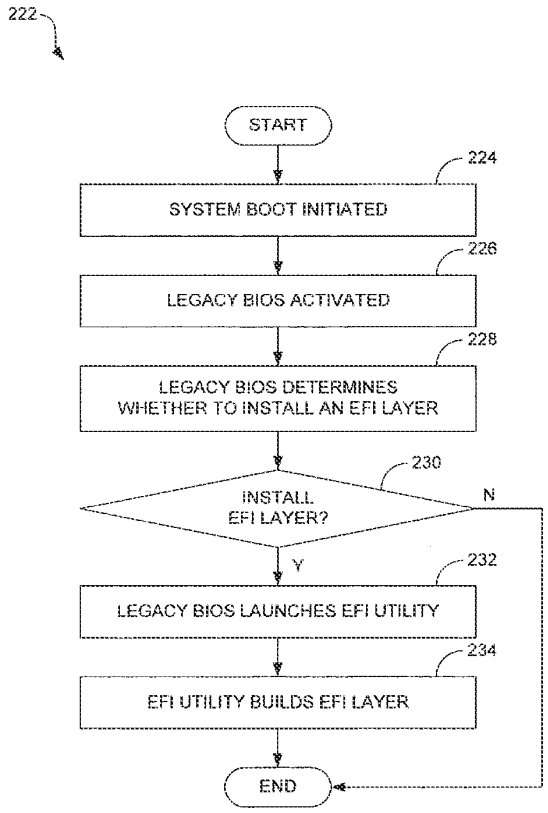
* cited by examiner

Primary Examiner—Thomas Lee
Assistant Examiner—Anand B. Patel

(57) **ABSTRACT**

Disclosed is a system and a method for implementing an extensible firmware interface (EFI). In one embodiment, a system and a method pertain to launching an EFI utility during a boot process of a computer system, and building an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility.

20 Claims, 5 Drawing Sheets



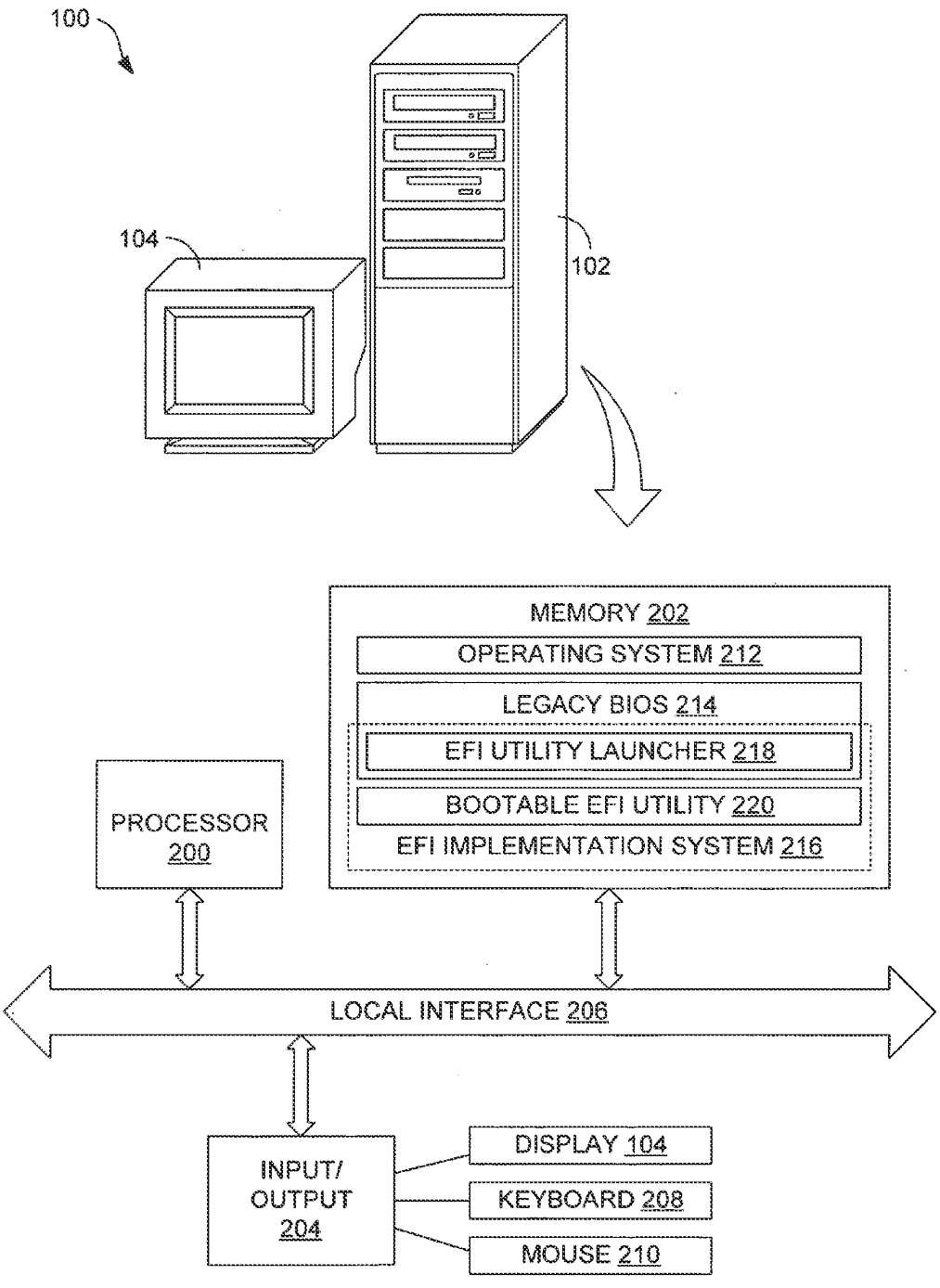

100
104
102
MEMORY 202
OPERATING SYSTEM 212
LEGACY BIOS 214
EFI UTILITY LAUNCHER 218
BOOTABLE EFI UTILITY 220
EFI IMPLEMENTATION SYSTEM 216
PROCESSOR 200
LOCAL INTERFACE 206
INPUT/ OUTPUT 204
DISPLAY 104
KEYBOARD 208
MOUSE 210

FIG. 1

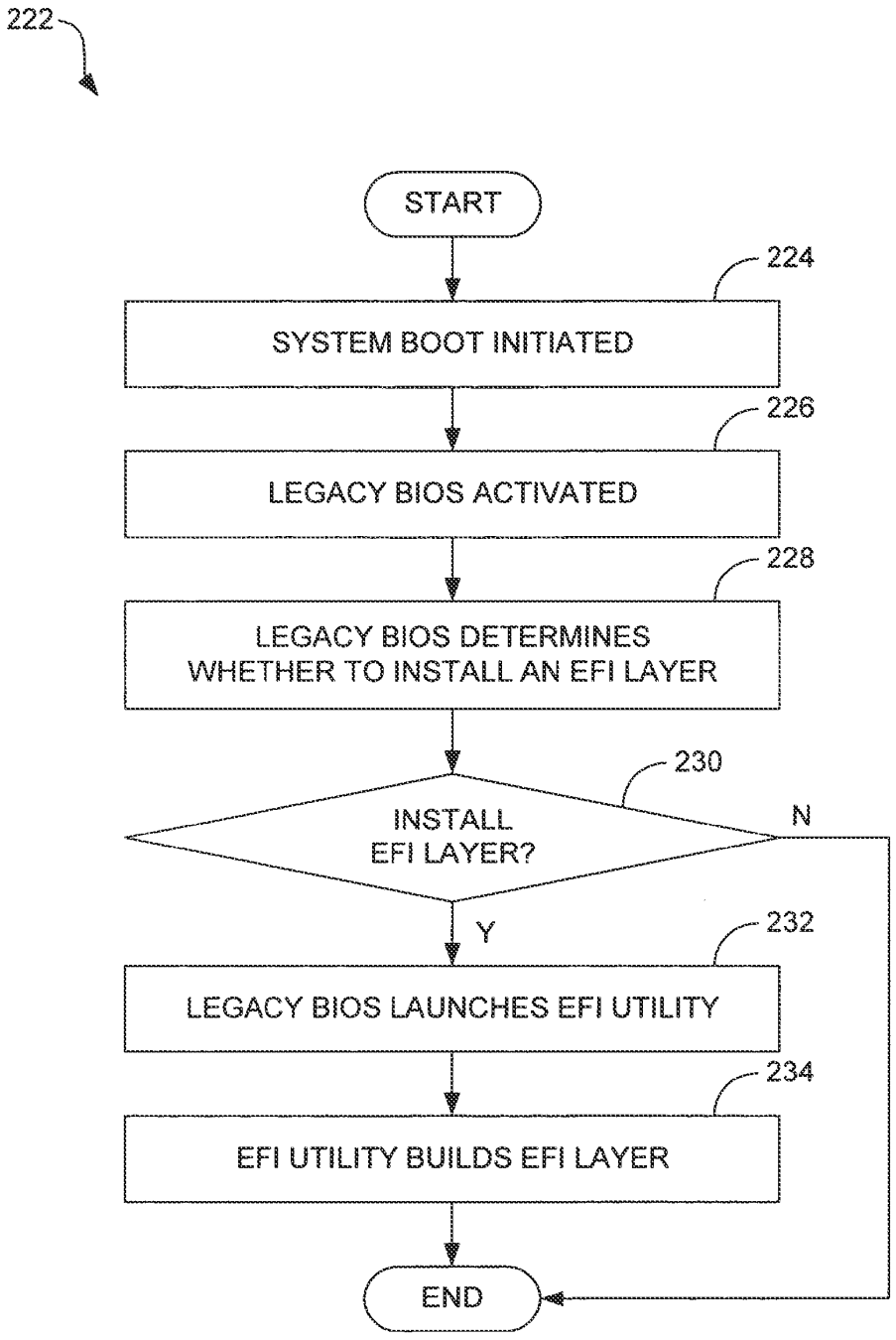

222
START
224
SYSTEM BOOT INITIATED
226
LEGACY BIOS ACTIVATED
228
LEGACY BIOS DETERMINES WHETHER TO INSTALL AN EFI LAYER
230
INSTALL EFI LAYER?
N
Y
232
LEGACY BIOS LAUNCHES EFI UTILITY
234
EFI UTILITY BUILDS EFI LAYER
END

FIG. 2

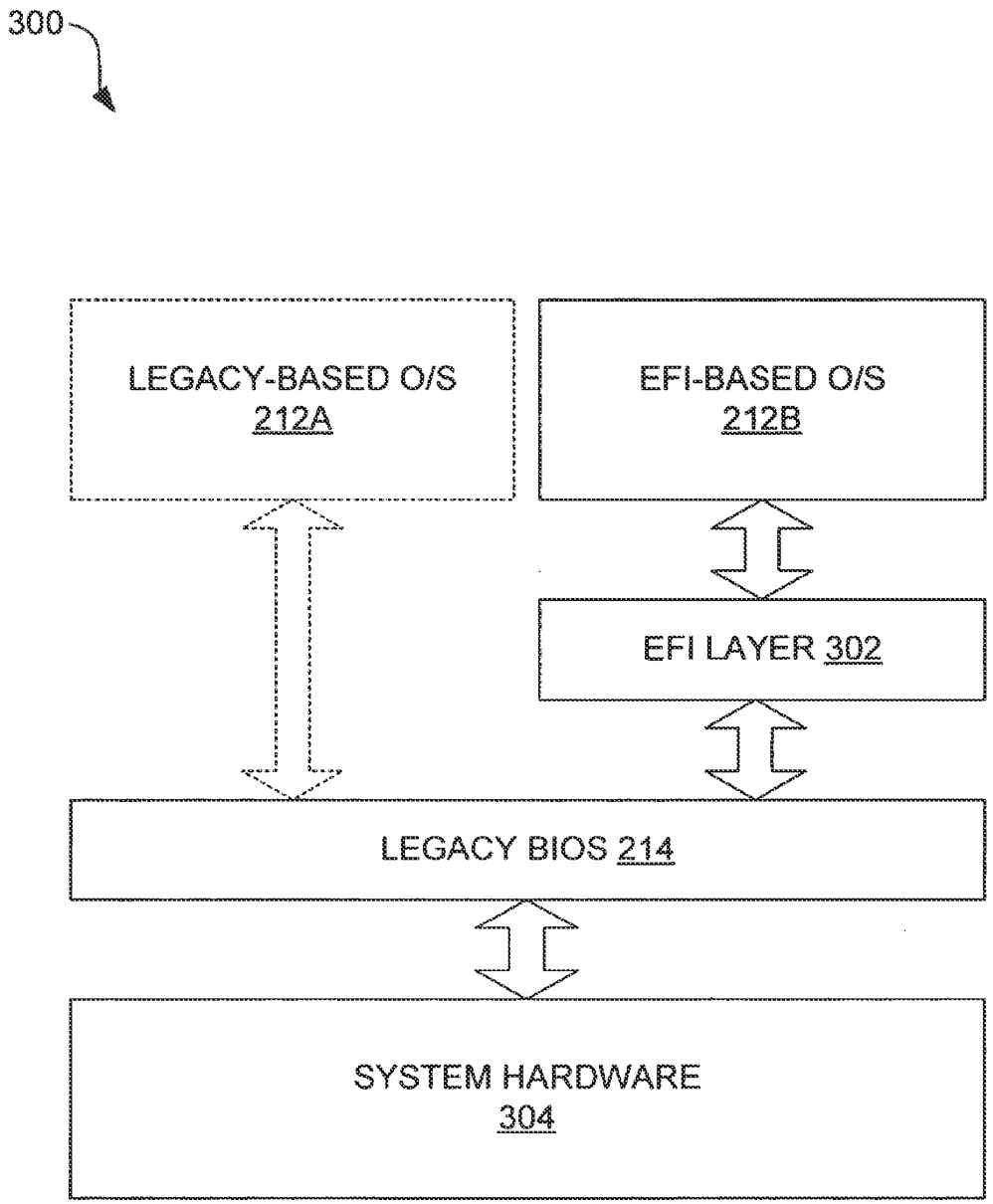

300
LEGACY-BASED O/S
212A
EFI-BASED O/S
212B
EFI LAYER 302
LEGACY BIOS 214
SYSTEM HARDWARE
304

FIG. 3

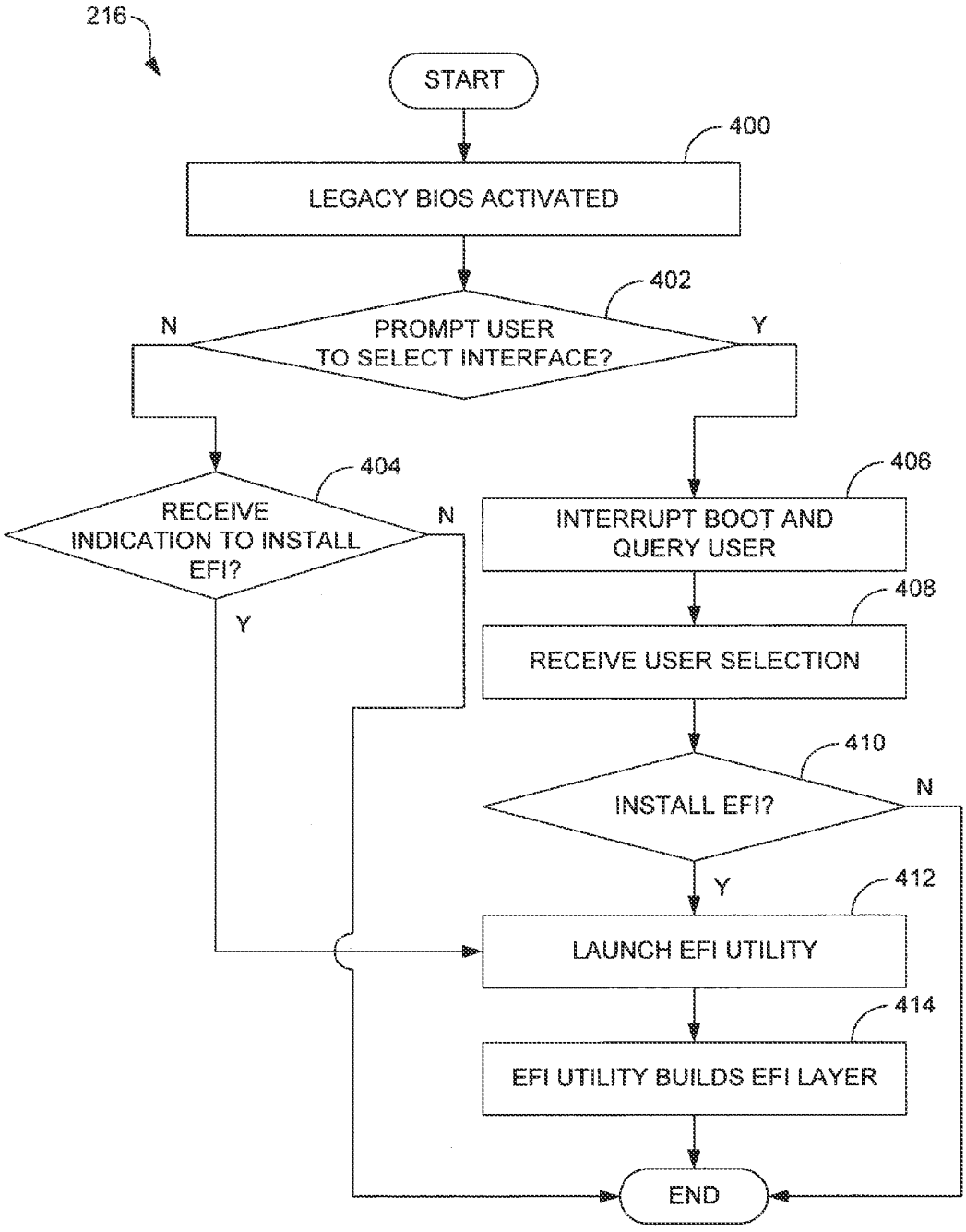

216
START
400
LEGACY BIOS ACTIVATED
402
PROMPT USER TO SELECT INTERFACE?
N
Y
404
RECEIVE INDICATION TO INSTALL EFI?
N
Y
406
INTERRUPT BOOT AND QUERY USER
408
RECEIVE USER SELECTION
410
INSTALL EFI?
N
Y
412
LAUNCH EFI UTILITY
414
EFI UTILITY BUILDS EFI LAYER
END

FIG. 4

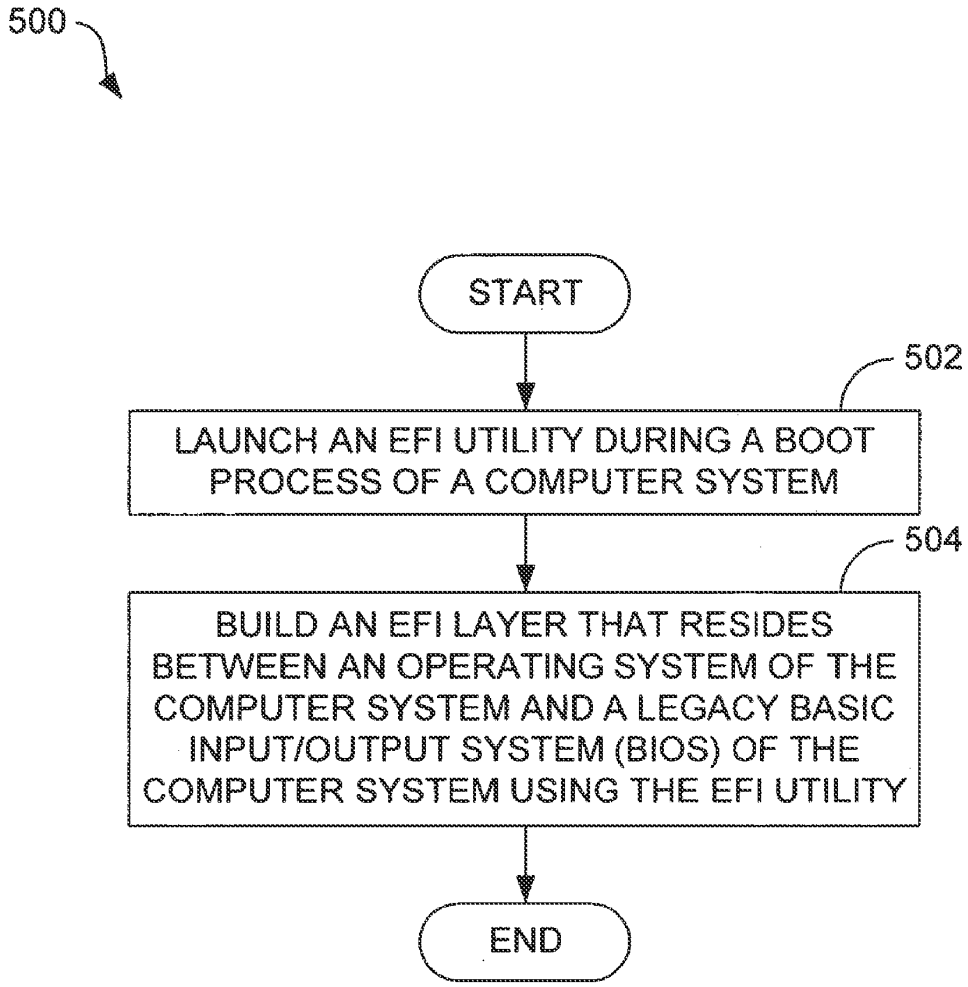

500
START
502
LAUNCH AN EFI UTILITY DURING A BOOT PROCESS OF A COMPUTER SYSTEM
504
BUILD AN EFI LAYER THAT RESIDES BETWEEN AN OPERATING SYSTEM OF THE COMPUTER SYSTEM AND A LEGACY BASIC INPUT/OUTPUT SYSTEM (BIOS) OF THE COMPUTER SYSTEM USING THE EFI UTILITY
END

FIG. 5

SYSTEM AND METHOD FOR USING AN EXTENSIBLE FIRMWARE INTERFACE (EFI) UTILITY TO BUILD AN EFI LAYER BETWEEN AN OPERATING SYSTEM AND A LEGACY BASIC INPUT/OUTPUT SYSTEM DURING A BOOT PROCESS

BACKGROUND

For years, personal computers (PCs) have used the so-called legacy basic input/output system (BIOS) to facilitate booting and to provide an interface between the computer operating system and various input/output (I/O) devices such as displays, keyboards, mice, and disk drives. Over the past several years, however, the Microsoft Corporation has been developing a new software interface, called the extensible firmware interface (EFI), that is expected to replace the legacy BIOS. The EFI provides a new model for the interface between operating systems and platform firmware and comprises data tables that contain platform-related information and boot and runtime service calls that are available to the operating system and its loader. In order to communicate with the EFI, the operating system must be specifically configured for use with the EFI, or other appropriate software must be developed to enable such communication.

Given that it appears that the PC industry will be transitioning to EFI-based systems, it is likely that future PCs will comprise the EFI. At least for the next few years, however, there will be many persons that will still use legacy-based operating systems that cannot communicate with the EFI. Accordingly, needed is an interim solution that supports both legacy BIOS and EFI so as to enable the use of either a legacy-based operating system or an EFI-based operating system.

Although both interface systems could be supported by installing both the legacy BIOS and the EFI in the system memory so that either is available for use, such a solution typically would require a larger system memory device (e.g., flash chip) to store both interface systems. Although such memory devices are available, their use would undesirably increase the cost of the PCs in which they are installed.

The Intel Corporation has proposed an alternative solution. Specifically, proposed is a backward-compatible solution in which the EFI layer is configured to emulate certain aspects of legacy BIOS operation so as to enable interface with legacy-based operating systems. In that solution, a compatibility support module (CSM) comprising a series of drivers chooses a legacy BIOS or EFI booting route depending upon the boot device(s) that is/are selected. Although this solution avoids the memory space problem identified above, such emulation is relatively complex and therefore requires significant time and resources to develop.

SUMMARY OF THE DISCLOSURE

Disclosed is a system and a method for implementing an extensible firmware interface (EFI). In one embodiment, a system and a method pertain to launching an EFI utility during a boot process of a computer system, and building an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility.

BRIEF DESCRIPTION OF THE DRAWINGS

The disclosed system and method can be better understood with reference to the following drawings. The components in the drawings are not necessarily to scale.

FIG. **1** is a schematic and block diagram of a computer system that incorporates a system for implementing an extensible firmware interface.

FIG. **2** is a flow diagram that illustrates an embodiment of a method for implementing an extensible firmware interface.

FIG. **3** is a block diagram representing a system configuration that results when an extensible firmware interface is implemented.

FIG. **4** is a flow diagram that illustrates an embodiment of operation of the EFI implementation system shown in FIG. **1**.

FIG. **5** is a flow diagram that illustrates a further embodiment of a method for implementing an extensible firmware interface.

DETAILED DESCRIPTION

Disclosed herein are embodiments of a system and method for implementing an extensible firmware interface (EFI). More particularly, disclosed are a system and method that comprise a legacy BIOS interface that is configured to, if desired and/or necessary, launch an EFI utility that installs an EFI layer between the BIOS and an EFI-based operating system during the boot process. With such operation, the system supports both legacy BIOS operation and EFI operation.

Referring now to the drawings, in which like numerals indicate corresponding parts throughout the several views, FIG. **1** illustrates a computer system **100** that incorporates a system for implementing an EFI and, more particularly, supporting both legacy BIOS and EFI operation. The computer system **100** comprises a central computing unit (CPU) **102** a display **104**. Also illustrated in FIG. **1** is an example architecture for the CPU **102**. In the illustrated embodiment, the CPU includes a processor **200**, memory **202**, and input/output components **204**, each of which is connected to a local interface **206**.

The processor **200** can include one or more microprocessors that control the logic of the digital devices of the computer system **100**. The memory **202** includes any one of or a combination of volatile memory elements (e.g., RAM) and nonvolatile memory elements (e.g., read only memory (ROM), flash memory, hard disk, etc.).

The I/O components **204** comprise the components with which inputs and outputs can be exchanged between the CPU **102** and a user input/output device such as the display **104**, a keyboard **208**, and a mouse **210**. In addition, the components **204** may, for example, comprise various ports, a modem, wireless transceiver, a telephonic interface, and a network card that enable communication with other devices.

The memory **202** comprises various programs including an operating system **212**, a legacy BIOS **214**, and an EFI implementation system **216**. The operating system **212** controls the execution of various computer system software and provides scheduling, input-output control, file and data management, memory management, and communication control and related services. As is described in the following, the operating system **212** may comprise a legacy-based operating system or an EFI-based operating system depending upon the particular implementation. Indeed, the CPU **102** may, in some embodiments, comprise both a legacy-based operating system and an EFI-based operating system.

The EFI implementation system **216** comprises the various programs that are used to enable interface with an EFI-based operating system. As is indicated in FIG. **1**, the EFI implementation system **216** comprises an EFI launcher **218** that is comprised by the legacy BIOS **214**, and a

bootable EFI utility **220** that is separate from the legacy BIOS **214**. By way of example, the legacy BIOS **214** resides within a non-volatile memory device provided on the system circuit board of the CPU **102**. The memory device may comprise a flash-based memory chip or an electrically-erasable, programmable read only memory (EEPROM). The memory device may be relatively small, for instance having a capacity of about 512 kilobytes (KB) so as to lower the cost of the device, and the CPU **102** in which it is provided.

The bootable EFI utility **220** is a utility that, when launched, builds an EFI as an interface layer between an operating system and a legacy BIOS. By way of example, the EFI utility **220** may comprise a publicly-available utility such as the EFI sample implementation source code (version 1.10.14.62) that is currently available from the Intel Corporation. Although use of that utility may be desirable due to its ready availability, other (e.g., later designed) versions of that utility or a custom-designed utility that is configured to build an EFI layer between an operating system and a legacy BIOS could be used instead.

Irrespective of its particular configuration, the EFI utility **220** resides outside of the legacy BIOS **214**. With such a configuration, the legacy BIOS **214** may be installed on the aforementioned memory device without the need to increase the size of that device. By way of example, the EFI utility **220** resides within a host protected area (HPA) of a hard disk of the CPU **102**. As is known, the HPA is an area of a computer hard disk that is available to the system BIOS but which is unknown and inaccessible to the operating system. In alternative arrangements, the EFI utility **220** may reside in another location, for instance on a bootable removable disk, such as a floppy disk, CD or DVD, that is inserted into a disk drive of the CPU **102**.

Various programs (logic) have been described above. These programs can be stored on any computer-readable medium for use by or in connection with any computer-related system or method. In the context of this disclosure, a computer-readable medium is an electronic, magnetic, optical, or other physical device or means that contains or stores a computer program for use by or in connection with a computer-related system or method. Programs can be embodied in any computer-readable medium for use by or in connection with an instruction execution system, apparatus, or device, such as a computer-based system, processor-containing system, or other system that can fetch the instructions from the instruction execution system, apparatus, or device and execute the instructions.

FIG. **2** is a flow diagram that describes a method **222** for implementing an EFI. Process steps or blocks in the flow diagrams of this disclosure may represent modules, segments, or portions of code that include one or more executable instructions for implementing specific logical functions or steps in the process. Although particular example process steps are described, alternative implementations are feasible. Moreover, steps may be executed out of order from that shown or discussed, including substantially concurrently or in reverse order, depending on the functionality involved.

Beginning with block **224** of FIG. **2**, a computer system boot is initiated, for instance when the user turns on the power of the computer system or resets the computer system. As a result of the boot initiation, the legacy BIOS is activated, as indicated in block **226**. After that activation, the legacy BIOS determines whether to install an EFI layer, as indicated in block **228**. That determination can be input in various ways. By way of example, the determination is made in relation to a selection input by the user. For instance, if the computer system comprises an EFI-based operating system, the user may elect to install the EFI layer.

Referring next to decision block **230**, flow from this point depends upon whether an EFI layer is to be installed. If not, flow for the EFI implementation portion of the boot process is terminated. In such a case, the legacy BIOS will complete the boot process and will operate as an interface between a legacy-based operating system and the various hardware of the computer system as is currently conventional. If, on the other hand, an EFI layer is to be installed, flow continues to block **232** at which the legacy BIOS launches the EFI utility that, for example, resides in the HPA. When the EFI utility is launched, the EFI utility takes control over the boot process and, as indicated in block **234**, builds an EFI layer between the operating system (presumably an EFI-based operating system) and the legacy BIOS. In such a case, the operating system will communicate with the EFI layer, but the legacy BIOS will still be available for communicating with the computer system hardware.

FIG. **3** illustrates an example implementation of an EFI. More particularly, FIG. **3** illustrates the system configuration that can result when the legacy BIOS launches the EFI utility as described above in relation to FIG. **2**. As is indicated in FIG. **3**, the system **300** comprises one or both of a legacy-based operating system **212**A and an EFI-based operating system **212**B. In cases in which the legacy-based operating system **212**A is present and is to be used, the operating system communicates directly with the legacy BIOS **214**. In cases in which the EFI-based operating system **212**B is to be used, however, the EFI utility builds an EFI layer **302** that resides between the operating system and the legacy BIOS **214**. Therefore, the operating system **212**B communicates with the EFI layer **302**. If the operating system **212**B needs to communicate with the system hardware **304**, for instance to read data from a system hard disk, a command is communicated to the EFI layer **302**, and the EFI layer makes an appropriate request of the legacy BIOS **214**, which is configured for direct communication with the system hardware.

FIG. **4** describes an example of operation of the EFI implementation system **214** shown in FIG. **1**. Beginning with block **400**, the legacy BIOS **214** is activated in a similar manner to that described above in relation to FIG. **2**. Once activated, the legacy BIOS **214** controls the boot process and, as indicated in decision block **402**, determines whether to prompt the user to select an interface to use. More particularly, the legacy BIOS **214** determines whether to prompt the user to elect between the default legacy BIOS interface or the EFI.

If the legacy BIOS **214** is not configured to prompt the user to choose, flow continues to decision block **404** at which it is determined whether an indication to install the EFI has been received. By way of example, such an indication can be input by the user by selecting an appropriate keyboard button, such as a function key (e.g., the "F10" key). If such an indication were made during the boot process, the legacy BIOS **214** could launch the EFI utility such that the EFI layer will be built and an EFI-based operating system can be used.

If no such indication is received, the system **216** defaults to the legacy BIOS interface and no EFI is installed. However, if an indication that the EFI is desired (e.g., if an EFI-based operating system will be used), flow continues to block **412** described below.

With reference back to decision block **402**, if the legacy BIOS **214** is configured to prompt the user to select an interface, flow continues to block **406** at which the BIOS

interrupts the boot process and queries the user to elect one of a legacy-based boot or an EFI-based boot. The legacy BIOS **214** then awaits a response to the query and, presumably, a user selection is received, as indicated in block **408**.

Referring next to decision block **410**, if the user selection indicates that the EFI is not to be installed, the legacy BIOS **214** completes the boot process and no EFI layer will be installed. If, on the other hand, an EFI layer is to be installed, flow continues to block **412** at which the EFI launcher **218** (e.g., of the legacy BIOS **214**) launches the EFI utility **220** that, for example, resides in the HPA. When the EFI utility **220** is launched, the EFI utility takes control over the boot process and, as indicated in block **414**, builds an EFI layer (**302**, FIG. **3**) between the operating system and the legacy BIOS in the manner described above and depicted in FIG. **3**.

With the above-described functionality, a single computer system can be created that is compatible with both legacy-based and EFI-base operating systems. Given that the EFI layer is not installed within the non-volatile memory device that comprises the legacy BIOS, the memory device can be kept relatively small. In addition, given that the legacy BIOS still resides within the computer system, the EFI need not emulate operation of the legacy BIOS, thereby providing a simple solution to the problem of supporting both legacy-based and EFI-based operating systems.

Although implementation of the EFI has been described in the foregoing as being user selected, alternative embodiments are possible. For instance, the legacy BIOS can be configured to determine the configuration of the operating system that will be used during the boot process (i.e., legacy-based or EFI-based) and automatically launch the EFI utility **220** if an EFI-based operating system is detected.

In view of the foregoing, a method **500** for implementing an extensible firmware interface can be described as is provided in FIG. **5**. As is indicated in that figure, the method **500** comprises launching an EFI utility during a boot process of a computer system (**502**), and building an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility (**504**).

What is claimed is:

1. A method for implementing an extensible firmware interface (EFI), the method comprising:

launching an EFI utility during a boot process of a computer system; and

building an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility.

2. The method of claim **1**, wherein launching an EFI utility comprises launching an EFI utility with the legacy BIOS.

3. The method of claim **1**, wherein launching an EFI utility comprises launching an EFI utility with the legacy BIOS that resides on a non-volatile memory device of the computing system.

4. The method of claim **1**, wherein launching an EFI utility comprises launching an EFI utility that resides in a host protected area (HPA) of a hard disk of the computing system.

5. The method of claim **1**, wherein launching an EFI utility comprises launching an EFI utility that resides on a removable disk that is insertable into a disk drive of the computing system.

6. The method of claim **1**, wherein building an EFI layer comprises building an EFI layer that is configured to communicate with an EFI-based operating system.

7. The method of claim **1**, further comprising prompting a user during the boot process to elect between installing the EFI layer and not installing the EFI layer.

8. The method of claim **7**, wherein prompting a user comprises interrupting the boot process with the legacy BIOS.

9. The method of claim **1**, further comprising receiving an indication from a user as to whether to install the EFI layer.

10. A system for implementing an extensible firmware interface (EFI), the system comprising:

means for determining whether to build an EFI layer on a computer system;

means for launching an EFI utility during a boot process of the computer system; and

means for building an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility.

11. The system of claim **10**, wherein the means for determining comprise part of the legacy BIOS.

12. The system of claim **10**, wherein the means for launching comprise an EFI utility launcher of the legacy BIOS.

13. The system of claim **10**, wherein the means for building comprise an EFI utility that resides in a host protected area (HPA) of a hard disk of the computing system.

14. An extensible firmware interface (EFI) implementation system stored on a computer-readable medium, the system comprising:

logic configured to launch an EFI utility during a boot process of a computer system; and

logic configured to build an EFI layer that resides between an operating system of the computer system and a legacy basic input/output system (BIOS) of the computer system using the EFI utility.

15. The system of claim **14**, wherein the logic configured to launch comprises an EFI utility launcher of the legacy BIOS.

16. The system of claim **14**, wherein the logic configured to build comprises the EFI utility that resides in a host protected area (HPA) of a hard disk of the computer system.

17. The system of claim **14**, wherein the logic configured to build comprises the EFI utility that resides on a removable disk that is insertable into a disk drive of the computer system.

18. The system of claim **14**, further comprising logic configured to determine whether to install an EFI layer.

19. A computer system, comprising:

a processor; and

a non-volatile memory device that comprises a legacy basic input/output system (BIOS), the legacy BIOS including an extensible firmware interface (EFI) utility launcher that is configured to launch an EFI utility during a boot process of the computer system; and

a hard disk that comprises an unprotected area that comprises an operating system and a host protected area (HPA) that comprises a bootable EFI utility that is configured to build an EFI layer that resides between the operating system and the legacy BIOS, the EFI layer being configured to communicate with the operating system and the legacy BIOS.

20. The computer system of claim **19**, wherein the non-volatile memory device comprises a flash-based memory chip.

* * * * *

EXHIBIT
1013

US005491788A

United States Patent [19]

Cepulis et al.

[11] Patent Number: 5,491,788

[45] Date of Patent: Feb. 13, 1996

[54] **METHOD OF BOOTING A MULTIPROCESSOR COMPUTER WHERE EXECUTION IS TRANSFERRING FROM A FIRST PROCESSOR TO A SECOND PROCESSOR BASED ON THE FIRST PROCESSOR HAVING HAD A CRITICAL ERROR**

[75] Inventors: **Darren J. Cepulis**, Houston; **Louis R. Gagliardi**, Tomball, both of Tex.

[73] Assignee: **Compaq Computer Corp.**, Houston, Tex.

[21] Appl. No.: **119,424**

[22] Filed: **Sep. 10, 1993**

[51] **Int. Cl.**[6] **G06F 11/00**

[52] **U.S. Cl.** **395/182.11**; 395/182.21; G06F/11/00

[58] **Field of Search** 395/575, 182.11, 395/182.21, 182.22; 371/11.3, 16.1, 67.1, 68.3

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,181,940	1/1980	Underwood et al.	364/200
4,634,110	1/1987	Julich	371/11.3
5,155,731	10/1992	Yamaguchi	371/16.5
5,247,659	9/1993	Carran et al.	395/575
5,408,647	4/1995	Landry	395/575
5,418,955	5/1995	Ikeda et al.	395/700

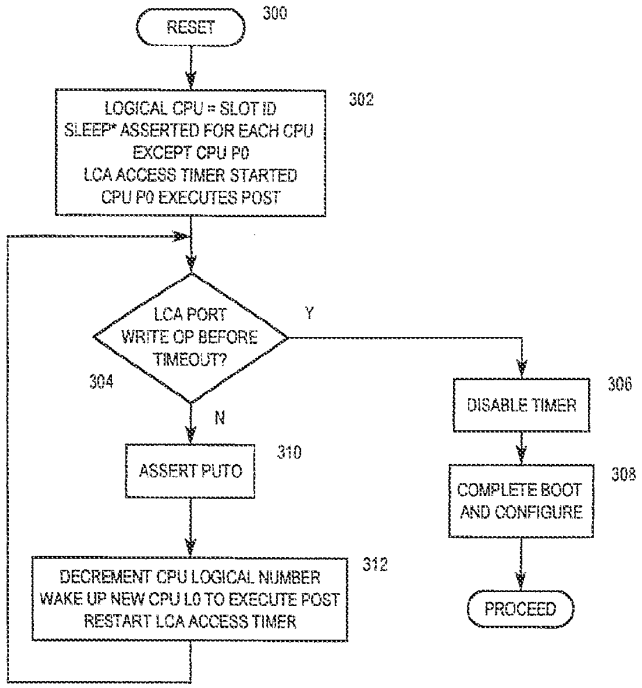
Primary Examiner—Hoa T. Nguyen

Attorney, Agent, or Firm—Pravel, Hewitt, Kimball & Krieger

[57] **ABSTRACT**

A multiprocessor computer system handles the failure of one or more of its processors without totally disabling the system. On power up, all of the CPUs are deactivated except for a CPU in a first physical slot. The power on self test routines review a log of errors and determine if certain critical errors have previously occurred. If so, the CPU in the first physical slot halts operation entirely. If the CPU in the first physical slot is not functioning properly or is halted, the hardware then awakens a CPU in a second physical slot, designates it as the first logical CPU, and the CPU then performs similar diagnostic checks. If it fails, the hardware again tries a third physical CPU and so on. When one CPU passes the initial error review, it proceeds with initialization of the computer system and performs further self testing. If it functions properly, it is designated as the first logical CPU, and retains its designation until the power is cycled. This first logical CPU then awakens the remaining CPUs and boots the rest of the system. If it fails this later self testing by having certain critical errors occur, the logical CPU 0 designation is transferred to another active CPU and the old CPU is halted. The new CPU commences operation effectively where the old CPU halted, so that system initialization is continued not restarted. The power on self test routines then further test the CPU in the first physical slot. Thus, if at least one CPU is operational, the computer system boots and operates.

8 Claims, 13 Drawing Sheets



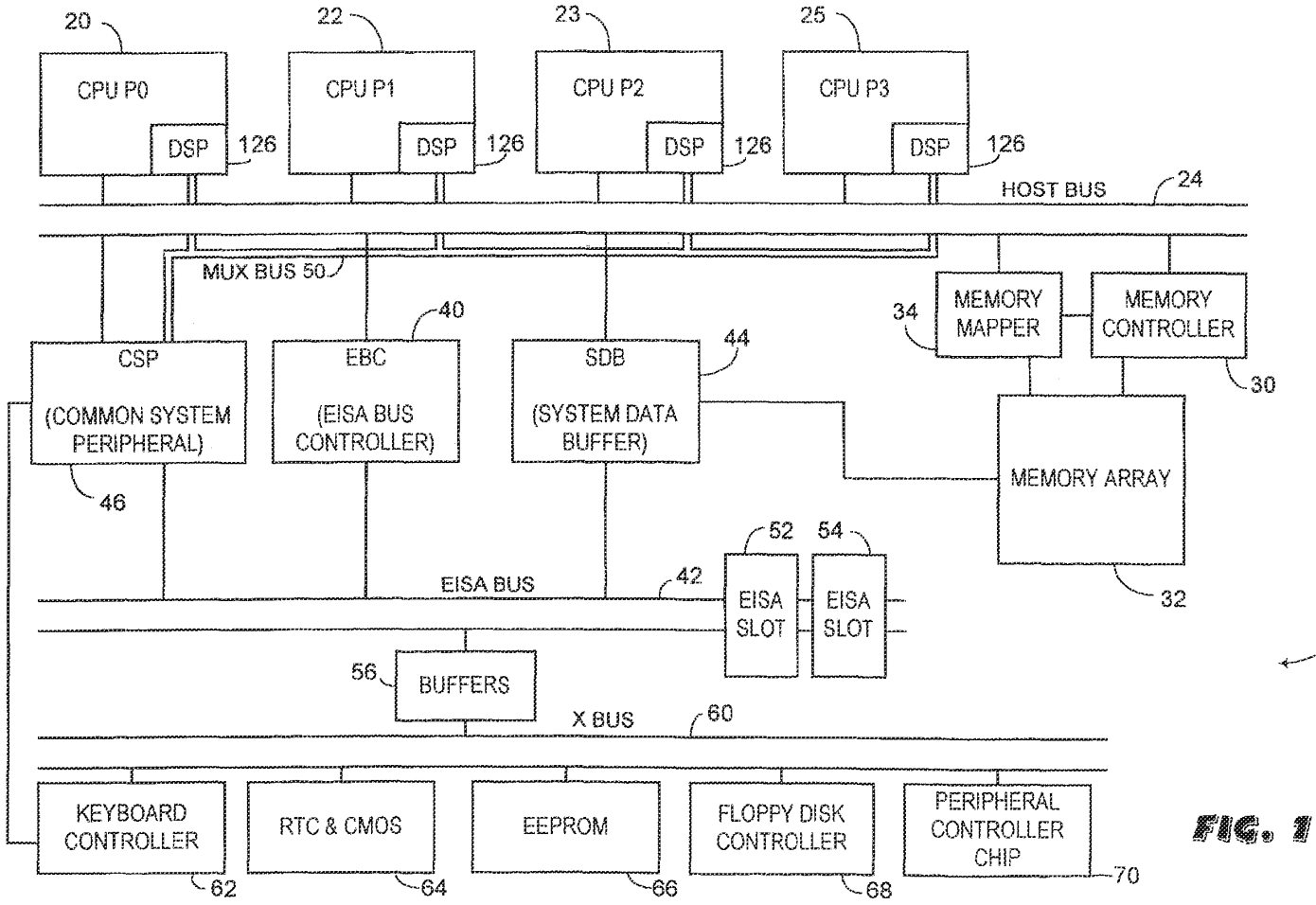

20
CPU P0
DSP
126
22
CPU P1
DSP
126
23
CPU P2
DSP
126
25
CPU P3
DSP
126
HOST BUS
24
MUX BUS 50
34
MEMORY MAPPER
MEMORY CONTROLLER
30
40
CSP
(COMMON SYSTEM PERIPHERAL)
46
EBC
(EISA BUS CONTROLLER)
SDB
(SYSTEM DATA BUFFER)
44
MEMORY ARRAY
32
52
54
EISA BUS
42
EISA SLOT
EISA SLOT
C
56
BUFFERS
X BUS
60
KEYBOARD CONTROLLER
62
RTC & CMOS
64
EEPROM
66
FLOPPY DISK CONTROLLER
68
PERIPHERAL CONTROLLER CHIP
70

FIG. 1

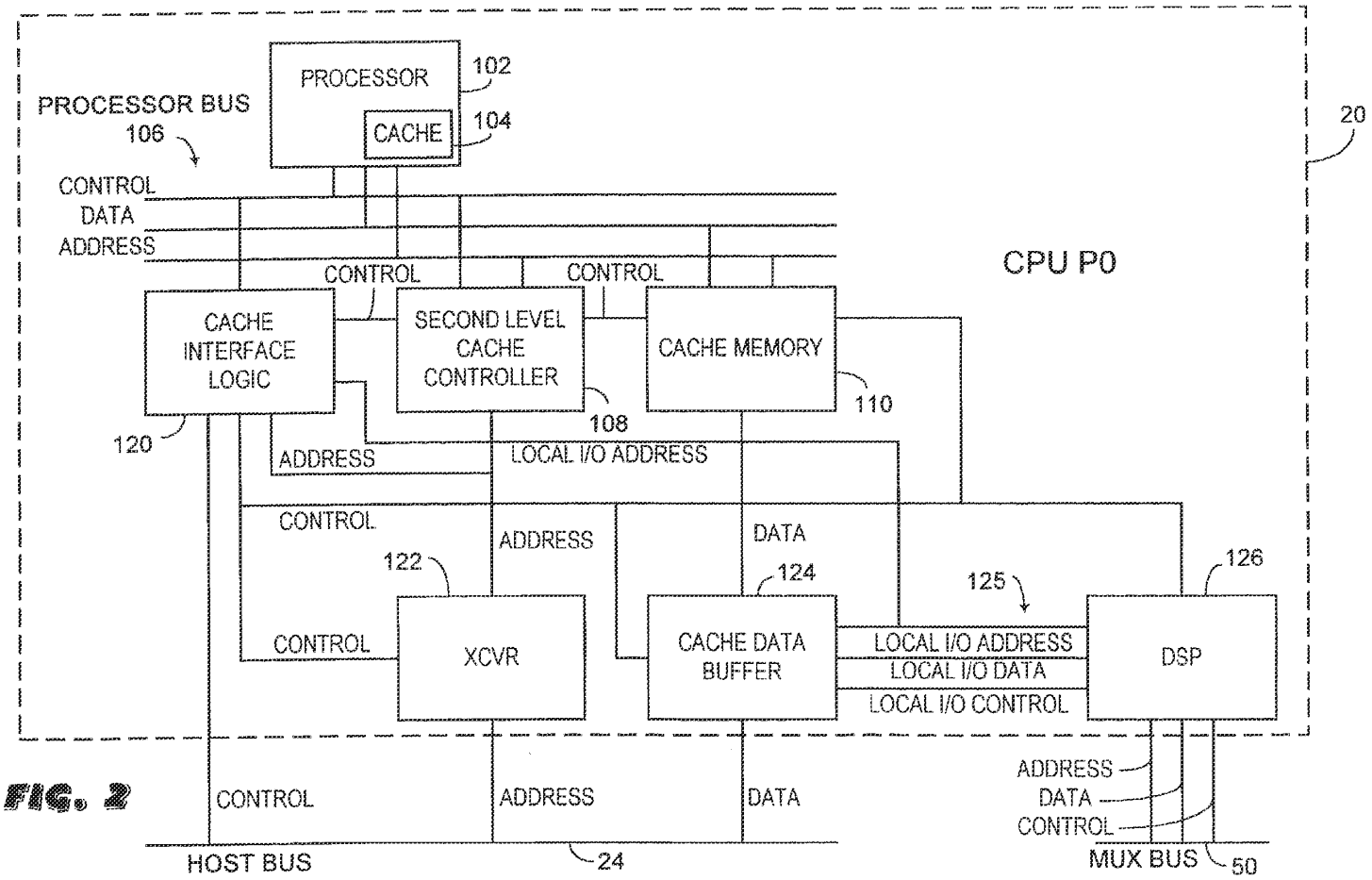

PROCESSOR
102
CACHE
104
PROCESSOR BUS
106
20
CONTROL
DATA
ADDRESS
CONTROL
CONTROL
CPU P0
CACHE INTERFACE LOGIC
SECOND LEVEL CACHE CONTROLLER
CACHE MEMORY
110
108
120
ADDRESS
LOCAL I/O ADDRESS
CONTROL
ADDRESS
DATA
122
124
125
126
CONTROL
XCVR
CACHE DATA BUFFER
LOCAL I/O ADDRESS
LOCAL I/O DATA
LOCAL I/O CONTROL
DSP
CONTROL
ADDRESS
DATA
ADDRESS
DATA
CONTROL
HOST BUS
24
MUX BUS
50

FIG. 2

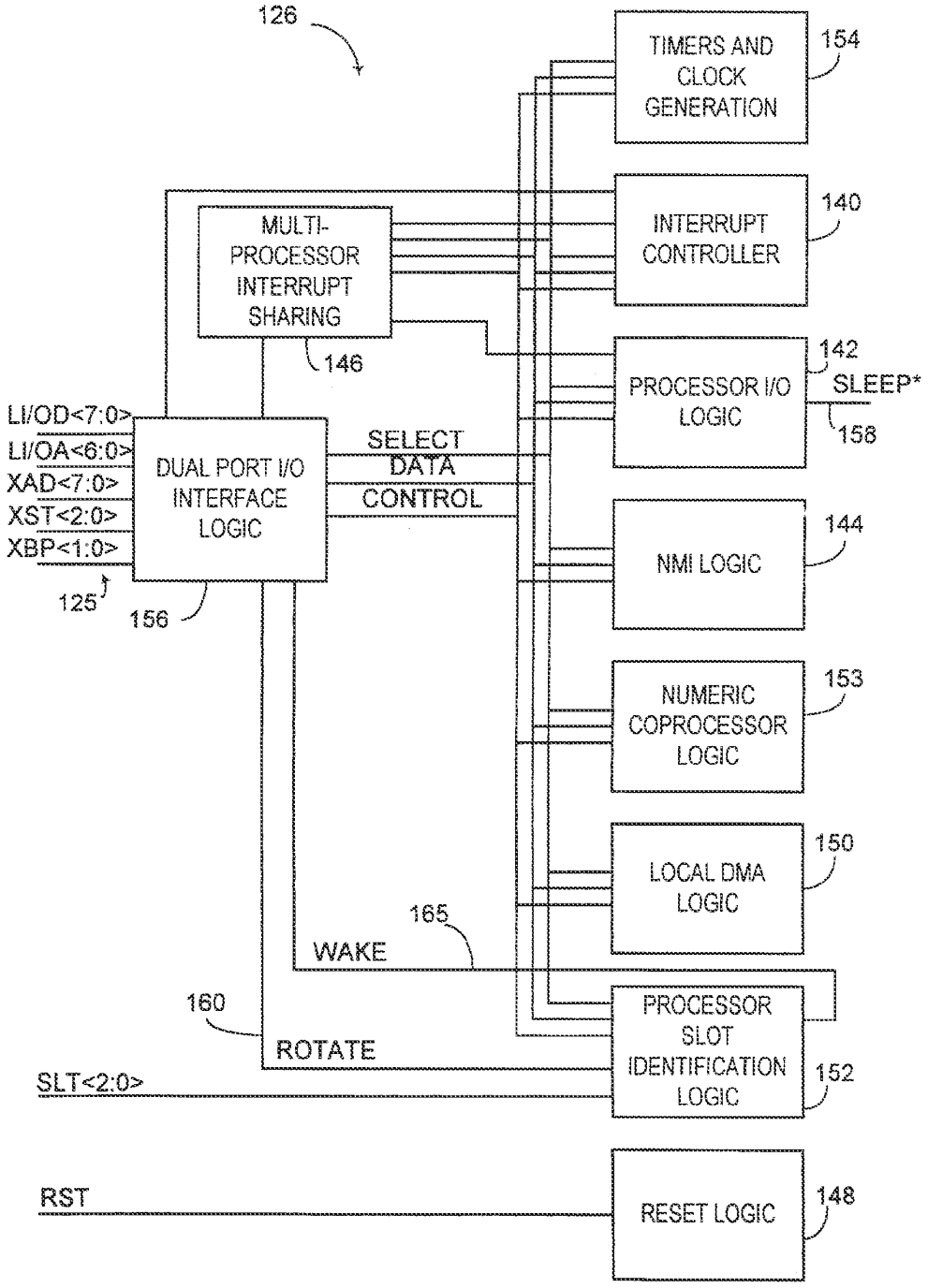

126
TIMERS AND CLOCK GENERATION
154
MULTI-PROCESSOR INTERRUPT SHARING
INTERRUPT CONTROLLER
140
146
PROCESSOR I/O LOGIC
142
SLEEP*
158
LI/OD<7:0>
LI/OA<6:0>
XAD<7:0>
XST<2:0>
XBP<1:0>
DUAL PORT I/O INTERFACE LOGIC
SELECT
DATA
CONTROL
125
156
NMI LOGIC
144
NUMERIC COPROCESSOR LOGIC
153
LOCAL DMA LOGIC
150
165
WAKE
160
ROTATE
PROCESSOR SLOT IDENTIFICATION LOGIC
152
SLT<2:0>
RST
RESET LOGIC
148

FIG. 3

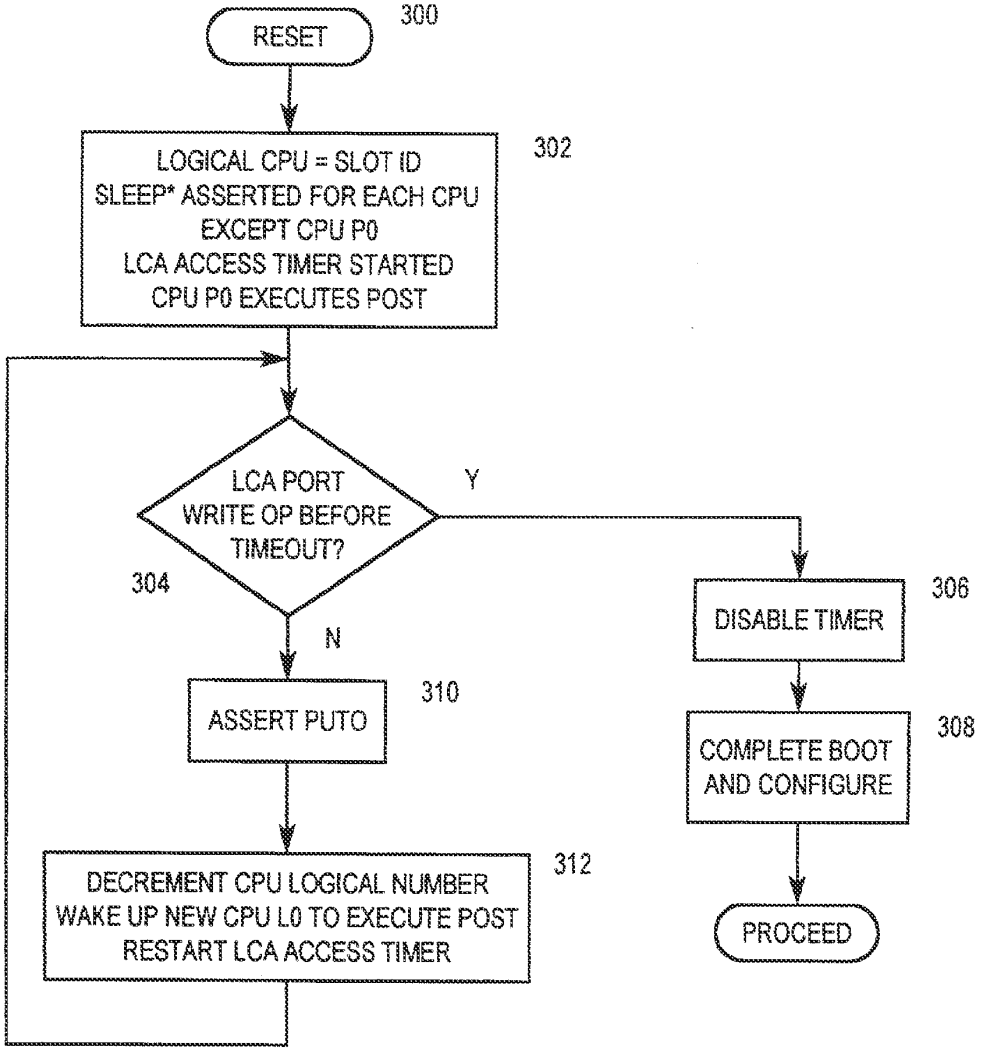

RESET
300
LOGICAL CPU = SLOT ID
SLEEP* ASSERTED FOR EACH CPU
EXCEPT CPU P0
LCA ACCESS TIMER STARTED
CPU P0 EXECUTES POST
302
LCA PORT
WRITE OP BEFORE
TIMEOUT?
304
Y
N
DISABLE TIMER
306
COMPLETE BOOT
AND CONFIGURE
308
PROCEED
ASSERT PUTO
310
DECREMENT CPU LOGICAL NUMBER
WAKE UP NEW CPU L0 TO EXECUTE POST
RESTART LCA ACCESS TIMER
312

FIG. 4

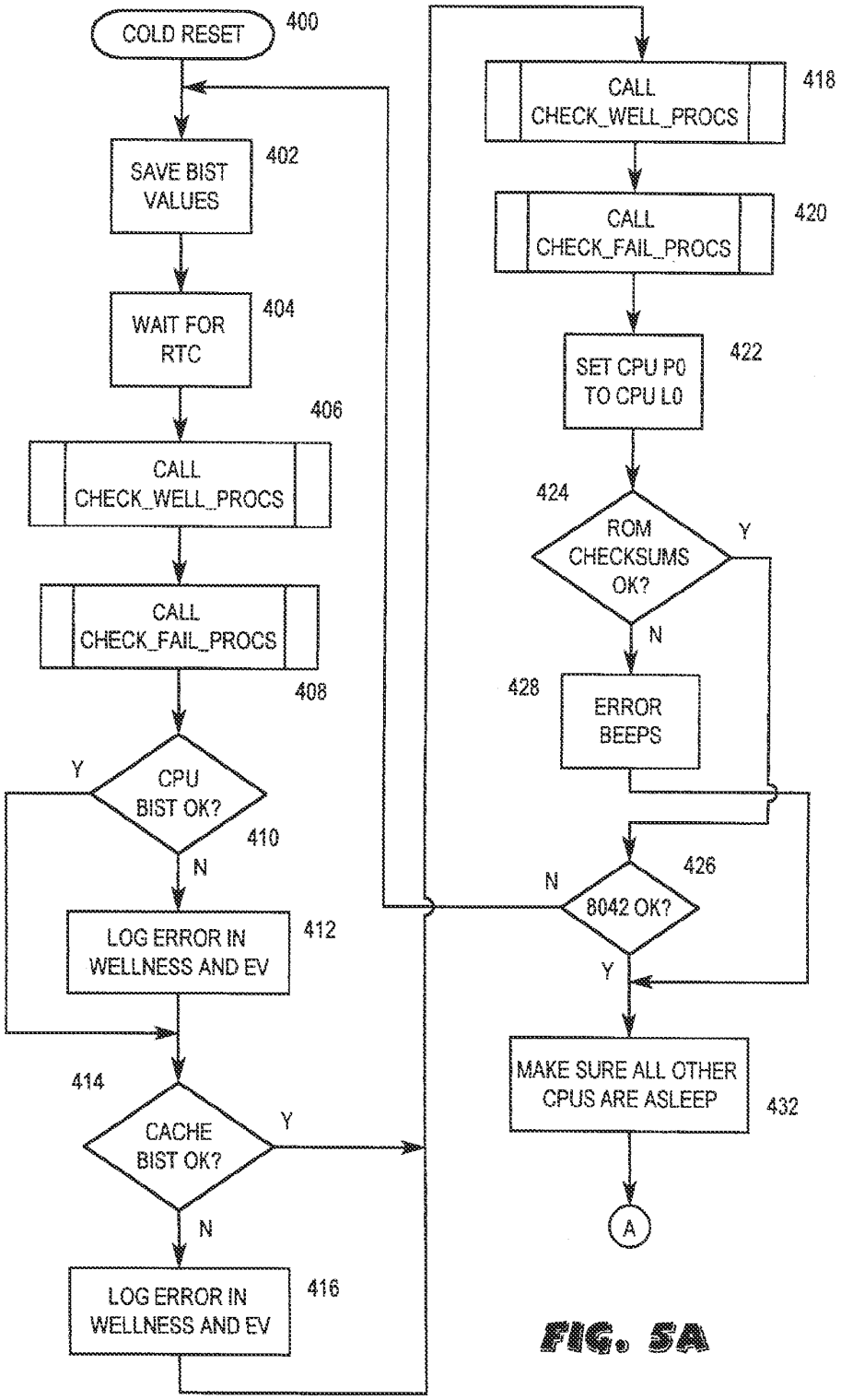

COLD RESET
400
SAVE BIST VALUES
402
WAIT FOR RTC
404
CALL CHECK_WELL_PROCS
406
CALL CHECK_FAIL_PROCS
408
CPU BIST OK?
410
Y
N
LOG ERROR IN WELLNESS AND EV
412
414
CACHE BIST OK?
Y
N
LOG ERROR IN WELLNESS AND EV
416
CALL CHECK_WELL_PROCS
418
CALL CHECK_FAIL_PROCS
420
SET CPU P0 TO CPU L0
422
424
ROM CHECKSUMS OK?
Y
N
428
ERROR BEEPS
426
N
8042 OK?
Y
MAKE SURE ALL OTHER CPUS ARE ASLEEP
432
A

FIG. 5A

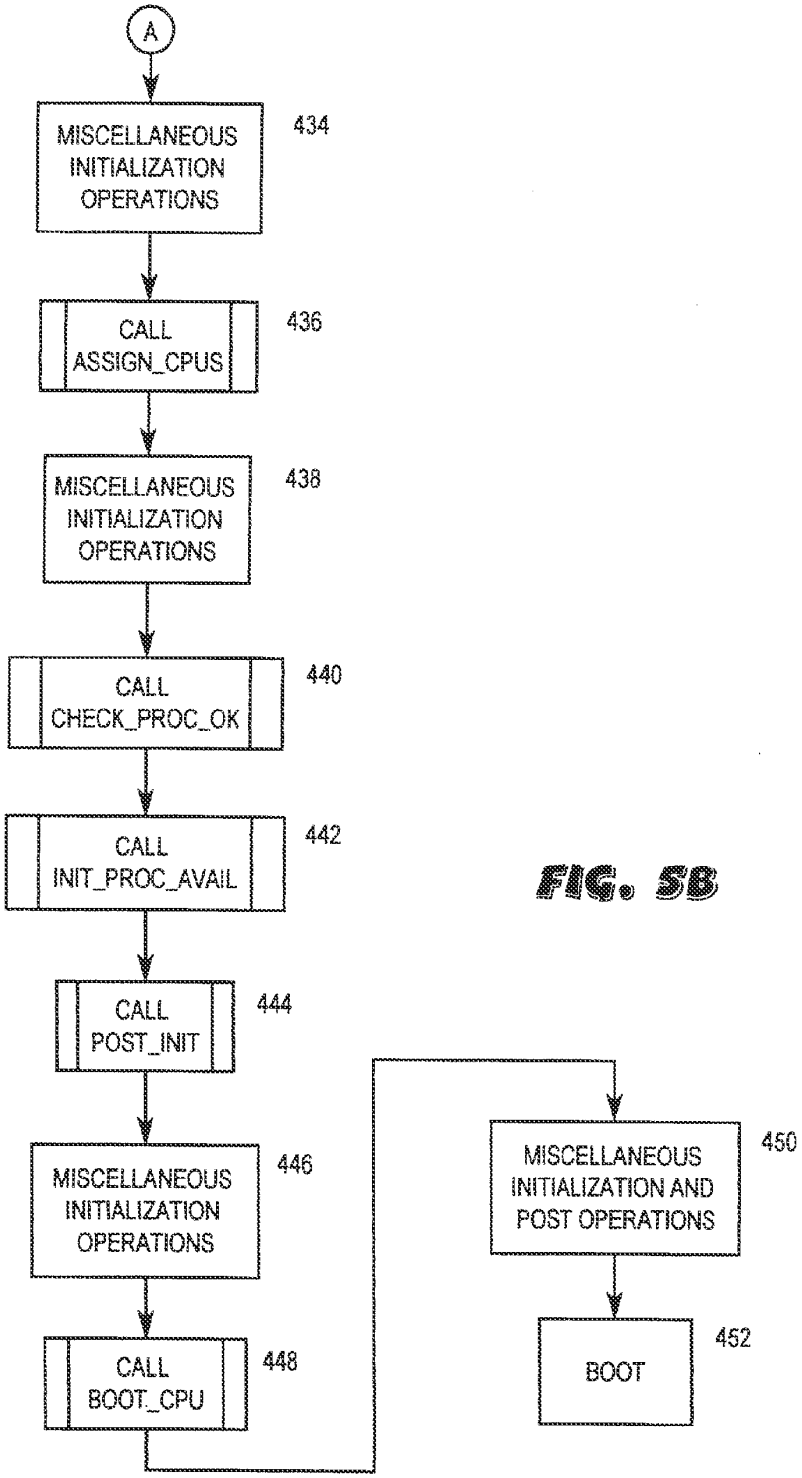

A
MISCELLANEOUS INITIALIZATION OPERATIONS
434
CALL ASSIGN_CPUS
436
MISCELLANEOUS INITIALIZATION OPERATIONS
438
CALL CHECK_PROC_OK
440
CALL INIT_PROC_AVAIL
442
CALL POST_INIT
444
MISCELLANEOUS INITIALIZATION OPERATIONS
446
CALL BOOT_CPU
448
MISCELLANEOUS INITIALIZATION AND POST OPERATIONS
450
BOOT
452

FIG. 5B

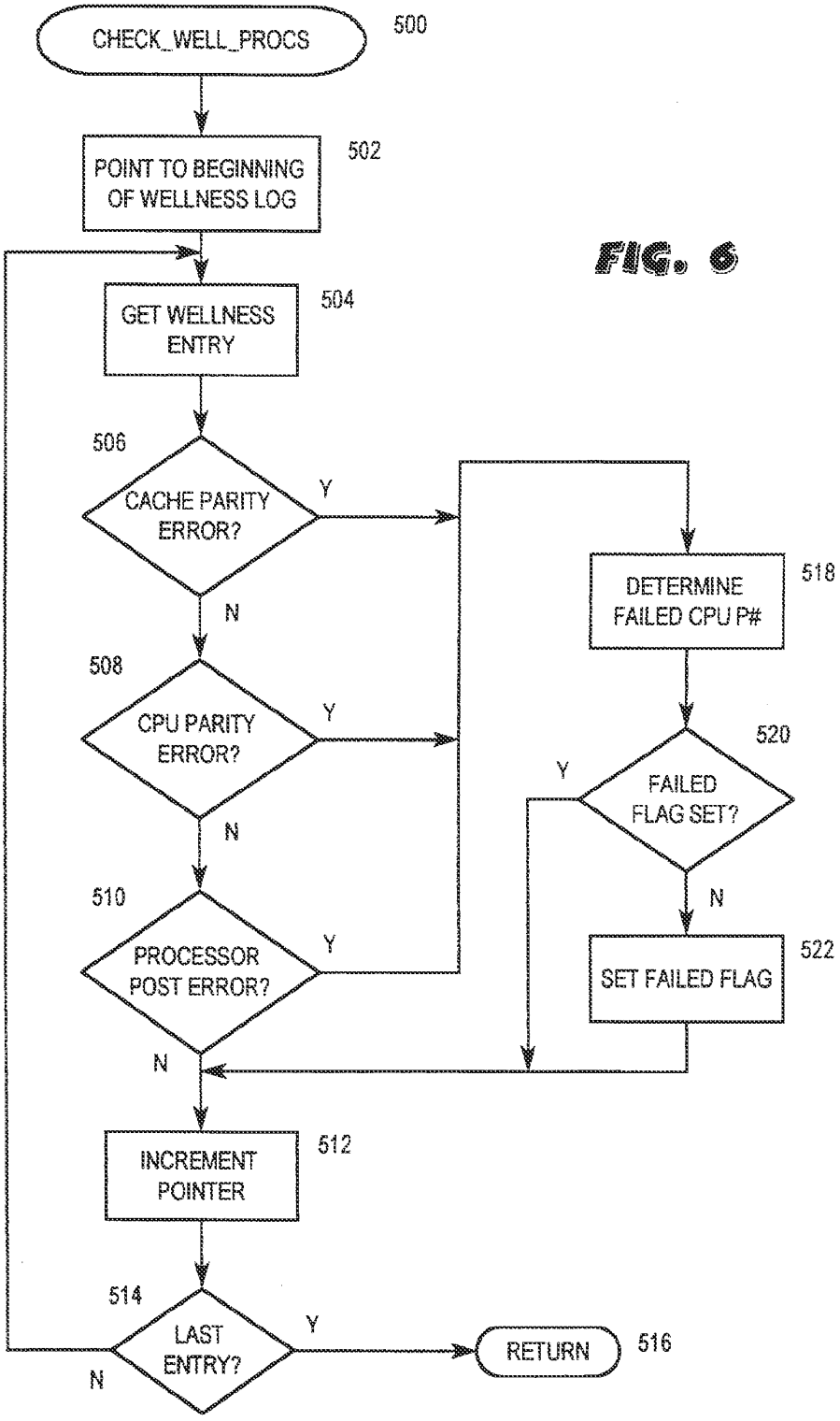

CHECK_WELL_PROCS
500
POINT TO BEGINNING OF WELLNESS LOG
502
GET WELLNESS ENTRY
504
506
CACHE PARITY ERROR?
Y
N
508
CPU PARITY ERROR?
Y
N
510
PROCESSOR POST ERROR?
Y
N
DETERMINE FAILED CPU P#
518
520
FAILED FLAG SET?
Y
N
SET FAILED FLAG
522
INCREMENT POINTER
512
514
LAST ENTRY?
Y
N
RETURN
516

FIG. 6

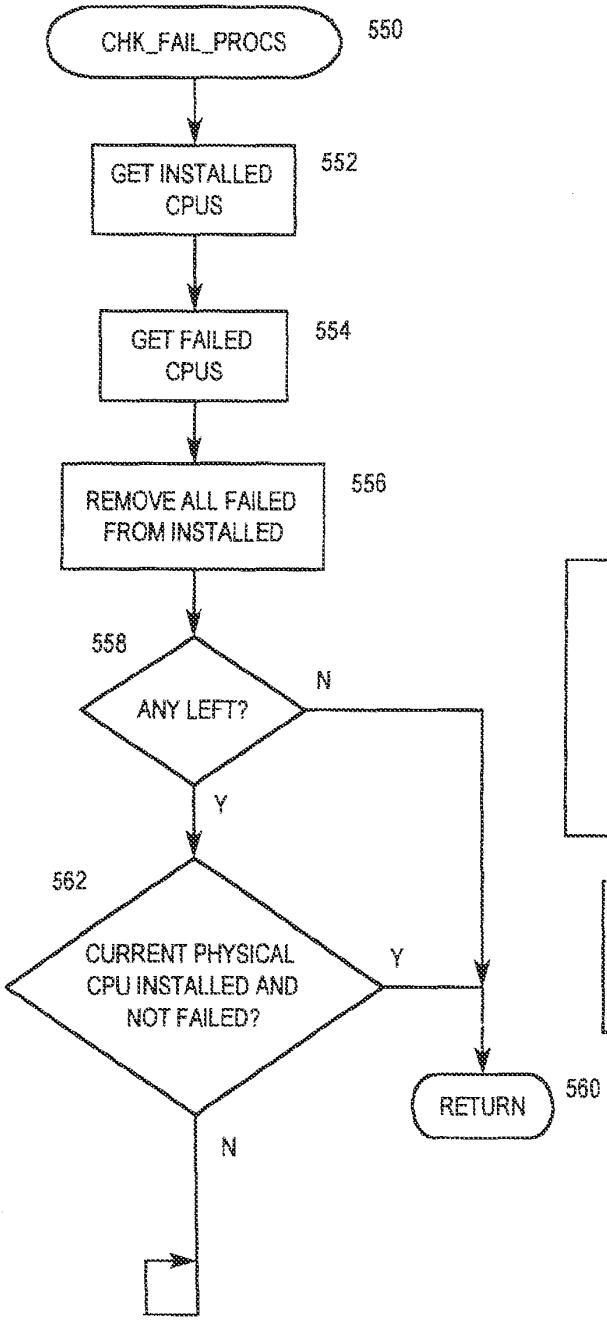

CHK_FAIL_PROCS
550
GET INSTALLED CPUS
552
GET FAILED CPUS
554
REMOVE ALL FAILED FROM INSTALLED
556
558
ANY LEFT?
N
Y
562
CURRENT PHYSICAL CPU INSTALLED AND NOT FAILED?
Y
N
RETURN
560

FIG. 7

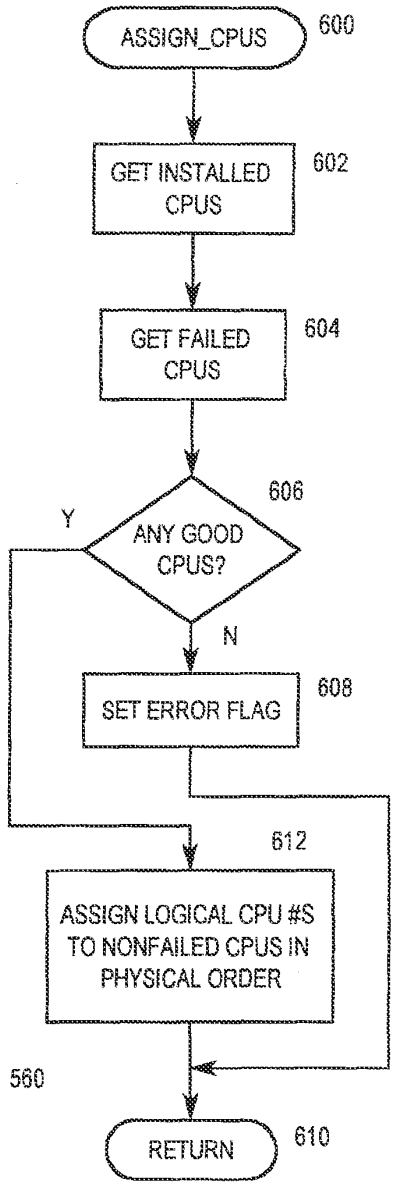

ASSIGN_CPUS
600
GET INSTALLED CPUS
602
GET FAILED CPUS
604
606
Y
ANY GOOD CPUS?
N
SET ERROR FLAG
608
612
ASSIGN LOGICAL CPU #S TO NONFAILED CPUS IN PHYSICAL ORDER
RETURN
610

FIG. 8

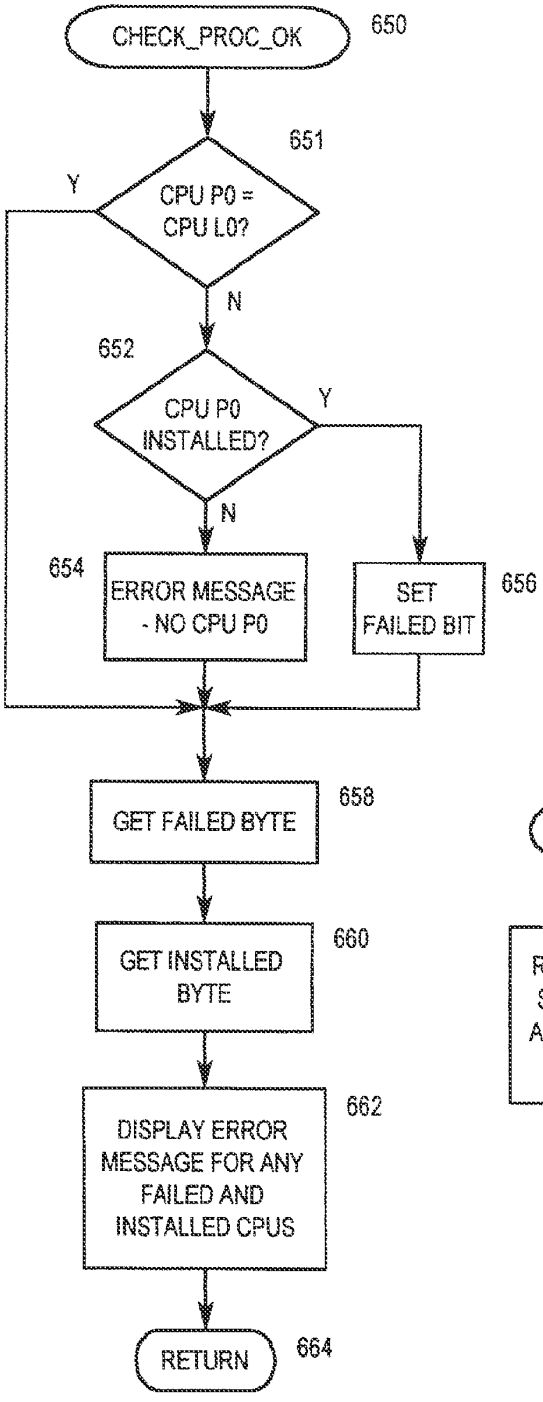

CHECK_PROC_OK
650
651
CPU P0 = CPU L0?
Y
N
652
CPU P0 INSTALLED?
Y
N
654
ERROR MESSAGE - NO CPU P0
SET FAILED BIT
656
GET FAILED BYTE
658
GET INSTALLED BYTE
660
DISPLAY ERROR MESSAGE FOR ANY FAILED AND INSTALLED CPUS
662
RETURN
664

FIG. 9

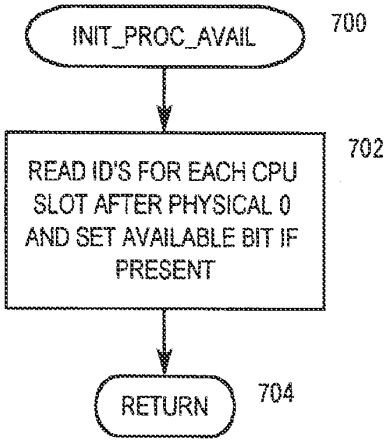

INIT_PROC_AVAIL
700
READ ID'S FOR EACH CPU SLOT AFTER PHYSICAL 0 AND SET AVAILABLE BIT IF PRESENT
702
RETURN
704

FIG. 10

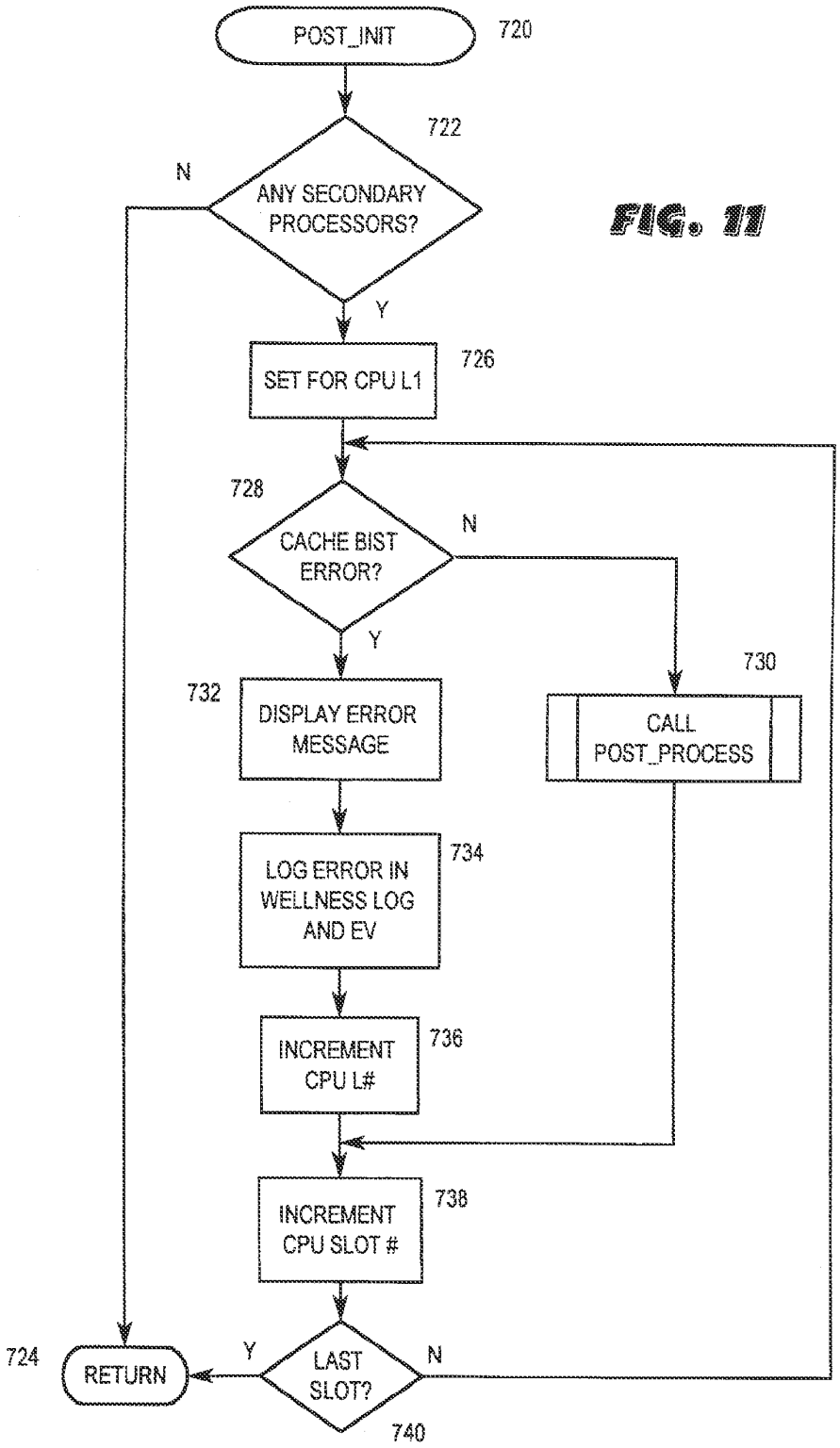

POST_INIT
720
722
ANY SECONDARY PROCESSORS?
N
Y
SET FOR CPU L1
726
728
CACHE BIST ERROR?
N
Y
730
732
DISPLAY ERROR MESSAGE
CALL POST_PROCESS
LOG ERROR IN WELLNESS LOG AND EV
734
INCREMENT CPU L#
736
INCREMENT CPU SLOT #
738
724
RETURN
Y
LAST SLOT?
N
740

FIG. 11

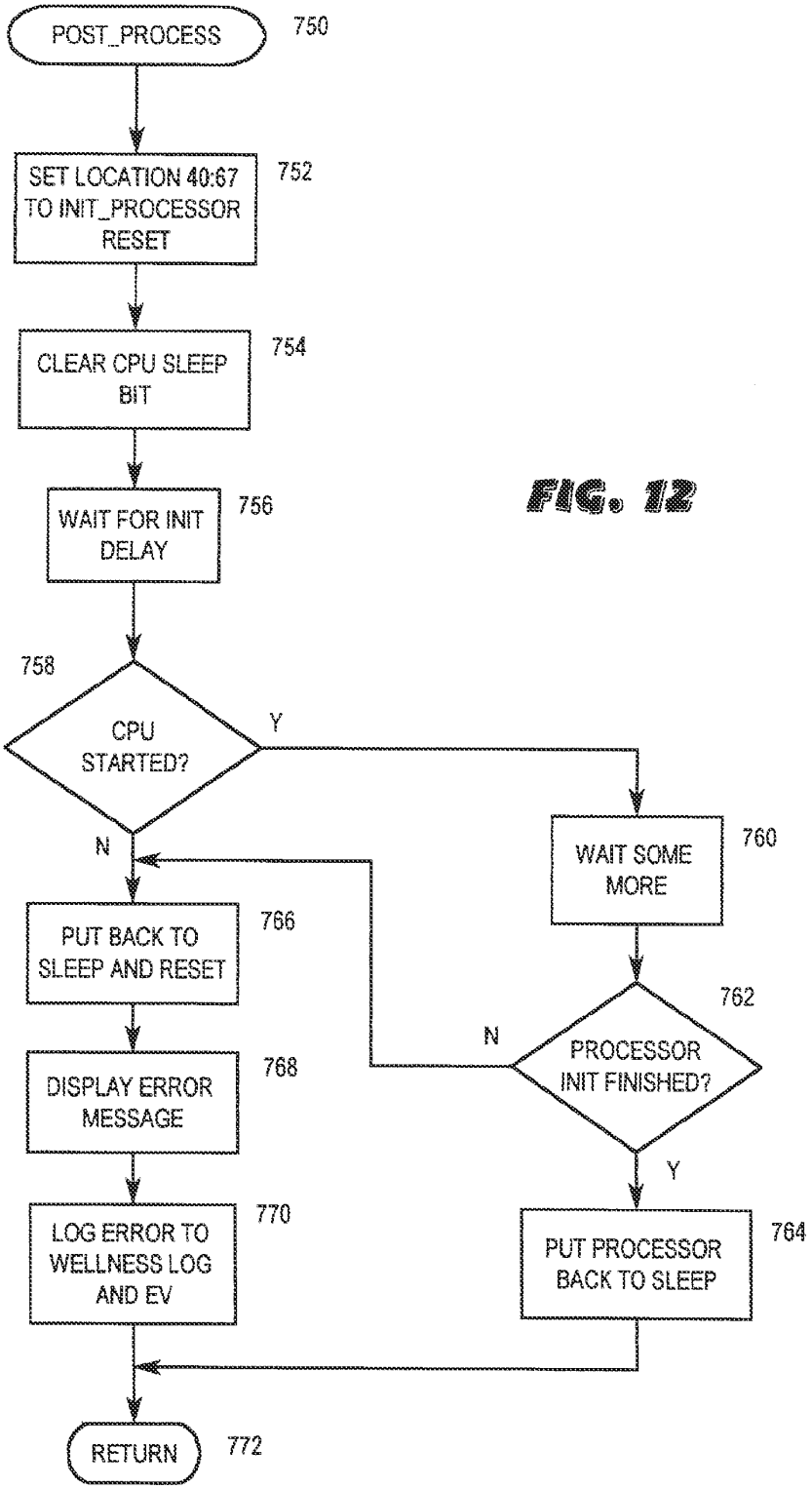

POST_PROCESS
750
SET LOCATION 40:67 TO INIT_PROCESSOR RESET
752
CLEAR CPU SLEEP BIT
754
WAIT FOR INIT DELAY
756
758
CPU STARTED?
Y
N
WAIT SOME MORE
760
762
PROCESSOR INIT FINISHED?
N
Y
PUT BACK TO SLEEP AND RESET
766
DISPLAY ERROR MESSAGE
768
LOG ERROR TO WELLNESS LOG AND EV
770
PUT PROCESSOR BACK TO SLEEP
764
RETURN
772

FIG. 12

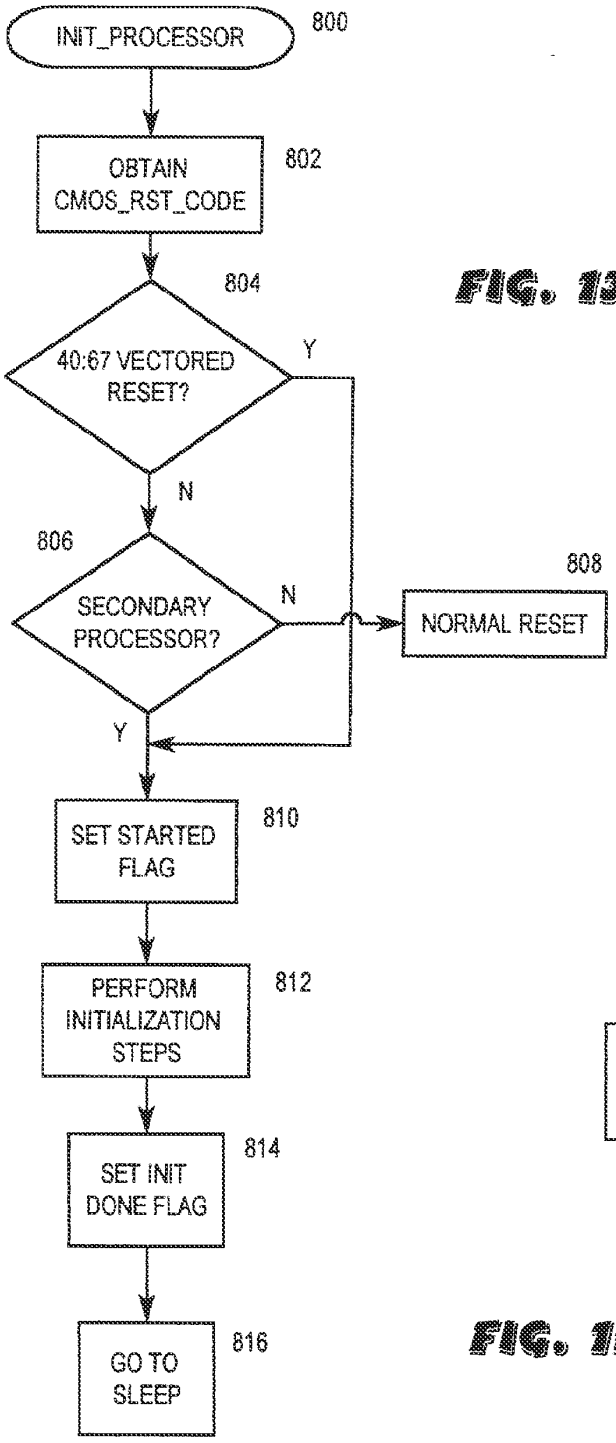

INIT_PROCESSOR
800
OBTAIN CMOS_RST_CODE
802
804
40:67 VECTORED RESET?
Y
N
806
SECONDARY PROCESSOR?
N
808
NORMAL RESET
Y
SET STARTED FLAG
810
PERFORM INITIALIZATION STEPS
812
SET INIT DONE FLAG
814
GO TO SLEEP
816

FIG. 13

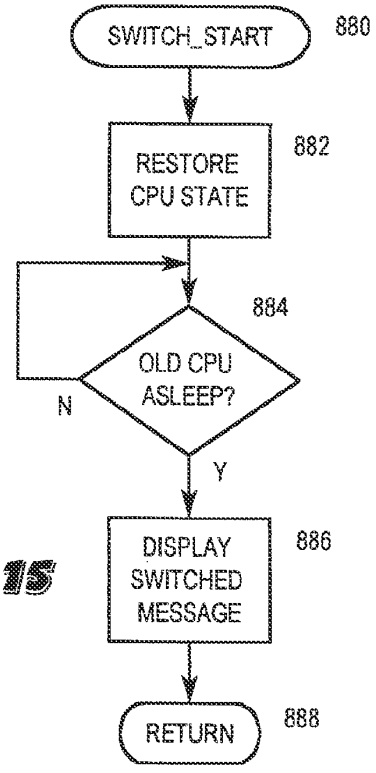

SWITCH_START
880
RESTORE CPU STATE
882
884
OLD CPU ASLEEP?
N
Y
DISPLAY SWITCHED MESSAGE
886
RETURN
888

FIG. 15

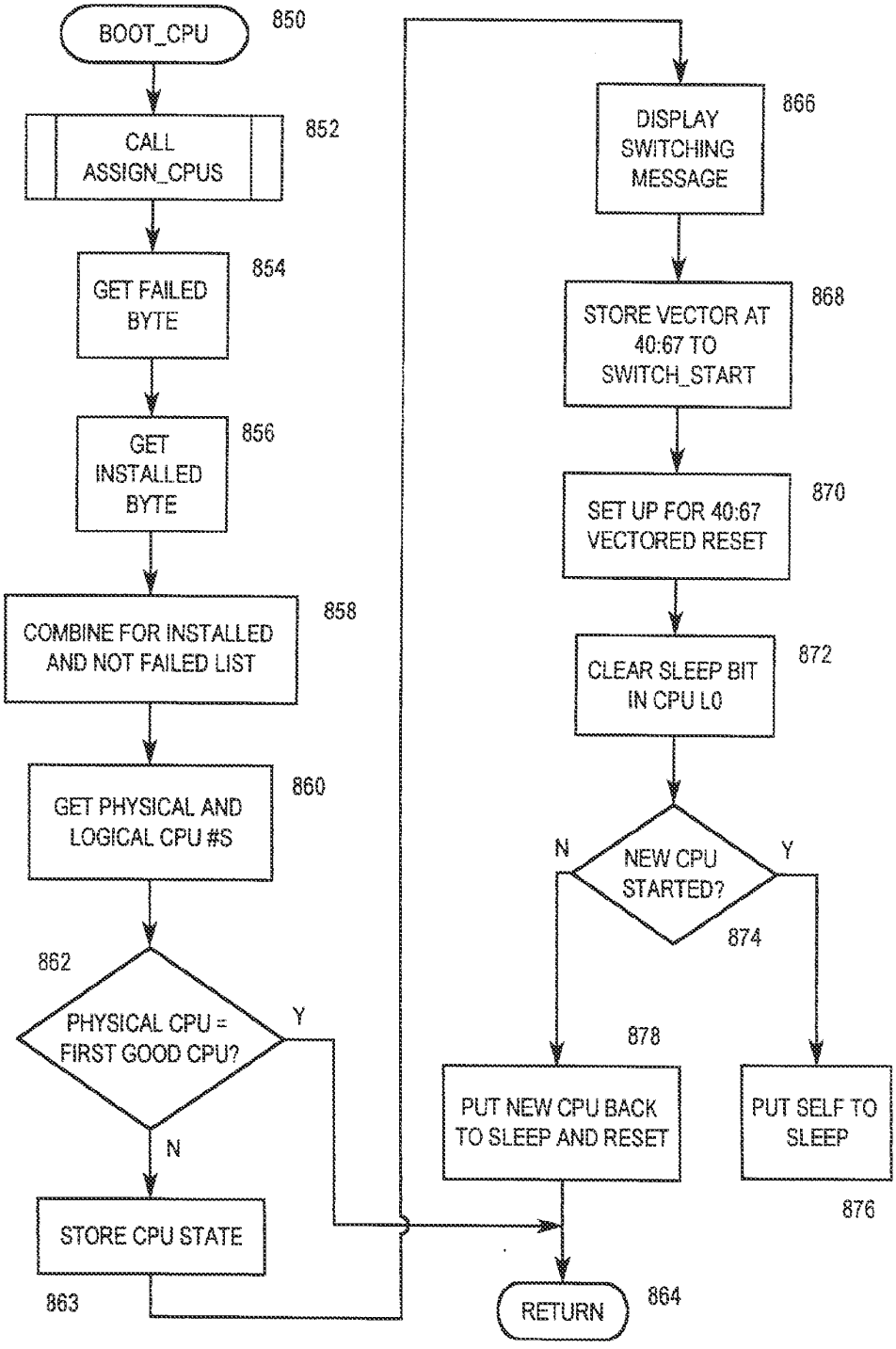

BOOT_CPU
850
CALL ASSIGN_CPUS
852
GET FAILED BYTE
854
GET INSTALLED BYTE
856
COMBINE FOR INSTALLED AND NOT FAILED LIST
858
GET PHYSICAL AND LOGICAL CPU #S
860
862
PHYSICAL CPU = FIRST GOOD CPU?
Y
N
STORE CPU STATE
863
DISPLAY SWITCHING MESSAGE
866
STORE VECTOR AT 40:67 TO SWITCH_START
868
SET UP FOR 40:67 VECTORED RESET
870
CLEAR SLEEP BIT IN CPU L0
872
NEW CPU STARTED?
N
Y
874
878
PUT NEW CPU BACK TO SLEEP AND RESET
PUT SELF TO SLEEP
876
RETURN
864

FIG. 14

METHOD OF BOOTING A MULTIPROCESSOR COMPUTER WHERE EXECUTION IS TRANSFERRING FROM A FIRST PROCESSOR TO A SECOND PROCESSOR BASED ON THE FIRST PROCESSOR HAVING HAD A CRITICAL ERROR

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to multiprocessor computer systems, and more particularly, to start-up logic for assigning logical central processing unit (CPU) designations among multiple CPUs, with the booting CPU being reassigned based on the existence of certain prior errors on that CPU.

2. Description of the Related Art

Advancements in computer technology proceed at a tremendous rate. Modern microprocessors operate at frequencies so high that processors only a few years old seem sluggish and lethargic in comparison. On the other hand, modern applications have become so complex and versatile that even the high performance delivered by modern systems is taxed to the limit. Consequently, the effort to develop yet more powerful and effective computer systems continues.

One well known method of improving computer performance is to provide multiple processors in a single system. Asymmetrical multiprocessor systems, in which one microprocessor is the master and another microprocessor performs specific functions as a slave of the master microprocessor, are common and well known. Although the master/slave relationship improves computer performance due to the division of tasks, the computer does not operate at maximum capability. This is because the slave processor performs only particularly designated operations, and thus remains idle when a task not designated for the slave processor is performed. While these operations are executed, the computer system is no more efficient than a single processor system.

The computer system's efficiency may be further enhanced by making the multiple processors symmetrical. In a symmetrical system, any processor can perform any required function. Thus, all microprocessors operate simultaneously, spending little or no idle time, and the computer system operates near its maximum efficiency. In addition, the system may be further improved by adding supplementary microprocessors as the workload increases. Adding microprocessors is particularly effective in file server systems having an array of independent functions to be performed simultaneously.

Although symmetrical multiprocessor systems are efficient, they are difficult to design. One of the many obstacles to overcome in designing a symmetrical multiprocessor system is the potential presence of a non-functional processor. A simple method of booting up a multiprocessor system is to power up one of the central processing units (CPUs), generally designated CPU**0**, and ignore the others. When the first CPU has booted up, the first CPU then turns on and tests the remaining CPUs and the various components of the system. If the first microprocessor does not function properly, however, it cannot turn on the remaining processors and the entire system is left incapacitated. Consequently, the computer owner or operator has a computer system with one or more operational CPUs, but ironically, the system is useless until the repairman arrives.

In addition, for many DOS based applications and for booting purposes, one of the CPUs must be designated as CPU**0**. CPU**0** commonly performs various functions for the system, like DRAM refresh operations, which make CPU**0** unique even in a symmetrical multiprocessor system. Thus, most multiprocessor systems require one of the microprocessors to be designated as CPU**0**. In many systems, a CPU residing in a particular physical location is always designated as CPU**0**. If one of the CPU locations is always designated as containing CPU**0**, however, the slot may be empty, or the CPU may fail, crippling the entire system.

One system, described in more detail below, addressed the problem of the CPU in the first physical slot failing by automatically rotating the CPU**0** designation to the CPU in the next physical slot if the first CPU did not perform a selected operation within a given time period. This solved the problem of the total failure of the CPU, as it was marked bad and the CPU**0** designation was rotated. However, it was not helpful in the cases of the more marginal failures, such as parity errors, which are relatively infrequent but are critical enough that the CPU cannot be considered dependable. The prior system would have simply continued until the CPU failed again, with a resulting loss of time and productivity and potential loss of information. Thus, it is desirable to have CPU**0** rotation on causes other than total CPU failure.

SUMMARY OF THE PRESENT INVENTION

The present multiprocessor computer system includes power up logic for finding a fully functioning CPU to designate as CPU**0**, particularly a CPU which has not previously had certain severe or critical errors which limit system dependability. Each microprocessor has a physical location designation which remains constant. Because the CPU designated as CPU**0** may not occupy physical location **0** in the computer, the designated CPU is referred to as logical CPU**0**, or CPU L**0**. When the system is powered up, all of the CPUs except the CPU in physical slot **0** (CPU P**0**) are initially placed in a SLEEP mode. The microprocessor in physical location **0** performs its power on self test (POST). An early part of this POST routine involves reviewing a log of errors and determining if a critical error has occurred. If so, the CPU halts operation. If there were no prior critical errors, the POST routine continues. Included in the POST routine are other tests of the CPU, both the processor itself and the associated cache memory system. If the CPU functions properly during all of these tests, the CPU is designated as CPU L**0**, and the CPU retains this designation until the system is power cycled. The CPU designated as CPU L**0** then awakens the remaining CPUs and boots up the rest of the computer system.

If CPU P**0** is not functioning properly and cannot perform even the most basic functions, it is designated as inoperative. The hardware system then awakens CPU P**1** and repeats the process of testing the CPU. If CPU P**1** is operational, then it is designated as CPU L**0**, and it boots the remainder of the system. On the other hand, if CPU P**1** also fails, it is also given an inoperative designation. The computer system then turns on CPU P**2**, and repeats the process. The process repeats until an operational microprocessor is found to perform the CPU L**0** functions. Consequently, if at least one of the microprocessors in the system is functioning, the computer system boots and operates.

If the early portion of the POST routine determines that there have been critical errors, then the halting of the CPU occurs sufficiently quickly that the rotation hardware per-

forms as described. If, however, the errors are not determined until the later tests of the CPU in the POST routine, the next operational CPU is activated, the CPU L0 designation passed to that CPU and the original CPU is noted as failed and is halted. The new CPU commences operation at approximately the point where the original CPU halted, so that completed operations of the POST routine are not repeated.

Thus, the multiprocessor system is not crippled by the failure or probable failure of a single processor, based on either total failure or certain critical intermittent failures, enhancing the dependability of the computer system.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

FIG. **1** is a block diagram of a computer system incorporating the present invention;

FIG. **2** is a block diagram of the CPU board of FIG. **1**;

FIG. **3** is a more detailed block diagram of the distributed system peripheral (DSP) of FIG. **2**;

FIG. **4** is a simplified flow chart of the power up sequence for finding an operational CPU and FIG. **5A**, **5B**, **6**, **7**, **8**, **9**, **10**, **11**, **12**, **13**, **14** and **15** are flowcharts illustrating transferring control of the computer system to an operational CPU based on the occurrence of critical errors.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to FIG. **1**, a computer system C is shown. The computer system C includes four CPU boards **20**, **22**, **23**, **25** connected to a host bus **24** and a memory controller **30**, which is also coupled to the host bus **24**. Although four CPUs **20**, **22**, **23**, **25** are used in the preferred computer system C, the present system functions for any multiprocessor system. Each CPU occupies a physical slot in the computer system by which the CPUs **20**, **22**, **23**, **25** are designated. Thus, CPU P0 **20** is the CPU that occupies physical slot **0** in the system. The CPUs are designated CPU P0 **20**, CPU P1 **22**, CPU P2 **23**, and CPU P3 **25**, in reference to each CPU's physical location.

A main memory array **32**, preferably comprised of dynamic random access memory (DRAM) modules, is coupled to the memory controller **30**. Memory mapper logic **34** is coupled to the host bus **24**, the memory controller **30**, and the memory array **32**. The memory mapper logic **34** provides memory mapping functions to facilitate memory accesses to the memory array **32**.

A bus controller **40**, preferably an EISA bus controller (EBC), is coupled between the host bus **24** and an expansion bus **42**, preferably an Extended Industry Standard Architecture (EISA) bus. The EBC **40** provides various bus cycle translation and conversion functions to facilitate transfers between the host bus **24** and the EISA bus **42**. A system data buffer (SDB) **44** is also coupled between the host bus **24**, the memory array **32** and the EISA bus **42**. A logic block referred to as the common system peripheral (CSP) **46** is coupled between the host bus **24** and the EISA bus **42**. The CSP **46** is also coupled through a MUX bus **50** to logic blocks referred to as distributed system peripherals (DSPs) **126** in each of the CPUs **20**, **22**, **23**, **25**. The CSP **46** includes various system components including a direct memory access (DMA) controller, an EISA arbitration controller, and numerous system board logic functions like memory refresh control and interrupt logic.

The EISA bus **42** includes a plurality of EISA slots **52** and **54** for receiving EISA expansion cards, like a network interface card or a hard disk interface card. The EISA bus **42** is coupled through buffers **56** to a bus referred to as the X bus **60**. A number of peripheral devices are coupled to the X bus **60** including the keyboard controller **62**, a real time clock (RTC) and CMOS memory **64**, an electrically erasable programmable read only memory (EEPROM) **66** used to store the BIOS and system functions, a floppy disk controller **68**, and a peripheral controller chip **70**, which includes numerous ports and UARTs (universally asynchronous receiver/transmitters).

The MUX bus **50** between the CSP **46** and the various DSPs **126** associated with the CPUs **20**, **22**, **23**, **25** includes an XAD bus. The XAD bus is an 8-bit bi-directional data bus on which interrupt, I/O, and special cycle data is transferred. The MUX bus **50** further includes a cycle status bus called the XST bus. This three-bit bus is controlled by the CSP **46** and defines the cycle currently being transferred on the 8-bit MUX bus **50**. The MUX bus **50** further includes a two-bit XBP bus, which is an input bus on which DMA byte pointer information is received from the DSPs **126**. Using the XAD, XST, and XBP buses, the CSP **46** interfaces with the DSPs **126** to provide interaction between the general computer system C and the CPUs **20**, **22**, **23**, **25**. The MUX bus **50** is a broadcast bus that transmits all signals to all of the CPUs **20**, **22**, **23**, **25**, and each CPU **20**, **22**, **23**, **25** determines whether it is the CPU being accessed.

For more information on the operation of the MUX bus **50**, please see related copending application Ser. No. 07/955,482 entitled MULTIPLEXED COMMUNICATION PROTOCOL BETWEEN CENTRAL AND DISTRIBUTED PERIPHERALS IN MULTIPROCESSOR COMPUTER SYSTEMS, filed on Oct. 2, 1992, which is hereby incorporated by reference. For more information regarding the CSP **46** and the DSPs **126**, please see related copending application Ser. No. 07/955,683, entitled ARRANGEMENT OF DMA, INTERRUPT AND TIMER FUNCTIONS TO IMPLEMENT SYMMETRICAL PROCESSING IN A MULTIPROCESSOR COMPUTER SYSTEM, filed on Oct. 2, 1992, which is hereby incorporated by reference.

The CSP **46** provides an interface between the MUX bus **50** and the EISA bus **42** and X bus **60** of the computer system C. Various transfers and operations are defined on the MUX bus **50** using the XST bus to define the transfer and the XAD bus to identify addresses, data, and interrupts. The CSP **46** implements a MUX bus master interface to assert interrupts and transfer data to and from the DSPs **126**. The CSP **46** in the present system also includes power up timer logic. When the system is powered up, a power up timer begins counting. If CPU **20** in physical location zero does not write to a particular DSP port accessed via the CSP **46** within a specific interval, which is two seconds in the preferred embodiment, the CSP **46** asserts a power up timeout (PUTO) signal to all of the DSPs **126**. The PUTO signal is generated as a miscellaneous data transfer on the MUX bus **50**. As described below, the PUTO signal causes the current CPU to be designated as inoperative and initiates testing of the next CPU.

Referring now to FIG. **2**, a block diagram of CPU P0 **20** is shown. For the purposes of this application, all of the CPUs **20**, **22**, **23**, **25** are substantially identical. In the following description, CPU P0 **20** is described for simplic-

ity, and the following description applies equally well to the other CPUs **22**, **23**, **25**. CPU P0 **20** includes a processor **102** which preferably includes an internal cache **104**. The processor **102** is preferably the Intel i**486** processor preferably running at 33/66 MHz or Pentium processor preferably running at 66 MHz. However, the use of other types of processors is also contemplated. The processor **102** is coupled to a processor bus **106** including control, data and address portions, as shown.

The CPU P0 **20** further has a cache memory system which includes an array of cache RAMs **110**, a cache controller **108**, and various cache interface logic **120** and buffers **124** to interface between the processor bus **106** and the CPU components. In the preferred embodiment, the cache RAMs **110** are C8 or 82490 cache RAMs produced by Intel Corporation, and the cache controller **108** is a C5 or 82495 cache controller, also produced by Intel. The cache controller is also connected to an address transceiver **122** for transmitting addresses to and from the host bus **24**. The transceiver **122** receives control signals from the cache interface logic **120**.

The cache data buffer is linked not only to the host bus, but is also connected by a local I/O bus **125** to the DSP **126**. The DSP **126** implements various logic functions that are closely related to the processor/cache system. Referring now to FIG. **3**, the DSP **126** includes interrupt controller logic **140**, preferably comprising two cascaded Intel 8259 interrupt controllers which provide 15 levels of interrupts. The DSP **126** also includes non-maskable interrupt (NMI) logic **144** and multiprocessor interrupt sharing logic **146**. The multiprocessor interrupt sharing logic **146** allows a CPU to interrupt other CPUs at any interrupt level in one I/O access, thus providing an efficient multiprocessor communication capability. Also included are reset logic **148**, DMA (direct memory access) logic **150**, numeric coprocessor logic **153**, timer clock generation logic **154**, and dual port I/O interface logic **156**. The dual port I/O interface logic **156** interfaces the various registers and logic in the DSP **126** with the local I/O bus **125** and the MUX bus **50**.

CPU processor I/O logic **142** on the DSP includes a processor control port, and processor slot identification logic **152** includes a CPU Identification (WHOAMI) port, an Index CPU port, and a Logical CPU Assignment (LCA) port. The processor control port controls miscellaneous processor functions, like cache enabling and flushing and processor interrupt signals. In particular, the processor control port enables and disables the SLEEP, signal **158** to the microprocessor **102**. When the system is power cycled, the SLEEP* signal **158** is disabled until the processor control port is written and the SLEEP, signal **158** is enabled. Any device, including the microprocessor **102** itself, may write to a CPU's processor control port to change the sleep bit in the port, which causes the SLEEP* signal **158** to be driven low by the DSP **126**. The SLEEP* signal **158** is asserted low as long as the value of the sleep bit in the processor control port remains **1**.

The SLEEP* signal **158** is provided to the cache interface logic **120**. In the preferred embodiment, the SLEEP* signal **158** causes a hold request to be asserted to the microprocessor **102**. When the hold request is acknowledged, the internal **486** first level cache **104** and the external second level C**5** cache **110** are flushed. After the flush is completed, the hold request signal remains asserted so that the microprocessor **102** stays inactive. The hold request signal is maintained as long as the SLEEP* signal **158** is asserted. Thus, to activate the microprocessor **102**, the processor control port of the DSP **126** is again written to disable the sleep state, which causes the SLEEP* and the microprocessor hold request signals to be negated, and the microprocessor **102** resumes operation.

Processor slot identification logic **152** consists of the logic required for physical CPU identification and logical CPU assignment. The logical CPU assignment is an arbitrary designation of a particular CPU, and each CPU may be addressed by its physical location or logical assignment using the MUX bus **50**. One of the ports in the processor slot identification logic **152** is the Index CPU port, which is used to select the physical CPU to access when the LCA port is accessed, or to select the logical CPU to access when other local ports are accessed from the EISA bus. Local ports can be accessed directly or via an Index Address port in the CSP. The Index CPU port must be written before being used to assure its contents. If the indicated CPU matches the assignment indicated by the DSP's **126** WHOAMI port, the CPU performs the local operation. Otherwise, the operation from the MUX bus **50** is ignored.

Each DSP **126** includes a Logical CPU Assignment (LCA) port. The LCA port is used by the software, as discussed below, to assign logical CPU designations to physical CPU locations. It is also used to determine the physical location of a logical CPU. Each CPU's LCA port is accessible to all elements of the system C, but is not locally accessible but is accessible only over the EISA bus **42** through the CSP **46**. The port contains CPU assignment bits used to assign the logical CPU number to the physical CPU location. The LCA port works in conjunction with the Index CPU port to access a particular physical CPU. Before the LCA port is written, the Index CPU port must be written with the desired CPU's physical location. When the LCA port is written on the next operation, the CPU having the indicated physical designation, instead of logical assignment, is accessed.

The WHOAMI port stores the logical CPU designation for the CPU and may only be read via the local bus **125**. When accessed, the WHOAMI port responds with a number indicating the individual CPU's logical designation. The CPU's logical designation is determined when the system C is powered up, as discussed below. When the system C is power cycled, the WHOAMI port initially receives a value corresponding to the CPU's physical location. By accessing this port, each CPU determines its own logical CPU assignment, and therefore determines whether it is the particular CPU being addressed on the MUX bus **50**.

Each CPU also receives a separate set of three SLT signals **161**. The SLT signals **161** are permanently connected to particular signals to correspond to individual slots. For example, the SLT signals **161** for physical CPU slot **0** always assert **000**, while those for slots **1** and **2** always assert **001** and **010**, respectively. The SLT signals **161** are provided to the processor slot identification logic **152** on the DSP **126** so that each CPU may determine its physical slot location.

The processor slot identification logic **152** further receives a ROTATE signal **160** generated by the dual port I/O interface logic **156**. The interface logic **156** asserts the ROTATE signal **160** when the PUTO signal is asserted on the MUX bus **50**. When the ROTATE signal **160** is asserted, the processor slot identification logic **152** decrements the CPU's logical assignment in the WHOAMI port. If the CPU's logical CPU assignment is **0** when the ROTATE signal is asserted, the WHOAMI port value is set to FFh, indicating a non-functioning CPU. If the CPU's logical assignment is FFh when the ROTATE signal **160** is asserted, the ROTATE signal is ignored.

If the CPU's logical assignment rotates to **0**, the processor slot identification logic **152** asserts a WAKE signal **165** to the interface logic **120**. The WAKE signal **165** causes the interface logic **120** to change the processor control port's sleep bit in the processor I/O logic **142**. As a result, the SLEEP* signal **158** is negated, and the CPU begins operation.

Referring now to FIG. **4**, the logical configuration of the CPUs is established after the computer system is power cycled. When the system is powered up, the computer system C resets in step **300** and, in step **302**, the DSP inputs the SLT signals to the WHOAMI port, so that each CPU receives an initial logical designation corresponding to its physical location. At the same time, each DSP asserts the SLEEP* signal on its CPU except for CPU **P0**, which remains awake and begins its power-on self test (POST) routine. In an alternative embodiment, the DSP for CPU **P0 20** may also enable its sleep bit, but CPU **P0 20** is then immediately awakened. When the SLEEP* signal is asserted for each CPU, all of the CPUs **22**, **23**, **25**, except for CPU **P0 20**, become inactive. Meanwhile, the power up timer in the CSP **46** loads a preset interval on reset and begins counting when the reset signal is negated. In the preferred embodiment, the preset interval is approximately **2** seconds. The power up timer counts until it is disabled or times out.

The timer may be disabled by a write operation to the LCA port. In step **304**, the CSP **46** determines whether a write operation to the LCA port has occurred before the timer times out. If CPU PO **20** is operating properly, the POST program directs the CPU **P0 20** to execute a write operation to the LCA port. The write operation to the LCA port is detected by the CSP **46**, which disables the timer in step **306**.

Next, in step **308**, the operational CPU, now designated CPU **L0**, completes its POST program and boots up the rest of the system C using standard boot software. In particular, CPU **L0** wakes up and boots the remaining CPUs **22**, **23**, **25**, with each CPU executing software to test itself and to assign an appropriate logical CPU designation. This process can conveniently be done by placing a special vector, which is interrogated early in the POST program, to special wake up code, which includes a handshake to CPU **L0**, so that each CPU can be brought up individually and indicate its status. A more detailed description is provided below. For more details on the basic operation of the startup and handshake procedure, please refer to application Ser. No. 07/431,743, originally entitled "COMMON RESET ROM", filed on Nov. 3, 1989, and continued as Ser. No. 08/051,601, filed on Apr. 22, 1993 and retitled "METHOD AND APPARATUS FOR RESETTING MULTIPLE PROCESSORS USING A COMMON ROM" which is hereby incorporated by reference. If one of the CPUs **22**, **23**, **25** is not functional as indicated by a failure to handshake within a given time, it is so designated in its DSP and the next consecutive functioning CPU is designated with the next logical CPU assignment. If the slot is empty, the logical CPU assignment sequence continues at the next slot occupied by an operational CPU. In addition to booting up the other CPUs, the designated CPU **L0** boots up the rest of the computer system C, and operates as CPU **L0** for all unique CPU **L0** functions. As noted, this sequence is described in more detail below, particularly when intermittent critical errors have occurred on the first CPU.

If the write operation to the LCA port does not occur before the timer times out, the CSP **46** generates the PUTO signal in step **310** and it is transferred onto the MUX bus **50**. When the PUTO signal is received by the DSPs **126**, each DSP asserts the ROTATE signal **160** in step **312** and the logical assignment in the WHOAMI port of each CPU is decremented by 1. The current CPU **L0** that failed to write the LCA port within the timer interval is assigned a value in its DSP **126** of FFh, indicating that the CPU is inoperative. A CPU having a logical assignment of FFh does not have its slot identification decremented when the power up timeout signal is asserted.

After the logical CPU assignments have been rotated, the new CPU **L0** is awakened by the WAKE signal and begins the same POST code described before, which includes the write to the LCA port. The power up timer is restarted, and the process returns to step **304**. The procedure repeats until an operational CPU writes to the LCA port before the timer times out and boots the rest of the system. If none of the CPUs are operational, the system will not boot up.

Referring now to FIG. **5A**, a more detailed version of the RESET sequence is shown. In this case the COLD RESET sequence **400** commences at step **402** where the BIST or basic internal self test value provided by the processor of CPU **P0 20** is stored for later examination. Control proceeds to step **404** which is a wait period until the RTC **64** becomes fully active. Control then proceeds to step **406**, which is a call to the CHECK_WELL_PROCS sequence **500** (FIG. **6**) to determine and update the failed flags. Preferably a series of flag bytes is maintained in the CMOS portion of the RTC **64** to store certain variables. One byte is used to indicate which processors are present, another to indicate which have failed. Control then proceeds to step **408**, which is a call to the CHK_FAIL_PROCS sequence **550** (FIG. **7**) which actually checks to see if any failed processors are present and if the current processor is a failed processor. Assuming that the current processor is not a failed processor, control proceeds to step **410** to determine if the CPU BIST value as stored in step **402** indicates that there are no problems. If there are problems, control proceeds to step **412** where the specific error is indicated, is logged in the wellness log and in environment variables. The wellness log is a running log contained preferably in the extended CMOS are contained in the RTC **64** or in another nonvolatile RAM source in the computer system C which keeps track of all of the errors which have occurred in the computer system C.

In most cases during run time operations, these errors generate an NMI or nonmaskable interrupt, with the NMI sequence then logging the error in the wellness log so that a maintenance record can be maintained. The environment variables are also areas in the extended CMOS area and which are used to set flags and pass information as to the state of the computer or for configuration values. In this case the error is just logged in an environment variable that indicates an error has occurred. The wellness log is used for more complete information. For more details, please refer to Ser. No. 07/955,849, entitled "COMPUTER FAILURE RECOVERY AND ALERT SYSTEM", filed on Oct. 2, 1992 and hereby incorporated by reference.

Control proceeds to step **414** after step **412** or if the CPU was okay in step **410**. In step **414** the cache controller system **104** present on the CPU is checked to determine if its basic internal self test value was okay. If not, control proceeds to step **416** where the error is logged in the wellness log and in the environment variable. If it was okay in step **414** or after **416**, control proceeds to step **418** where the CHECK_WELL_PROCS sequence **500** is again called. This call is to reupdate the failed status based on the errors which may have been determined in steps **410** and **414**. Control then proceeds to step **420**, where the CHK_FAIL_PROCS sequence **550** is called to shutdown a processor if necessary.

Control proceeds to step **422** where physical CPU**0** is set to indicate to be a logical CPU**0**. This is done by a write operation to the LCA port as described above. This operation will have the effect of disabling the timer so that automatic, hardware-based CPU L**0** rotation does not occur.

Control then proceeds to step **424** to determine if the checksum in the EEPROM **66** is okay. This is a check of the integrity of the EEPROM **66**. If it checks okay, control proceeds to step **426** to determine if the 8042 keyboard controller **62** is okay. If not, control returns to step **402** and operations repeat. If the checksum did not validate in step **424**, control proceeds to step **428** where an error is indicated by the appropriate number of beeps. If the keyboard controller **62** was okay in step **426** or after step **428**, control proceeds to step **432** where values are written to ensure that all of the other CPUs in the system are asleep so that they do not interfere with the POST and booting process. Control then proceeds to step **434** (FIG. 5B) where miscellaneous initialization operations occur. After a number of these operations, control proceeds to step **436** where the call ASSIGN_CPUs sequence **600** (FIG. **8**) is called. Control then proceeds to step **438**, where more miscellaneous initialization operations occur. Control proceeds to step **440** where the CHECK_PROC_OK sequence **650** (FIG. **9**) is called, which checks to see if any CPUs have failed and if so provides a message. Control then proceeds to step **442** where the INIT_PROC_AVAIL sequence **700** (FIG. **10**) is called. This sequence **700** sets available bits in a stored flag byte for each of the CPUs after physical CPU**0**, as that is the one assumed to be operating. Control then proceeds to step **444**, where the POST_INIT sequence **720** (FIG. **11**) is called. This sequence **720** is used to start and initialize any secondary processors, which are processors other than CPU L**0**, in the system. Control then proceeds to step **446** where more miscellaneous initialization operations occur. Control proceeds to step **448** where the BOOT_CPU sequence **850** (FIG. **14**) is called. This sequence **850** is used to check if the CPU currently executing has developed errors during the initialization operations and thus is sufficiently dependable to continue the boot process. Control proceeds to step **450** where miscellaneous initialization and POST or power-on self test operations occur. Ultimately control proceeds to step **452**, which is the booting process of the computer system C.

It is noted that on the preferred microprocessors, step **422** will be executed in well less than the hardware CPU rotation time, but that any steps after step **422** are assumed to have occurred well after the rotation time. Particularly, step **448** will have occurred well after the rotation time.

The CHECK_WELL_PROCS sequence **500** (FIG. **6**) commences operation at step **502** where a pointer is set to the beginning of the wellness log. Control proceeds to step **504** to get the wellness entry indicated by the pointer. In step **506** a determination is made whether the wellness entry indicates a cache parity error. If not, control proceeds to step **508** to determine if it indicates a CPU parity error. If not, control proceeds to step **510** to determine if the entry indicates that a processor POST error has occurred, that is an error during the POST procedure described below. If not, then none of these critical errors have occurred, and the particular CPU is considered sufficiently reliable to continue processing. Control proceeds to step **512**, where the pointer value is incremented. At step **514** a determination is made whether the last entry in the wellness log has been checked. If not, control returns to step **504** and the next entry is obtained. If so, control proceeds to step **516**, which is a return from the sequence **500**.

If errors have been indicated in steps **506**, **508** or **510**, control proceeds to step **518** to determine the physical CPU number of the CPU indicating the failure. Control then proceeds to step **518** to determine if a failed flag is set for that particular CPU. Preferably this would have been set when the error occurred during normal operations when the wellness entries were actually made, but it could have occurred at other times as seen below. If the flag is not set, control proceeds to step **522** where it is set for that particular processor and control then proceeds to step **512**. If the flag was already set, control proceeds from step **520** to step **512**. Thus the sequence **500** scans the wellness log and indicates CPU failure for certain critical errors which have previously occurred.

The CHK_FAIL_PROCS sequence **550** commences at step **552** where a value indicating which CPUs are installed is obtained. This can be obtained by referencing a value contained in the CMOS memory or can be done by checking each processor slot. Control proceeds to step **554** where a value indicating the failed CPUs is obtained, such as the value developed in the CHECK_WELL_PROCS sequence **500**. Preferably the installed and failed CPU values are contained in the environment variable area, with particular bits indicating particular processors. Control proceeds to step **556**, where all of the failed CPUs are removed from the installed value. Control then proceeds to step **558** to determine if any CPUS are indicated as remaining. If not, control proceeds to step **560**, which is a return from the sequence **550**. If so, control proceeds to step **562** to determine if the current physical CPU is installed and has not failed. The current physical CPU is determined by doing a call to the appropriate location as described above and the failed flag is then compared. If it is installed and has not failed, this indicates that the CPU is considered sufficiently reliable and control proceeds to step **560**.

If the failed flag is set, then the control essentially proceeds to a closed loop, commonly referred to as JMP $ instruction. This is quite proper in that the CHK_FAIL_PROCS sequence **550** is called from the COLD RESET sequence **400** well within the **2** second period used by the CPU rotation timer to indicate the need for the hardware to rotate the logical CPU value. Thus in this manner if the computer C goes through a cold reset, i.e. a power up condition, then its failed status of the booting processor is quickly determined and if certain critical errors have previously occurred, the CPU simply shuts down and the hardware proceeds to rotate the logical CPU number so that the booting process can continue. As discussed above, the reason for this rotation of the CPUs is that in many cases it may be possible for a processor to actually complete the entire POST and boot process before it fails. This is particularly true where parity error occurred as they are relatively infrequent and if they do occur, the consequences are relatively extreme. Thus it is considered that a CPU which has previously had a parity error in the processor itself or in the cache system is not sufficiently dependable over the long term to allow it to fully operate the computer C.

The ASSIGN_CPUS sequence **600** (FIG. **8**) commences at step **602** where the installed CPUs are obtained. Control proceeds to step **604** where the list of failed CPUs is obtained. Control proceeds to step **606** to determine if there are any good CPUs. If not, an error flag is set in step **608** and control proceeds to step **610**, which is a return from the sequence **600**. If there are good CPUs, control proceeds from step **606** to step **612**, where logical CPU numbers are assigned to nonfailed physical CPUs in physical order. Control then proceeds to step **610**.

The CHK_PROC_OK sequence **650** (FIG. **9**) commences at step **652** to determine if CPU **P0** is equal to CPU **L0**. That is, is the physical **0** CPU also the logical CPU **0**, indicating that it is apparently properly booting the system and has not previously failed. If not, control proceeds to step **651** to determine if CPU **P0** is actually installed. If not, control proceeds to step **654** where a no CPU **P0** error message is provided to the user. If it is installed, control proceeds to step **656** where the failed bit is set in the failed byte to indicate the CPU **P0** has failed. If the CPU **P0** is equal to CPU **L0** in step **650** or after steps **654** and **656**, control proceeds to step **658**, where the failed byte is obtained. Control then proceeds to step **660** where the installed byte value is obtained. Control proceeds to step **662** where an error message is provided to the user to indicate any failed and installed CPUs. Control then proceeds to step **664**, which is a return.

The INT_PROC_AVAIL sequence **700** (FIG. **10**) has a step **702**. In step **702**, the computer system C reads the identifications for each CPU slot after physical **0**, as physical **0** is assumed to be the one booting, and if there is a value, then a CPU is present and the available bit is set in the available byte. Control then proceeds to step **704** which is a return from the sequence.

The POST_INT sequence **720** commences at step **722** to determine if there are any secondary processors. If not, control proceeds directly to step **724**, which is a return, as there is nothing to initialize. If a secondary processor is present, control proceeds to step **726**, where a pointer is set to indicate CPU **L1**. Control proceeds to step **728** to determine if the cache for that particular CPU had a BIST error. If not, control proceeds to step **730** where the POST_PROCESS sequence **750** (FIG. **12**) is called. This sequence **750** initializes operation of the secondary processor. After step **730**, control proceeds to step **738**. If there was an error, control proceeds to step **732** where an error message is displayed. In step **734** the error is logged in both the wellness log and the environment variable. Control proceeds to step **736** where the logical CPU number is incremented. Control proceeds to step **738** where the physical CPU slot number is incremented. Control proceeds to step **740** to determine if this was the last slot. If not, control returns to step **728** where the process is repeated. If it was the last slot, control proceeds to step **724**.

The POST_PROCESS sequence **750** commences at step **752**, where a vector value is provided at the address 40:67h to a sequence called INIT_PROCESS or RESET **800** (FIG. **13**). The 40:67h vector sequence is described in the referenced applications, but briefly is a location previously defined in IBM PC compatibles to receive a vector location used under certain conditions after a reset is issued to the processor. The original reason for this vector was to allow an 80286 processor to return from protected mode to real mode, but the vector location is now utilized for other operations. Control then proceeds to step **754** where the particular CPUs SLEEP bit is cleared. This will allow the CPU to begin operations. Control proceeds to step **756**, where a period of time equal to an initialization delay is passed. This time period is provided to allow a certain period to have passed for the particular CPU to begin operations. Control proceeds to step **758** to determine if the CPU has indicated that it has started the initialization procedure. If so, control proceeds to step **760**, where a further period of time elapses. Control proceeds to step **762** to determine if a flag has been set by the CPU which is initializing to indicate that the CPU in initialization operation has finished. If so, this is an indication the CPU has fully initialized, so control proceeds to step **764** where it is put back to sleep pending execution for further tasks.

If the CPU did not indicate START in the allotted time or it did not indicate finished in the allotted time, control proceeds from steps **758** and **762** to step **768**, where the CPU is put back to sleep and a reset condition is indicated. Control proceeds to step **768**, where an error message is displayed and then to step **770** where the error is logged in the wellness log and environment variable. Control then proceeds to step **772**, which is a return from the sequence **750**, which is also where execution proceeds after step **764**.

The INIT_PROCESSOR_RESET sequence **800** commences at step **802**, where the CMOS_RST_CODE value is obtained. As noted above, 40:67h is utilized for certain reset operations as a vector pointer. The way this is indicated is that a value contained in the CMOS memory portion of the RTC **64** is utilized to indicate the particular type of reset operation occurring. If a certain type is occurring, then a vector from 40:67h is executed, so that a more complete initialization and reset sequence is not executed. Control proceeds to step **804** to determine if the 40:67h vector reset is in effect. If not, control proceeds to step **806** to determine if this sequence **800** is being executed on a secondary processor. If not, control proceeds to step **808** which is the normal reset sequence. Effectively step **808** causes an entry into step **432** of the COLD RESET sequence **400**. If vectored reset was indicated in step **804** or if this was a secondary processor in step **806**, control proceeds to step **810**, where the STARTED flag is set to inform the other CPU in step **758** that the CPU initialization operation has started. Control proceeds to step **812** where initialization steps occur. Control then proceeds to step **814** where the INIT_DONE flag is set to indicate to the other CPU that initialization has completed. Control then proceeds to step **816**, where the CPU puts itself to sleep.

The BOOT_CPU sequence **850** commences at step **852**, where the ASSIGN_CPUS sequence **300** is called. Control then proceeds to step **854** where the failed byte is obtained and to step **856**, where the installed byte is obtained. Control proceeds to step **858**, where the two values are combined to develop an installed and not failed list. Control then proceeds to step **860** where the physical and logical CPU numbers for the particular processor on which the sequence is operating are obtained. Control then proceeds to step **862** to determine if the physical CPU as indicated by step **860** is equal to the first good CPU as indicated in the combined installed and not failed value provided in step **858**. If so, this is an indication that everything is satisfactory and control proceeds to step **864**, which is a return from the sequence. If the physical CPU is not the first good CPU, this is an indication that an error has previously occurred on CPU **P0** such that it can no longer be considered reliable. Then in step **863** the state of all the registers in the particular CPU are stored for later recall. Control then proceeds to step **866**, where a message is displayed indicating control of the system is switching from one CPU to another active CPU in the system. Control then proceeds to step **868**, where the vector pointing to the SWITCH_START sequence **880** (FIG. **15**) is stored at 40:67h. Control proceeds to step **870** where operation is set up to perform a vectored reset from 40:67. This would include probably setting the CMOS values. Control proceeds to step **872**, where the SLEEP bit in CPU **L0** is cleared. By definition, this sequence can not be operating on CPU **L0** as the ASSIGN_CPUS sequence **300** executed in step **852** would not have assigned a logical value to this particular CPU as it has been indicated as being failed. Thus by clearing the SLEEP bit in logic in CPU **L0**,

the first active processor will actually have been activated. Control then proceeds to step **874** to determine if the new CPU has started operations by looking for the START flag. If so, control proceeds to step **876** where the CPU puts itself to sleep. If it has not started after a proper time, control proceeds to step **878** where the new CPU is put back to sleep and reset. Control then proceeds to step **864**.

The SWITCH_START sequence **880** commences at step **882**, where the CPU state which had been saved by the prior processor in step **864** is restored, so that effectively the new processor has the same state as the prior processor so operation can commence at effectively same point. Control proceeds to step **884** to determine if the old CPU is asleep yet. Control remains at step **884** until it has. Once it has gone to sleep, control proceeds to step **886**, where a message is provided to indicate that the switch has occurred. Control then returns in step **888**. A return is proper because by the state being restored, operation is able to be continued as though there was no processor switch. Effectively the return step **888** is the same as the return step **864** and returns operation to proceed at step **450**. Thus in the BOOT_CPU sequence **850** the processors are rotated automatically based on the critical error log for errors which occurred during later initialization operations and had not occurred by the initial operations. The switch is done using the BOOT_CPU sequence **850** because step **448** occurs well after the preferred 2 second time for the hardware CPU rotation.

Therefore the computer system according to the preferred embodiment not only automatically rotates CPUs to the first fully operational CPU in case of major hardware failure of the boot processor but also transfers control should the processor have developed certain critical errors during its later operation such that it would be considered unreliable and undependable for long term operation. By this further improvement it is clear that the system will boot completely on only fully functional and dependable CPUs, thus reducing user aggravation and avoiding the potential loss of data.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

We claim:

1. A method for booting a multiprocessor computer system with a processor, comprising the steps of:

maintaining a log of predetermined critical errors which have occurred on the computer system;

resetting the computer system and beginning execution of initialization operations on a first processor of the multiprocessor computer system;

determining if said first processor has previously had one of said predetermined critical errors; and

if one of said predetermined critical errors had occurred, disabling said first processor and transferring execution to a second processor of the multiprocessor computer system to perform initialization operations.

2. The method of claim **1**, further comprising the step of:

transferring execution from the first processor to the second processor if the first processor has not performed a predetermined one of said initialization operations in a predetermined time, and

wherein said step of determining if said first processor has previously had one of said predetermined critical errors occurs before said predetermined time, and

wherein said step of disabling said first processor occurs before said predetermined time and includes halting operation of said first processor before performing said predetermined operation so that said step of transferring execution if the first processor has not performed said predetermined operation causes transfer of execution.

3. The method of claim **2**, further comprising the steps of:

if none of said predetermined critical errors had occurred, performing said predetermined operation;

performing tests on said first processor after performing said predetermined operation and logging a critical error if one occurs;

determining if one of said tests indicates one of said predetermined critical errors has occurred;

if one of said predetermined critical errors has occurred, storing the processor state, activating said second processor and halting operation of said first processor; and

restoring said stored processor state to said second processor and performing the remaining initialization operations on said second processor.

4. The method of claim **1**, wherein said step of determining if said first processor has previously had one of said critical errors includes the steps of:

performing tests on said first processor and logging a critical error if one occurs; and

determining if one of said tests indicates one of said predetermined critical errors has occurred, and

wherein said step of disabling said first processor includes the steps of:

if one of said predetermined critical errors has occurred, storing the processor state, activating said second processor and halting operation of said first processor; and

restoring said stored processor state to said second processor and performing the remaining initialization operations on said second processor.

5. A system for booting a multiprocessor computer system with a reliable processor, the system comprising:

means for detecting and maintaining a log of predetermined critical errors which have occurred on the computer system; and

means for initializing the computer system including:

means for resetting the computer system and beginning execution of initialization operations on a first processor of the multiprocessor computer system;

means for determining if said first processor has previously had one of said predetermined critical errors; and

means for disabling said first processor and transferring execution to a second processor of the multiprocessor computer system to perform initialization operations if one of said predetermined critical errors had occurred to said first processor.

6. The system of claim **5**, wherein said means for initializing the computer system further includes:

means for transferring execution from the first designated processor to the second designated processor if the first designated processor has not performed a predetermined one of said initialization operations in a predetermined time; and

wherein said means for determining operates before said predetermined operation is performed, and

wherein said means for disabling halts operation of said first processor before said predetermined operation is performed.

7. The system of claim 6, wherein said means for initializing the computer system further includes:

means for performing tests on said first processor and logging a critical error in said log if one occurs;

means for determining if one of said tests indicates one of said predetermined critical errors has occurred;

means for storing the processor state of said first processor and halting operation of said first processor and for activating said second processor if one of said predetermined critical errors has occurred; and

means for restoring said stored processor state to said second processor and performing the remaining initialization operations on said second processor.

8. The system of claim 5, wherein said means for determining further includes:

means for performing tests on said first processor and logging a critical error in said log if one occurs; and

means for determining if one of said tests indicates one of said predetermined critical errors has occurred; and

wherein said means for disabling further includes:

means for storing the processor state of said first processor, activating said second processor and halting operation of said first processor if one of said predetermined critical errors has occurred; and

means for restoring said stored processor state to said second processor and performing the remaining initialization operations on said second processor.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,491,788 Page 1 of 1
DATED : February 13, 1996
INVENTOR(S) : Darren J. Cepulis et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

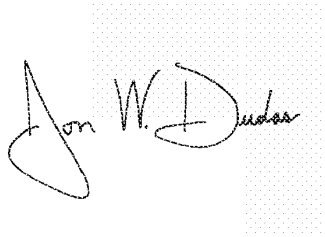
Title page,
Item [54], Title, delete "**TRANSFERRING**" and insert therefor -- **TRANSFERRED** --.

Column 3,
Line 27, delete "CPU" and insert therefor -- CPU; --.
Line 27, the sentence starting with "FIG. 5A," should start a new paragraph.

Column 14,
Line 39, before "processor," delete "reliable".

Signed and Sealed this

Twenty-ninth Day of November, 2005



JON W. DUDAS
Director of the United States Patent and Trademark Office

EXHIBIT
1009

Commercial Fault Tolerance: A Tale of Two Systems

Wendy Bartlett, *Member, IEEE Computer Society*, and Lisa Spainhower, *Member, IEEE*

Abstract—This paper compares and contrasts the design philosophies and implementations of two computer system families: the IBM S/360 and its evolution to the current zSeries line, and the Tandem (now HP) NonStop® Server. Both systems have a long history; the initial IBM S/360 machines were shipped in 1964, and the Tandem NonStop System was first shipped in 1976. They were aimed at similar markets, what would today be called enterprise-class applications. The requirement for the original S/360 line was for very high availability; the requirement for the NonStop platform was for single fault tolerance against unplanned outages. Since their initial shipments, availability expectations for both platforms have continued to rise and the system designers and developers have been challenged to keep up. There were and still are many similarities in the design philosophies of the two lines, including the use of redundant components and extensive error checking. The primary difference is that the S/360-zSeries focus has been on localized retry and restore to keep processors functioning as long as possible, while the NonStop developers have based systems on a loosely coupled multiprocessor design that supports a "fail-fast" philosophy implemented through a combination of hardware and software, with workload being actively taken over by another resource when one fails.

Index Terms—Computer systems implementation, fault tolerance, high availability.

1 Introduction

Since the 1960s, there have been a number of commercial systems introduced with a specific focus on high availability or fault tolerance. This paper describes the design points and evolution of two of them—IBM's S/360 and its evolution to the zSeries, and Tandem's (now HP's) NonStop systems. Other well-known systems in this space include Stratus and Marathon.

For enterprise customers, definitions of what constitutes "availability" vary and may factor in not only the ability to provide a service but also acceptable response time and whether the results are accurate and consistent. Application availability requirements also vary tremendously, including different applications for the same customer. The need for availability may be "7 x 24 x forever" (e.g., ATM/POS applications and manufacturing floors for three-shift operations), or it may be for 100 percent availability within given timeframes (e.g., stock markets), or it may allow for infrequent, short disruptions (many applications). A framework for describing levels of availability is shown in Fig. 1.

Both system families described in this paper were originally targeted toward the High Availability/Fault Tolerant quadrant and since then have evolved toward Continuous Availability. While the goal has been the same in both cases, their paths toward achieving it have been different. The details appear in later sections, but Table 1 gives a high-level comparison of their approaches.

- *W. Bartlett is with Hewlett Packard, 19333 Vallco Parkway MS 4421 Cupertino, CA 95014. E-mail: wendy.bartlett@hp.com.*
- *L. Spainhower is with IBM, 2455 South Road, Mail Station P314, Poughkeepsie, NY 12601. E-mail: lisa@us.ibm.com.*

Manuscript received 10 June 2004; accepted 20 Aug. 2004.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0084-0604.

1545-5971/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

2 Initial Target Audiences/Applications for the Systems

Both systems serve commercial business audiences for applications that require very high to continuous availability. From the beginning, IBM mainframes were designed with general-purpose business processing in mind. In the mid-1960s, most real-time business processes were manual and System/360 mainframes were used primarily for background jobs and overnight batch processing. The primary availability goals were to detect failures quickly enough to preserve data integrity and to locate and repair those failures quickly enough to not interfere with getting the tasks done by their deadlines. Over time, mainframes were used for more and more of the real-time work, although it tended to be local with well-defined hours of usage during the day and days during the week. This increased the requirements to lessen the frequency and impact of failures. As businesses became more global and moved to 24 x 7 operations, the demand for continuous operation became common. Limiting planned downtime required for change and upgrade became an increasingly important goal. IBM mainframes responded to each of these evolutionary needs for business computing.

The original Tandem NonStop systems were targeted at a few specific applications that had very high availability requirements, including back-end support for Automated Teller Machines (ATMs) or Point Of Sale (POS) devices and reliable message switching for applications such as credit card authorization. The expectation at the time for ATMs was that the infrastructure might be taken down for service at designated times (typically, 2:00-6:00 a.m. on Sundays) but otherwise was expected to supply 100 percent overall availability, so the emphasis was on fault tolerance rather than continuous operations. Individual machines might fail, but whole sites should not be

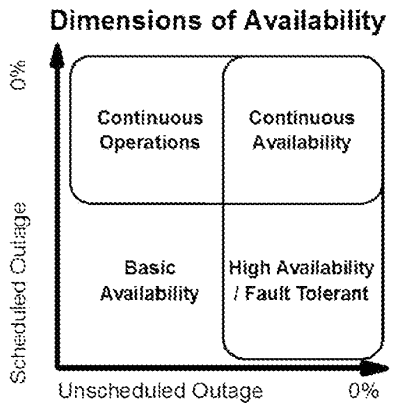


Fig. 1. Availability dimensions.

unavailable. Typically, a single site such as a bank branch would have two ATMs, connected via two different networks to two different systems, or at least to two different processors in the same system. NonStop servers soon moved into other financial markets such as stock exchanges and then into the telecommunications arena—not as switch replacements but for high-availability off-switch functions such as collection of call data records and billing. Use in other markets, including manufacturing shop floors, emergency 911 services, retail, and health care, followed shortly thereafter.

3 Initial Fault Tolerance Philosophies

3.1 zSeries

There was partial, but not complete, overlap in the original design principles for the two systems. Both focused on providing fault tolerance through duplicate components and paths. They also emphasized rapid error detection, as data integrity was and is a key requirement in their markets.

Three of the System/360 founding concepts are specific to dependability:

1. "It must be possible and straightforward to assemble systems with redundant I/O, storages, and CPUs so that the system can operate when modules fail.
2. As CPUs become increasingly reliable, built-in thorough checking against hardware malfunction is imperative for all systems, regardless of application.
3. Since the largest servicing problem is diagnosis of malfunction, built-in hardware fault-locating aids are essential to reduce downtimes. Furthermore, identification of individual malfunctions and of individual invalidities in program syntax would have to be provided" [1].

These concepts have come to be embodied as three prioritized principles on how to react to faults:

1. Ensure system integrity.
2. Provide uninterrupted applications.
3. Repair without disruption.

TABLE 1
System Comparison

	IBM zSeries	HP (Tandem) NonStop
Scope	General purpose server (Supports multiple OSs: zOS , VM, TPF, Linux) Integrated zOS Cluster (Parallel Sysplex)	Integrated system (Server, disk, comm, OS, DB, Txn monitor) Specific classes of target applications Loosely-coupled cluster of systems (ServerNet Clusters)
Fault Model	Two types: transient, permanent	Two types: recoverable, nonrecoverable
FT Operation	Explicit redundancy in support subsystems; implicit redundancy in computational subsystems Parallel Sysplex: no SPOF, distributed applications, data sharing	Multiple hardware components and paths Process pairs, mainly at lower levels of the system Persistent processes plus transactions at application level ServerNet Clusters: Distributed applications
Recovery Techniques	Begin with local retry Recover from transient Spare cache, memory, CPUs Spare or degrade from permanent Parallel Sysplex: Transaction-level restart	Takeover by backup processes in case of processor failure caused by either hardware or software Restart of persistent processes Transaction backout Switch paths
Data Integrity	Redundant CPU logic, Inline checking in I/O subsystem, ECC	Lockstepped microprocessors, ECC, end-to-end disk checksums, CRC

The zSeries, like its System/360-390 standalone mainframe predecessors, is designed to maximize hardware resource utilization during normal, fault-free operation, retry and recover locally when failures are detected, and continue operation without interrupting the ongoing applications. System packaging facilitates online maintenance of hardware components. Firmware and system software are designed to allow updates concurrent with normal operation in most cases. For well-tested software, clusters of zSeries mainframes, Parallel Sysplex, can be configured for no single point of failure in hardware or software. Maintenance, for repair or upgrade, can be performed online as far as the application is concerned.

Ensuring system integrity is paramount, even if application disruption is required in order to preserve it. These design principles apply equally well to the NonStop line, but its implementation philosophy is somewhat different.

3.2 NonStop Server

NonStop systems always have been designed to keep properly designed applications running after a single failure of any kind, to support online repair and reintegration of hardware components, and to preserve data integrity by preventing silent data corruption. For hardware, these goals have been met with a loosely coupled design based on multiple modules with multiple interconnections among them. The modularity helps to delineate fault containment zones and facilitate online repair. Individual modules are designed for ease of maintenance, in order to minimize repair time and reduce opportunities for service-induced failures. Fault detection is built-in to provide "fail-fast" capability to detect errors and isolate the offending hardware from the rest of the system. In contrast to most architectures, the software is an equal partner to the hardware in implementing fault tolerance and preservation of data integrity. The operating system provides its own layer of fault detection and fail-fast response, and supplies the infrastructure that allows applications to survive single processor failures. For a more detailed overview of the NonStop fault tolerance philosophy, see [2].

4 Design Principles

Although they use different system recovery designs, both HP and IBM mainframes rely on near-instantaneous error detection. Whether the aim is to fail fast and rapidly switchover or to transparently retry and recover locally, it is imperative that the scope of any faulty operation be strictly contained.

4.1 zSeries

There has been a steady advancement in the fault tolerant characteristics of IBM mainframes in the 40 years since System/360 was introduced [3]. As previously mentioned, the goal was and is to meet the requirements of general purpose business processing. System/360 was designed with robust error checking and was the first commercial computer to include Error Correction Code (ECC) in memory. Early system requirements were to reduce downtime by improving diagnosis. In the 1960s, computer downtime could be easily masked from customers and clients. Much of the computer workload was overnight batch and businesses depended on manual fall-back methods to continue processing bills, invoices, and service requests during downtime.

In the 1970s, designers improved mainframe fault tolerance with recovery from transient failures and increased the ability to mask hard failures with graceful degradation. For a specialized class of applications, S/370 Extended Recovery Facility was developed to permit one mainframe to monitor another and to assume its workload upon detection of a failure.

In the 1980s and early 1990s, as more and more business applications, particularly online transaction processing (OLTP), started to run 24 x 7, continuous availability grew as a mainframe requirement. To meet the demand, one major change was that online repair capabilities, formerly found only on multiprocessors, were greatly enhanced for all IBM mainframes. Initially, support subsystems—power, cooling, service processor—were either duplicated or designed to be noncritical to the ongoing processing of the computer. Later, those functional elements replicated for higher performance, like CPUs and I/O channels, were designed so that an operational element could take over in a transparent fashion for one that had failed. Increasingly, redundant elements were packaged so that they could be removed, replaced, and restarted without interrupting processing. Another type of online repair, electronic sparing, was also introduced. Mean time to application crash increased to 10 years with these enhancements.

The mid-1990s System/390 semiconductor technology transition from Emitter Coupled Logic (ECL) to Complementary Metal Oxide Semiconductor (CMOS) began a new evolution of fault tolerant design [4]. Fault tolerant enhancements were added with each new generation of CMOS, and today's zSeries has the most extensive detection and recovery in the mainframe history, as well as mean time to hardware-caused application crash in excess of 30 years.

Today, the zSeries memory hierarchy uses store-through (write-through) cache design, spare elements, and Error Correction Codes (ECC) for data redundancy [5]. Pending instruction, results are kept in both a store-through microprocessor cache (L1) and an ECC-protected store buffer and completed results are immediately stored into L2. L1 data is thus always replicated, allowing byte parity to be sufficient within the cache. A transient L1 failure is recovered by CPU instruction retry. For a permanent failure, depending on the scope, a cache line or quarter-cache delete is performed dynamically. A deleted line may be restored with a spare line at the next power on. Shared Level 2 (L2) cache is protected by ECC. Permanent faults in L2 that might result in an uncorrectable data error can be avoided by using a cache delete capability. Faulty locations either in the data array or in the address directory can be dynamically marked as invalid and a spare line can be substituted for the failed one. Each main memory (L3) array chip contributes one bit to an ECC word, so a (72,64) SEC/DED code can protect the system from catastrophic failures due to any single array chip failure. Automatic online repair of faulty DRAMs is done using built-in spare

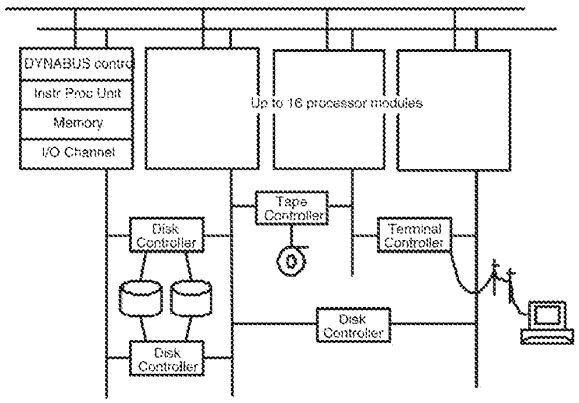


Fig. 2. Original tandem system architecture (1976).

chips. Counts of correctable errors are maintained on a per chip basis. When a threshold is exceeded, the data from the overthreshold chip is dynamically transferred to an error-free spare chip.

The I/O subsystem exploits the redundant paths between all devices and main memory to minimize the scope of any failures [6]. All paths are normally active and are used to enhance performance as well as to provide backup for one another if one should fail. The I/O channel adapters perform direct memory access with robust memory protection on behalf of I/O devices such as disk and tape storage, network communications, and server-to-server cluster (Parallel Sysplex) services. They prevent I/O devices from unauthorized memory read operations and from memory write operations into arbitrary memory locations. I/O channel adapters also provide error isolation by preventing the propagation of interface errors into the system. The bulk of the I/O subsystem hardware is packaged into various circuit cards that provide many different I/O interfaces and most of these cards can be concurrently replaced. Each I/O card position has an interface that controls the soft switch on the card. Before an I/O card is removed, the soft switch is deactivated causing the power on the card to drop. After a card is inserted, the soft switch is activated and the power to the card is turned on. To reduce human errors during concurrent card replacement, a group of LED indicators, one for each card position, is used to positively identify the card to be replaced.

To meet mainframe availability requirements, the power and cooling subsystem is designed with no single points of failure and concurrent repair capability of virtually all components.

4.2 NonStop Server

The Tandem NonStop system shown in Fig. 2, introduced in 1976, consisted of two to 16 independent processors connected by a pair of interprocessor buses collectively referred to as the Dynabus. Each processor had its own memory and ran its own copy of the operating system. Each processor also had an I/O bus. Each dual-ported I/O controller was connected to two processors' I/O buses, and had internal logic that selected which port was currently the primary path. If a processor or its I/O bus were to fail, the controllers whose primary paths were currently configured to use that I/O bus would switch ownership to their backup paths. Controller configuration was flexible enough that the workload of a failing processor could be spread over multiple surviving processors rather than all being taken over by a single processor.

The general design principle was that there be at least two of everything, including power supplies and fans as well as the more obvious processors, controllers, and peripherals. Dual-ported controllers and dual-ported peripherals provided four paths to each device. For disks, the use of host-based RAID-1 mirrored pairs of drives provided eight paths to the data and offered improved data integrity.

There were several special requirements on the hardware design. The first was that the Dynabus interfaces had to protect against a single processor failing in a way that disabled both buses or induced errors on both buses when power was removed from the processor module. The second was that power be managed in a way that allowed a power supply failure to be tolerated. The third was that hot insertion of boards into powered slots be supported, which enabled both online repair and online expansion. Battery backup, which preserved memory contents and supported power-fail restart, was standard [7]. Online repair was designed in for all hardware components, including power supplies and fans.

Beyond those requirements, there was a general need for extensive error checking using appropriate techniques such as parity, parity prediction, ECC, self-checking, and checksums. Controllers had their own fail-fast requirements to not corrupt data and, if possible, make available diagnostic information on the failure. End-to-end checksums were generated at the host and written to disk along with the data, with the checksum being recalculated and compared when the data was read back. The hardware was, in a sense, fault-intolerant—it was designed to fail quickly and cleanly at the first sign of a nonrecoverable error.

The system was architected to maximize useful, active modules and minimize the existence of "redundant," inactive modules, both for cost reasons and to prevent the accumulation of undetected, latent errors. This maximization was primarily the responsibility of the operating system software. As examples, processes were run in all processors—there were no "standby" processors; the message system used both paths of the Dynabus; and the disk process issued reads against both halves of the mirror (which also improved read performance).

Both the hardware and the software were optimized to support the message-based operating system design, including microcode-level support for sending and receiving messages. The choice of design yielded a system that is both highly distributed and fault tolerant. In addition to being easy to repair, the system could be easily expanded online.

Software fault tolerance was built into the operating system from the beginning [8], [9]. The system was structured as a set of processes communicating via interprocess messages, influenced by Dijkstra's "THE" system [10] and Brinch Hansen's implementation of a message-based operating system nucleus [11]. Each processor was

preconfigured with a small set of daemon-type processes that were responsible for specific tasks within that processor such as process control and memory management, along with a supporting set of library routines, interrupt handlers, etc. The key software abstraction, process pairs, allowed two communicating processes running in different processors to collectively represent a named resource such as a disk volume, communications line, or group of terminals, with one process functioning as the primary at any point in time and checkpointing necessary state to its backup. If a primary process or its processor were to fail, the backup process would take over and the operating system would transparently redirect requests to it without the application seeing an error. Operating system infrastructure was put in place to reliably maintain the named resource table that provides the redirection transparency.

Process pairs are an effective design for recovering from transient problems ("Heisenbugs" [12]) such as race conditions, where the identical conditions are unlikely to be in effect when the backup process takes over. They are vulnerable to deterministic defects ("Bohrbugs"), but those are relatively rare in a well-tested system. See [13] for an empirical study on this topic.

The initial focus of the system was to provide tolerance against single faults. Over time, the direction has been to add more support for continuous operations in order to serve the needs of customers who have been under increasing pressure to provide true continuous availability.

5 Advanced Design

5.1 Chip Technology

The designers of both system lines have had to continuously redesign their error checking logic in order to deal with both evolving technologies and the increasing rate of transient errors that is the inevitable side effect of the decreasing silicon feature size associated with newer, faster microprocessors and other components [14].

5.1.1 zSeries

From System/360 through System/390 ECL mainframes, increasingly complete and effective inline error checking was built into the circuitry. Within the CPU, parity and ECC were used extensively for all arrays, data paths, and control paths. Control logic, such as state machines and ALUs, used a variety of mechanisms such as parity prediction or illegal state detection.

For CMOS, a totally new design approach was needed [15]. The requirement for dynamic recovery, coupled with other design constraints (e.g., power, performance, second-level packaging) pointed to a highly checked single-chip microprocessor. However, inline checking of control and arithmetic logic was prohibited by performance penalties. As shown in Fig. 3, the decision was made to duplicate I-unit and E-unit and compare outputs. With a carefully laid out floorplan, there are no cross-chip performance-limiting paths. The cycle time is the same as if the same design were implemented on a smaller chip with only one unchecked I-unit and E-unit. The compare and detect cycle is completely overlapped in the instruction execution pipeline, so cycles per instruction (CPI) is not increased.

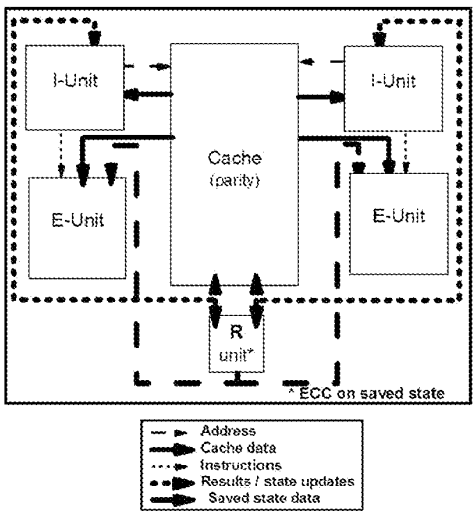


Fig. 3. Logical layout of the zSeries microprocessor chip.

Most of the microprocessor area is devoted to the cache where data is primarily moved around unchanged. Updates to memory are maintained in an ECC-protected store buffer during instruction execution and transferred to an ECC-protected L2 cache when instruction execution completes. Thus, for the on-chip cache, low-overhead parity checking is sufficient.

A key element of the microprocessor's R-unit is an ECC-protected register file, the checkpoint array. The checkpoint array keeps track of the entire state of the CPU including register contents and instruction addresses. Completed instructions are known to be error-free and the checkpoint array and the store buffer accurately describe the state of the machine at successful instruction execution.

The R-unit permits recovery to be completely controlled by hardware and identical for all instructions. The R-unit manages instruction retry when an error is detected as follows: Except for the R-unit itself, the CPU is reset and store buffer contents are sent to L2. The state of the CPU is returned to what it was at the last instruction completion. Serialized instruction processing starts up, retrying the instruction that caused the error. If it is successful, the failure was transient and the CPU resumes pipelined instruction processing. If not successful, then the error is permanent and the CPU will be stopped. For permanent failures, the service element moves the checkpoint to another, typically spare, microprocessor and instruction execution resumes with no apparent interruption.

Total circuit overhead for cache parity, Register Unit ECC and duplicate I-unit and E-unit is about 30 percent. Design, verification, and test are greatly simplified compared to inline checking.

5.1.2 NonStop Server

The original NonStop system was designed in 1974-1975. The hardware logic was based on 7400 series TTL

integrated circuits, with the most complex being the 4-bit ALU. As gate density within an IC package increased and gate arrays became available, the designers were able to increase the self-checking logic. Because processor design was under Tandem's control, it was possible to build in as much self-checking logic as the developers deemed necessary, using techniques such as parity on buses and branch prediction. This approach continued to be used for more than 10 years. In the late 1980s, Tandem moved to ECL logic for its Cyclone processor [16]. By then, it was becoming evident that the economics of building its own processors from the ground up were no longer attractive, and the Cyclone was the last internally designed processor in the line.

In 1991, Tandem moved to building its processors around commercial microprocessors, choosing the MIPS 3000 for its initial CMOS system. Because the amount of error checking within the microprocessor was not within Tandem's control and was considered to be inadequate for its target markets, the design changed to a scheme based on tight lockstepping of a pair of MIPS chips. That approach has continued to be used up until now, and a variant is being developed for future Intel Itanium®-based platforms. The new approach enables use of a looser, but still effective, level of synchronization between microprocessors. This design allows individual microprocessors to temporarily diverge in their instruction streams, e.g., to perform transient error recovery, without their associated logical processors being declared down. It also supports a triplex configuration for customers who wish to retain processor-level hardware single fault tolerance and data protection during processor repair actions.

5.2 Other Changes

5.2.1 zSeries

The next step in the evolution of availability beyond a single operating system and mainframe, whether uniprocessor or SMP, was the tightly coupled multiprocessor. Such mainframes were comprised of two mainframes, which could be run as one operating system image or as two isolated images. When run as two, there was no common hardware, thus no single point of failure. A very common operational mode was to run a tightly coupled multiprocessor as one operating system image and, when needed for maintenance or when a failure occurred, degrade to a single mainframe. This is in keeping with the general design philosophy of delivering all available capacity during normal operation and only using capacity for availability in unusual conditions.

Logical partitioning, (LPAR), further enhanced the ability to design a single mainframe for higher availability. LPAR allows multiple operating system instances to concurrently reside on one mainframe. Each is isolated from the other, test and production can be run at the same time, and new releases run with old. With the advent of LPAR came Extended Recovery Facility (XRF), which allowed a backup partition to be created within the same or a different mainframe. The partition needed very few resources during normal operation as it monitored the primary operating partition. It did require a high priority such that, upon failure, it could take over the necessary resources to keep the primary workload running on the backup. XRF in one mainframe had no software single point of failure (SPOF) and running across two mainframes, no software or hardware SPOF.

A major new design paradigm came about with the introduction of Parallel Sysplex, a data-sharing cluster design [17]. Multiple mainframes (up to 32) can be connected together via a coupling facility, or for the most robust installations, two coupling facilities. All mainframes, or at the least, all mainframes in the Sysplex sharing data, have equal access to shared disks as well as to the coupling facilities which manage the sharing. The basic hardware structure of the Sysplex is therefore to have no SPOF in the hardware.

To get the same results in the software, it is necessary to spread identical workloads across each (or a subset) of the mainframes in the Sysplex, shown in Fig. 4. At first, the notion of putting all critical software on each of the mainframes is somewhat counter intuitive if a primary guideline has always been to isolate workloads from one another on different hardware. With Parallel Sysplex, by contrast, the ideal high availability configuration is one in which each mainframe is a clone of the others, from a software perspective. Transactions are distributed to the mainframes via a Workload Manager (WLM) that determines which mainframe is least busy for that particular type of work at the moment. In the event of a failure anywhere within the Sysplex, there is no formal switch-over—merely a cessation of assigning work to a no-longer-available application, operating system, or mainframe. The design makes it unnecessary to preassign backups and, during normal operation, all available capacity can be used by the ongoing workload.

For those customers without the stringent availability needs of Parallel Sysplex but for whom the software reliability of a standalone mainframe is insufficient, it is possible to use LPAR to construct a partition (or partitions) that will serve as coupling facilities and to build a "Sysplex in a box." Given the highly robust mainframe hardware, with a mean time to unplanned customer application loss in excess of 30 years, this middle ground is adequate for many applications that are effectively mission-critical.

5.2.2 NonStop Server

As the engineers gained experience after the initial shipment, they adjusted algorithms to reflect what was seen in the field. Data integrity was and is the key system requirement, and several refinements were made to strengthen end-to-end data checks. Changes included adding end-to-end checksums on messages and sequence numbers to individual packets (originally, messages were simply checksummed at the packet level), and including the disk unit number in sector checksums to enable detection of the right data being written to the wrong disk. A less obviously availability-related feature was the addition of instrumentation to help debug performance problems; this was important because a poorly performing application effectively is unavailable.

The primary hardware change for NonStop servers (aside from the move to CMOS technology) has been to

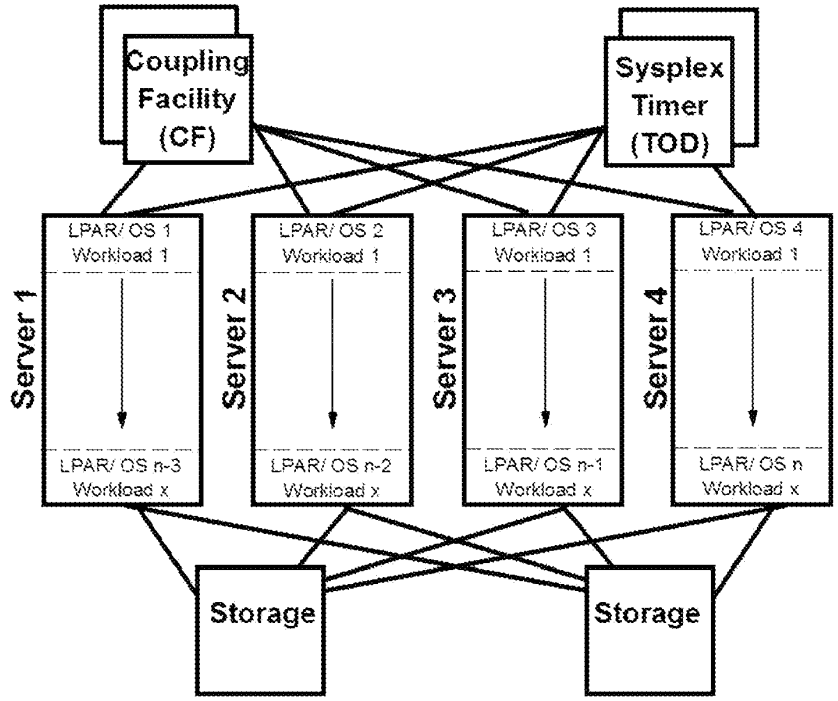


Fig. 4. zSeries parallel sysplex architecture.

rearchitect the system interconnect. As was shown earlier, the original NonStop architecture used a combination of I/O channels and interprocessor buses for its interconnect. Both were designed in-house, once again offering the opportunity to build in an appropriate level of checking. However, the Dynabus had to be greatly overdesigned in terms of capacity in order to support up to 16 processors and allow for future online upgrades to faster processors. In 1997, the NonStop S-series systems were introduced with a completely redesigned interconnect architecture. This architecture, shown in Fig. 5, takes a more network-oriented approach, which by its nature includes the capability to scale up bandwidth in all dimensions as the system grows.

The heart of the design is ServerNet, which carries both I/O and interprocessor traffic. The new interconnect had requirements to improve system performance, both through higher bandwidth and through offloading of part of the software communication stack, while meeting all of the expected fault tolerance requirements [18]. At the time, there were no commercially available interconnects that met all of the requirements, so ServerNet was designed internally. The original design was a 6-port router with 300MB/sec aggregate bandwidth. Wormhole routing was and is utilized to minimize both latency and the need for buffering at the router. DMA transfers, with appropriate protection, remove the need for host-level execution at the remote node. Single-bit command signal errors map to invalid commands. Each packet is protected with a cyclic redundancy check (CRC), and each router recalculates the CRC and appends either a "This Packet Good" or "This Packet Bad" symbol to the packet, making it easy to isolate link failures. ServerNet has subsequently been redesigned to provide even better bandwidth and diagnosability [19].

NonStop servers are inherently clustered due to their loosely coupled design. In the late 1980s, that design was

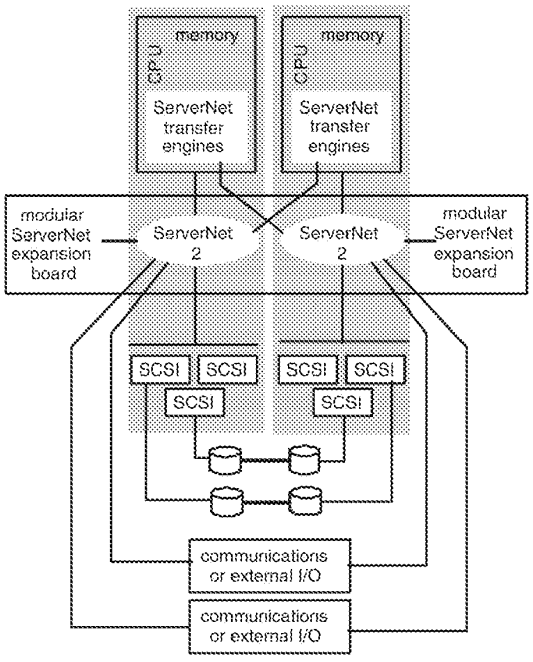


Fig. 5. Tandem NonStop Server S-series architecture (two processors shown).

extended to build the EXPAND networking product supporting homogeneous "clusters of clusters," allowing multiple interconnected systems to function in many ways as a single system image [20]. The original interconnect was via X.25 and other WAN protocols, with support added later for SNA, LAN, and IP. The process-and-message architecture provided a base that could be very straightforwardly extended to support multiple systems. Customers also began to require the ability to group together multiple systems in a data-center environment using high-speed, low-latency connections. A fiber-optic interconnect, FOX, was developed in 1983 to connect systems at subkilometer distances, and EXPAND was optimized to take advantage of the high reliability and speed of the new interconnect. For the S series, the data-center interconnect is ServerNet, which can cover distances of up to 15 KM.

The original software design assumed that programmers would work "close to the metal" and use a programming language, TAL, whose functionality was similar to C, with some customers possibly even supplying their own file systems. However, COBOL support also was available from almost the beginning. It soon became evident that most customers wanted their programmers to be able to focus on the business problem at hand without having to do any special coding to achieve fault tolerance and scalability. The PATHWAY transaction processing monitor [21] was developed to meet this need by handling the distributed and fault-tolerant aspects of the work on the programmer's behalf, with a high degree of transparency as far as the application is concerned. The programmer uses normal business logic and indicates transaction boundaries, and the system takes care of aborting incomplete transactions, restarting server instances, and redirecting requests when an application server process fails.

The other need that soon became evident was for data consistency as well as data integrity—that is to say, a database that is transactionally consistent. The Transaction Management Facility (TMF) was developed in the early 1980s to coordinate data accesses and updates in a manner that guaranteed the ACID properties of atomicity, consistency, isolation, and durability. In order to take advantage of TMF, an application programmer need only determine the appropriate locations in a program to begin, commit, and, if necessary, abort transactions. The Remote Database Facility (RDF) allows a TMF-protected database to be replicated at one or multiple additional sites at any geographic distance.

6 Design Trade Offs

Trade offs in computing system design are generally made among price, performance, and reliability. Most commercial servers are optimized for price, performance, or price/performance, and reliability enhancements which detract from the optimization point are closely scrutinized. Over time, reliability features once found only on servers optimized for reliability have become standard on virtually all servers. For instance, memory ECC and redundant power are now common. Reasons for the evolution include changing economics (e.g., when spare parts and warranty costs are included in the equation, certain reliability features may save money), technology failure rates which are unacceptably high for a broad range of servers, and increased customer expectation for better system availability at almost every price point.

For Tandem and zSeries systems, from the beginning, the target markets had extremely high data integrity requirements and the systems had to be designed accordingly. As described above, both systems were constructed using proprietary processors, which allowed for extensive self checking to be incorporated. Microprocessors have a component to fetch and decode instructions (I-unit), one or more instruction execution elements (E-unit), and a cache. Modern microprocessors usually include parity for cache data and data paths. Checking the control, arithmetic, and logical functions in the I-unit and E-unit is considered to be difficult, time-consuming, and to introduce performance penalties. As a result, other than zSeries, today's microprocessors leave these components unchecked. The Tandem design also checked those components in early systems, but since 1981 the approach has been to use off-chip logic to create perfectly checked microprocessor pairs.

In the industry, accepted wisdom is that single chip microprocessors are sufficiently reliable, that failures are rare and that other mechanisms—timeouts and software detection, for instance—will discover the few problems that do occur. This philosophy depends on continual reliability improvements; in reality, predicting chip technology failure mechanisms is problematic. Future CMOS technologies are expected to be less reliable; latches and dynamic logic will become more susceptible to alpha particles and other cosmic radiation [14]. Latent manufacturing defects and design errors that manifest themselves as transient problems also are sources of customer-visible errors.

Even today there is an exposure to malfunctions in the hardware resulting in incorrect data being written to memory. The model proposed by Horst et al. [22] projects a pessimistic bound of one in 75 unchecked microprocessors causing a data corruption error each year. Horst et al.'s prediction is based on an optimistic evaluation of software detection. It is true that, in some cases, timeouts and software detection will prevent data integrity problems; however, software does not reliably fail fast when errors are injected, resulting in corrupted data [23]. Even when the detection is successful, a system with no recovery will usually hang, causing application downtime, unacceptable for systems designed for high availability. zSeries and NonStop servers are the most well-known commercial systems designed to prevent data corruption by including extensive error checking in all microprocessor functional elements—combinatorial logic as well as arrays.

There are subtle advantages of building in a high level of integrity checking and retry/recovery logic to handle errors. For example, the resulting system is relatively resilient to design flaws. It may be possible to ship a system with one or more remaining occasional transient design errors if the occurrence rate is relatively low and the cost of recovery is not visible at the application level. Another advantage is that test and bringup time can be significantly reduced. The ability to fail-fast often makes it much easier to determine failure sources and uncover design and other problems.

Another important design trade off is "build versus buy." Particularly for subsystems which are compliant with industry standards, like I/O subsystems, there are notable

advantages in using high volume components. Commodity components are used wherever they provide a cost or time-to-market advantage, if the developers are able to provide the expected level of data integrity. For instance, most I/O adapters do not meet zSeries or NonStop server needs for memory protection; that is, inadequate checking is provided to assure that data does not get written to or read from an incorrect location. Some level of customization, through techniques such as lockstepping and end-to-end checksums, is added to incorporate commodity components into the two systems described herein.

6.1 zSeries

zSeries runs multiple operating systems—VM, Linux, and TPF as well as its flagship high-availability operating system, z/OS, the successor of MVS. Similarly to the hardware design principles, z/OS recognizes the inevitability of errors, resulting in a comprehensive approach of error isolation, identification, and recovery. z/OS provides many software reliability and availability features, quite a few of which can also be used by applications which run on z/OS.

Much of the z/OS kernel operating system is dedicated to providing advanced reliability, availability, and serviceability capabilities. All z/OS platform code is written to a set of guidelines to assure that errors can be detected, isolated, and documented. For example, "all code must be protected by a recovery routine including the code of recovery routines themselves; all control areas and queues must be verified before continuing; recovery and retry must be attempted if there is hope of success; all failures which cannot be transparently recovered must be isolated to the smallest possible unit, e.g., the current request, a single task (thread), or single address space (process); diagnostic data must be provided which, as an objective, will allow the problem to be identified and fixed after a single occurrence. The diagnostic data is provided even when retry is attempted and successful" [24].

Approximately 30 percent of the z/OS kernel operating system code is written to adhere to these guidelines, in addition to component code supporting the guidelines or code specifically designed to handle processor, storage, or I/O error recovery. Overall, about half of the z/OS kernel code is devoted to error handling. In addition to adhering to the guidelines, z/OS system software is committed to ensuring system integrity. Data and system functions are protected from unauthorized access, whether accidental or deliberate.

6.2 NonStop Server

Percentages of recovery code are not available for the NonStop server kernel, but the intent is similar to that of z/OS: to identify and contain the scope of faults in order to minimize application-level impact while preserving system and data integrity. Software consistency checking was built into the Tandem operating system from the beginning. As an example, the process ready list is kept as a doubly linked list, with the immediate links checked on every insertion and deletion. Data being written to disk is checkpointed to the backup disk process, but data being written to comm devices is not checkpointed as the performance penalty has been deemed to be too high and comm protocols have built-in recovery mechanisms. Data being written to disk includes host-generated sector checksums, which introduces a small I/O performance penalty and, until recently, required the drives to be specially formatted using 514-byte sectors.

7 The People Factor

In 1985, Gray [12] analyzed a set of Tandem failure data and published his results. His conclusion was that the key to high availability is tolerating operations and software faults—even then, fault-tolerant systems were very effective at masking hardware faults. Almost 20 years later, his conclusion still is valid. Recent evaluation of three very large Internet sites finds operator error to be the number one contributor to system downtime [25]. If anything, increased system complexity has offered even greater opportunities for fatal operational mistakes.

The best tool for reducing operator errors is automation, both to prevent operator errors and to better handle fault analysis and recovery. Automation can help the latter in three ways: It can analyze a long series of events more quickly and thoroughly than a human can, it can speed up recovery by initiating corrective action, and it can reduce the chance of an already perilous situation turning into a full-blown outage due to "fat-finger" mistakes. In cases where an automated event analysis subsystem cannot pinpoint the faulty component, it can at least narrow the options that the human needs to consider. Ideally, a fault-tolerant system also is a self-tending system, monitoring its own health, making corrections as necessary, and alerting the outside world where appropriate.

7.1 zSeries

In 2001, IBM introduced its Autonomic Computing initiative, a longterm strategy to invent, design and deploy computing systems, which operate to achieve business-level policies [26]. The realization of Autonomic Computing will be systems that demonstrate self-CHOP capabilities, that is, self-configuring, healing, optimizing, and protecting. Certain characteristics of zSeries servers exemplify what is meant by each of these.

Dynamic CPU Sparing, a process by which a failed CPU can be transparently replaced by a spare, was earlier described and is a good example of self-healing. Spare CPUs can also be brought online to meet increased customer resource requirements, a process known as Capacity Upgrade on Demand, and an example of self-configuring. LPAR, earlier mentioned as a mechanism for running multiple concurrent operating systems on one mainframe and keeping them isolated from one another, is also a self-optimizing feature. LPAR will maximize resource utilization in the server based on user established priorities. If a high priority partition is not using its allowed resources, LPAR will make them available to lower priority partitions. It will attempt to keep the mainframe as busy as possible all the time. A partition can also be capped, meaning that, even if all other operating system partitions are in a wait state, the capped partition can have only a certain amount of resources. This is useful for service providers who are "renting" capacity to others, but can also serve as a self-protecting mechanism to prevent denial of service attacks from taking over the entire mainframe.

7.2 NonStop Server

Automation is easier to construct if there is a unified logging subsystem to provide access to an integrated stream of information about what is going on in the system. In the mid 1980s, Tandem built a Distributed System Management subsystem (DSM) [27] to meet this need. DSM provides a consolidated log and logging infrastructure that are available to partners and end user applications as well as to the system provider, and includes a call-home capability. DSM events are tokenized, and can be viewed in filtered form to meet specific needs for programmatic response, Web viewing, PDA viewing, hardware incident analyzers, and so forth. However, the logging infrastructure alone is not sufficient; subsystems need to be intelligently instrumented to use it, and incident analysis software needs to be kept updated as new patterns are seen in the field. NonStop servers have long had a high degree of self-tending built into the system, both within the operating system and individual subsystems and, especially for hardware, in the form of incident analyzers.

8 Conclusion

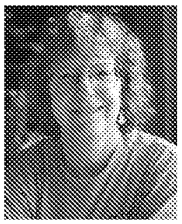
Fault tolerance and rapid fault detection for both hardware and software are key building blocks for constructing continuously available applications, but are not in themselves sufficient to accomplish the task. Similarly, built-in error recovery is important for handling the increasing rate of transient hardware and software errors. But, for builders of highly available systems, much of the current challenge lies in addressing the "second 90 percent" of the problem —the network, the environment, cyber attacks and, most importantly, the people and operational procedures responsible for keeping applications and systems running on an ongoing basis. The move toward self-tending systems by the producers of today's highly available commercial systems is a recognition of that need, and a trend that the authors expect to see continue.

Acknowledgments

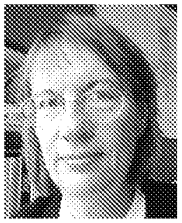
Wendy Bartlett would like to thank Joel Bartlett, Bob Taylor, and Kate Wilson. Lisa Spainhower would like to thank Bob Abrams.

References

[1] G.M. Amdahl, G.A. Blaauw, and F.B. Brooks, "Architecture of the IBM System/360," *IBM J. Research and Development*, vol. 8, pp. 87-101, 1964.
[2] W.B. Bartlett and B. Ball, "Tandem's Approach to Fault Tolerance," *Tandem Systems Rev.*, vol. 8, pp. 84-95, Feb. 1988.
[3] L. Spainhower, "System/360 to zSeries: Dependability in IBM Mainframes," *Dependable Computing Systems: Paradigms, Performance Issues and Applications*, 2004.
[4] L. Spainhower and T.A. Gregg, "IBM S/390 Parallel Enterprise Server G5 Fault Tolerance: A Historical Perspective," *IBM J. Research and Development*, vol. 43, nos. 5/6, pp. 863-874, 1999.
[5] P.R. Turgeon, P. Mak, M.A. Blake, M.F. Fee, C.B. Ford III, P.J. Meaney, R. Seigler, and W.W. Shen, "The G5/G6 Binodal Cache," *IBM J. Research and Development*, vol. 43, nos. 5/6, pp. 661-670, 1999.
[6] T.A. Gregg, "S/390 CMOS Server I/O: The Continuing Evolution," *IBM J. Research and Development*, vol. 41, nos. 4/5, pp. 449-462, 1997.
[7] J.A. Katzman, "System Architecture for NonStop Computing," *Proc. IEEE CS Int'l Conf. Technologies for the Information Superhighway*, pp. 77-80, 1977.
[8] J.F. Bartlett, "A 'NonStop' Operating System," *Proc. Hawaii Int'l Conf. System Sciences*, pp. 103-119, 1978.
[9] J.F. Bartlett, "A NonStop Kernel," *Proc. Eighth SIGOPS European Workshop*, pp. 22-29, 1981.
[10] E.W. Dijkstra, "The Structure of the 'THE' Multiprogramming System," *Comm. ACM*, vol. 11, pp 341-346, 1968.
[11] P.B. Hansen, "The Nucleus of a Multi-Programming System," *Comm. ACM*, vol. 13, pp. 238-241, Apr. 1970.
[12] J. Gray, "Why Do Computers Stop and What Can Be Done About It?" Technical Report TR85.7, Tandem Computers, Cupertino, Calif., June 1985.
[13] I. Lee, "Software Dependability in the Tandem Guardian System," *IEEE Trans. Software Eng.*, vol. 8, pp 455-467, May 1995.
[14] C. Constantinescu, "Impact of Deep Submicron Technology on Dependability of VLSI Circuits," *Proc. 2002 Int'l Conf. Dependable Systems and Networks*, pp. 205-209, 2002.
[15] C.F. Webb and J.S. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," *IBM J. Research and Development*, vol. 41, nos. 4/5, pp. 463-473, 1997.
[16] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluation*. Burlington, Mass.: Digital Press, pp. 586-648, 1992.
[17] J.M. Nick, B.B. Moore, J.Y. Chung, and N.S. Bowen, "S/390 Cluster Technology: Parallel Sysplex," *IBM Systems J.*, vol. 36, no. 2, pp. 172-201, 1997.
[18] B. Horst, "TNet: A Reliable System Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 37-45, Feb. 1994.
[19] D. Garcia and W. Watson, "ServerNet II," *Proc. Parallel Computing, Routing, and Comm. Workshop*, 1997.
[20] Tandem Computers, *EXPAND Reference Manual*. Cupertino, Calif., 1986.
[21] Tandem Computers, *Introduction to Pathway*. Cupertino, Calif., 1985.
[22] R. Horst, D. Jewett, and D. Lenoski, "The Risk of Data Corruption in Microprocessor-Based Systems," *Proc. Fault-Tolerant Computing Symp.*, pp. 576-585, 1993.
[23] S. Chandra and P.M. Chen, "How Fail-Stop Are Faulty Programs?" *Proc. 28th Int'l Symp. Fault Tolerant Computing*, pp. 240-249, 1998.
[24] S. Turner, C. Henry, G. Horvath, and J. Kibble, "Selecting a Server—Value of S/390," *IBM Redbook SG24-4812-01*, 1999.
[25] D.A. Patterson and D. Oppenheimer, "Architecture and Dependability of Large-Scale Internet Services," *IEEE Internet Computing*, pp. 41-49, Sept.-Oct. 2002.
[26] J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41-50, Jan. 2003.
[27] P. Homan, B. Malizia, and E. Reisner, "Overview of DSM," *Tandem Systems Rev.*, vol. 4, no. 3, Oct. 1988.



Wendy Bartlett received the BS degree in statistics and the MS degree in computer science: computer engineering, both from Stanford University. She is a distinguished technologist in Hewlett Packard's NonStop Enterprise Server Division. She was employed at Four Phase from 1976-1978, and has been employed by Tandem Computers, Compaq, and HP since then. She currently heads the indestructible scalable computing initiative for the NonStop server. She is a member of the IEEE Computer Society.



Lisa Spainhower is a graduate of the University of Michigan. She is a distinguished engineer in the Systems Technology and Architecture organization in IBM's Systems and Technology Group in Poughkeepsie, New York. She is a member of the IBM Academy of Technology, the IEEE, and the IEEE Computer Society.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

EXHIBIT
1003

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent of Michaelis *et al.*	§	Attorney Docket No.: 185945.012400
	§	
U.S. Patent No. 7,627,781	§	Customer No.: 67395
	§	
Issue Date: December 1, 2009	§	
	§	
Filing Date: October 25, 2004	§	
	§	
For: SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR	§	

DECLARATION OF ANDREW WOLFE, Ph.D IN SUPPORT OF REQUEST FOR *EX PARTE* REEXAMINATION OF

U.S. PATENT NO. 7,627,781

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

TABLE OF CONTENTS

TABLE OF MATERIALS CONSIDERED ..... **iv**

TABLE OF MATERIALS SUBMITTED WITH DECLARATION ..... **v**

I. INTRODUCTION **1**

A. Overview 1

B. Qualifications 1

C. Materials Considered 4

II. LEGAL PRINCIPLES **4**

A. Novelty and Obviousness 4

1. General Framework 5

2. Analogous Art 8

3. Combining Known Elements in the Art, Common Sense 9

4. Teaching Away 10

5. Secondary Considerations 11

B. Claim Construction 11

III. OPINION **12**

A. Summary of My Opinions 12

B. Level of Ordinary Skill in the Art 12

C. Background Technology 12

1. Boot Processor 13

D. Challenged Claims Are Unpatentable 13

IV. Ground 1: Kalyanasundharam in view of Kennedy Presents Substantial New Questions of Patentability **14**

A. Kalyanasundharam in view of Kennedy Renders Claims 1-6 and 9-15 of the '781 Patent Obvious to a POSITA 14

1. Overview of Kalyanasundharam 14

2. Overview of Kennedy 14

3. Overview of Motivations to Combine Kalyanasundharam and Kennedy 14

4. Reasonable Expectation of Success 15

5. *KSR* Rationales 15

V. Ground 2: Safford in view of Kennedy Presents Substantial New Questions of Patentability **19**

A. Safford in view of Kennedy Renders Claims 1-6 and 9-15 of the '781 Patent Obvious to a POSITA 19

1. Overview of Safford 20
2. Overview of Motivations to Combine Safford and Kennedy 20
3. Reasonable Expectation of Success 20
4. *KSR* Rationales 21

VI. Ground 3: Kennedy in view of Nguyen and Fox Presents Substantial New Questions of Patentability 25

A. Kennedy in view of Nguyen and Fox Renders Claims 1-30 of the '781 Patent Obvious to a POSITA 25
1. Overview of Nguyen 25
2. Overview of Fox 25
3. Overview of Motivations to Combine Kennedy, Nguyen, and Fox 25
4. Reasonable Expectation of Success 26
5. *KSR* Rationales 27

VII. GROUND 4: SAFFORD IN LIGHT OF CEPULIS PRESENTS SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY 32

A. Safford Discloses Each and Every Limitation of Claims 1 and 6 of the '781 32
1. Overview of Cepulis 32

VIII. DETAILED APPLICATION OF THE PRIOR ART TO EVERY CLAIM FOR WHICH REEXAMINATION IS REQUESTED 32

A. Proposed § 103 Rejections for the Challenged Claims 32
1. Ground 1: Kalyanasundharam in view of Kennedy 32
2. Ground 2: Safford in view of Kennedy 44
3. Ground 3: Kennedy in view of Nguyen and Fox 54

B. Proposed § 102 Rejections for the Challenged Claims 61
1. Ground 1: Safford in light of Cepulis 61

C. Secondary Considerations 64

IX. Declaration in lieu of OATH 64

TABLE OF MATERIALS CONSIDERED

Exhibit	Description
Ex. 1001	U.S. Patent No. 7,627,781 to Michaelis et al. (the "'781 Patent")
Ex. 1002	U.S. File History of the '781 Patent ("U.S. File History")
Ex. 1004	U.S. Patent No. 7,085,959 to Safford ("Safford")
Ex. 1005	U.S. Patent No. 5,450,576 to Kennedy et al. ("Kennedy")
Ex. 1006	U.S. Patent No. 7,251,746 to Fox ("Fox")
Ex. 1007	U.S. Patent No. 7,328,371 to Kalyanasundharam et al. ("Kalyanasundharam")
Ex. 1008	U.S. Patent Application Pub. No. 2004/0123201 to Nguyen et al. ("Nguyen")
Ex. 1009	Wendy Bartlett, et. al., *Commercial Fault Tolerance: A Tale of Two Systems, IEEE Transactions on Dependable and Secure Computing*, Vol. 1, No. 1 (January-March 2004) ("Bartlett")
Ex. 1010	Intel, *Extensible Firmware Interface Specification*, Version 1.10 (Dec. 1, 2002) ("Intel EFI Specification")
Ex. 1011	U.S. Patent No. 7,191,328 to Hobson ("Hobson")
Ex. 1012	Bozidar Levi, *UNIX Administration, A Comprehensive Sourcebook for Effective Systems and Network Management* (2002) ("Levi")
Ex. 1013	U.S. Patent No. 5,491,788 to Cepulis et al. (Cepulis)
Ex. 1014	*Intel IA-64 Architecture Software Developer's Manual*, Revision 1.1 (July 2000) ("Intel Itanium Manual")
Ex. 1015	*Intel Multiprocessor Specification*, Version 1.4 (May 1997) ("Intel Multiprocessor Specification")
Ex. AA	Appendix A: Claim chart comparing claims to Kalyanasundharam in view of Kennedy, as set forth in Ground 1.
Ex. BB	Appendix B: Claim chart comparing claims to Safford in view of Kennedy, as set forth in Ground 2.
Ex. CC	Appendix C: Claim chart comparing claims to Kennedy in view of Nguyen and Fox, as set forth in Ground 3.
Ex. DD	Appendix D: Claim chart comparing claims to Safford in light of Cepulis, as set forth in Ground 4.

TABLE OF MATERIALS SUBMITTED WITH DECLARATION

Exhibit	Description
Ex. A	Curriculum Vitae of Andrew Wolfe, Ph.D.

I. INTRODUCTION

A. Overview

1. I have been retained by Unified Patents, LLC as an expert witness in the field of computer systems and computer technology including bus technology. I have been asked to analyze certain claims of U.S. Patent 7,627,781 ("the '781 Patent") and provide my opinions on the patentability of certain claims of the '781 Patent based on 35 U.S.C. §§ 102 and 103. In addition to this Declaration, I am simultaneously providing declarations related to two other patents that are incorporated by reference in the '781 Patent: U.S. Patent No. 7,502,958 (the "'958 Patent") and U.S. Patent No. 7,624,302 (the "'302 Patent").

2. I am being compensated at my standard rate for my work performed on this matter and I am being reimbursed for my reasonable approved expenses. My compensation is not dependent on the content of my opinions. I have no financial interest in the outcome of this proceeding or in Unified Patents, LLC. I am over the age of 18 and am competent to make this Declaration.

B. Qualifications

3. My professional qualifications, experience, publications and presentations, and a listing of previous cases in which I have provided expert testimony are set forth in my CV, attached as Exhibit A.

4. I am the founder and sole employee of Wolfe Consulting. Through Wolfe Consulting, I provide technical and business analytics to businesses on processor technology, computer systems, consumer electronics, software, design tools, data security, cryptography and intellectual property issues. I have more than thirty-five years' experience developing products, researching, consulting, and teaching in those fields. In that time, I have worked as a computer architect, computer system designer, and as an executive in the PC and electronics business. I have also taught at some of the world's leading institutions in those fields, including Stanford University, Princeton University, Carnegie Mellon University, and Santa Clara University.

5. In 1985, I earned the B.S.E.E. degree in Electrical Engineering and Computer Science from the Johns Hopkins University. In 1987, I received the M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University, and in 1992, I received the Ph.D. degree in Computer Engineering from Carnegie Mellon University. My doctoral dissertation proposed a new approach for the architecture of a computer processor.

6. In 1983, I began designing touch sensors, microprocessor-based computer systems, and I/O (input/output) cards for personal computers as a senior design engineer for Touch Technology, Inc. During the course of my design projects with Technology, I designed I/O cards for PC-compatible computer systems, including the IPM PC-AT, to interface with interactive touch-based computer terminals that I designed for use in public information systems. I continued designing and developing related technology as a consultant to the Carroll Touch division of AMP, Inc., where in 1986 I designed one of the first custom touch-screen integrated circuits. I designed the touch/pen input system for the Linus WriteTop, which many believe to be the first commercial tablet computer.

7. From 1986 through 1987, I designed and built a high-performance computer system as a student at Carnegie Mellon University. From 1986 through early 1988, I also developed the curriculum, and supervised the teaching laboratory, for processor design courses.

8. In the latter part of 1989, I worked as a senior design engineer for ESL-TRW Advanced Technology Division. While at ESL-TRW, I designed and built a bus interface and memory controller for a workstation-based computer system, and also worked on the design of a multiprocessor system for real-time defensive weapons applications.

9. At the end of 1989, I (along with some partners) reacquired the rights to the technology I had developed at Touch Technology and at AMP, and founded The Graphics Technology Company. Over the next seven years, as an officer and a consultant for The Graphics Technology Company, I managed the company's engineering development activities and personally developed dozens of touch screen sensors, controllers, and interactive touch-based computer systems.

10. I have consulted, formally and informally, for a number of fabless semiconductor companies, particularly in the area of processor development. In particular, I have served on the technology advisory boards for two processor design companies: BOPS, Inc., where I chaired the board, and Siroyan Ltd., where I served in a similar role for three networking chip companies – Intellon, Inc., Comsilica, Inc., and Entridia, Inc. – and one 3D game accelerator company, Ageria, Inc.

11. I have also served as a technology advisor to Motorola and to several venture capital funds in the U.S. and Europe. Currently, I am a director at Turtle Beach Corporation (NASDAQ:

HEAR), providing guidance in its development of premium audio peripheral devices for a variety of commercial electronic products.

12. From 1991 through 1997, I served on the Faculty of Princeton University as an Assistant Professor of Electrical Engineering. At Princeton, I taught undergraduate and graduate-level courses in Computer Architecture, Advanced Computer Architecture, Display Technology, and Microprocessor Systems, and conducted sponsored research in the area of computer systems and related topics. From 1999 through 2002, I taught a Computer Architecture course to both undergraduate and graduate students at Stanford University multiple times as a Consulting Professor. At Princeton, I received several teaching awards, both from students and from the School of Engineering. I have also taught advanced microprocessor architecture to industry professionals in IEEE and ACM sponsored seminars. I am currently a lecturer at Santa Clara University teaching courses on Microprocessor Systems, Advanced Logic Design, Real-Time Computing, and Mechatronics.

13. From 1997 through 2002, I held a variety of executive positions at a publicly held fabless semiconductor company originally called S3, Inc. and later called Sonicblue Inc. I held the positions of Chief Technology Officer, Vice President of System Integration Products, Senior Vice President of Business Development, and Director of Technology, among others.

14. In the roles listed above, I worked with and managed architecture and design teams responsible for the development of general-purpose and special-purpose processors.

15. My teams also developed the Rio MP3 players and a music delivery platform and webstore backend service for selling music. In 1999, this music delivery system was spun out as a separate company called RioPort.com. I served on the RioPort.com board of directors and became involved in their product and technology strategy. I also managed engineering and marketing for the Rio product line for a period of time as an interim general manager. During my time at SonicBlue we launched more than 30 new consumer electronics products.

16. I served as a board member and technical advisor as KBGear Inc. from 1999-2001. KBGear Inc. designed and produced digital cameras and music players.

17. I have published more than fifty peer-reviewed papers in computer architecture and computer systems and IC design. I have also chaired IEEE and ACM conferences in microarchitecture and integrated circuit design and served as an associate editor for IEEE and ACM journals. I served on the IEEE Computer Society Awards committee. I am a Fellow of the

IEEE and a Distinguished Contributor to the IEEE Computer Society. I am a named inventor on at least fifty-seven U.S. patents and thirty-seven foreign patents.

18. I have been the invited keynote speaker at the ACM/IEEE International Symposium on Microarchitecture and at the International Conference on Multimedia. I have also been an invited speaker on various aspects of technology or the PC industry at numerous industry events including the Intel Developer's Forum, Microsoft Windows Hardware Engineering Conference, Microprocessor Forum, Embedded Systems Conference, Comdex, and Consumer Electronics Show as well as at the Harvard Business School and the University Illinois Law School. I have been interviewed on subjects related to technology and the electronics industry by publications such as the Wall Street Journal, New York Times, LA Times, Time, Newsweek, Forbes, and Fortune as well as CNN, NPR, and the BBC. I have also spoken at dozens of universities including MIT, Stanford, University of Texas, Carnegie Mellon, UCLA, University of Michigan, Rice University, and Duke University.

C. Materials Considered

19. In addition to relying on my extensive experience in the field of computer systems (summarized above), I have considered certain materials in formulating my opinions. Those materials include the exhibits listed in the Table of Materials Considered and Table of Materials Submitted with Declaration above.

II. LEGAL PRINCIPLES

A. Novelty and Obviousness

20. I am a technical expert and do not offer any legal opinions. However, I have been informed about certain legal principles regarding patentability and related matters under United States patent law, which I have applied in performing my analysis and arriving at my technical opinions in this matter.

21. I understand that, for *ex parte* reexamination, invalidity must be shown under a preponderance of the evidence standard, and this is the standard I have used throughout this Declaration. Further, I understand that each patent claim is considered separately for purposes of unpatentability.

22. I have been asked to evaluate novelty and obviousness as these legal concepts apply against certain claims of the '781 Patent. I am not a lawyer, and therefore, I have been

given certain instructions regarding the law of novelty and obviousness and have been asked to apply those principles in conducting my analysis. I have done so.

1. **General Framework**

23. I am informed that a person cannot obtain a patent on an invention if the differences between the invention and the prior art are such that the subject matter as a whole would have been obvious to a person having ordinary skill in the art at the time of the invention. I am informed that a conclusion of obviousness may be founded upon more than a single item of prior art. I am also informed that obviousness is determined by evaluating the following factors: (1) the scope and content of the prior art, (2) the differences between the prior art and the claim at issue, (3) the level of ordinary skill in the pertinent art, and (4) secondary considerations of non-obviousness. In addition, the obviousness inquiry should not be done in hindsight. Instead, the obviousness inquiry should be done through the eyes of a person having ordinary skill in the art at the time of the alleged invention (often referred to as a "PHOSITA" or "POSITA").

24. In considering whether certain prior art renders a particular patent claim non-novel or obvious, I am informed that one must consider the scope and content of the prior art, examples of which include disclosures in patents, trade publications, journal articles, conference papers, industry standards, product literature and documentation, texts describing competitive technologies, requests for comment published by standard setting organizations, and materials from industry conferences.

25. I am informed that, for a patent claim to be found obvious, the proper perspective to consider is that of a POSITA at the time of the alleged invention. I am informed that factors to consider in determining the level of ordinary skill in the art include (1) the educational level and experience of people working in the field at the time the invention was made, (2) the types of problems faced in the art and the solutions found to those problems, and (3) the sophistication of the technology in the field.

26. I understand that, to determine whether a claimed invention is patentably distinct, I must consider whether there are reasons why a POSITA would have concluded that the invention defined in the claim at issue would have been an obvious variation of the invention defined in a reference claim. I also understand that a claimed apparatus or device is not patentably distinct over a reference claim directed towards a method of using that apparatus or device.

27. I understand that even though the specification of the reference patent is not technically considered to be prior art, it may still be used to interpret the claims including the reference claim. I understand that the specification of the reference patent can be used as a dictionary to learn the meaning of a term in the reference claim. I also understand that, to determine if the claim at issue merely defines an obvious variation of what was earlier disclosed and claimed in the reference patent, portions of the specification of the reference patent that provide support for the reference claim may also be used to determine whether what is later-claimed in the claim at issue is an obvious variation of the reference claim.

28. I also understand that, because the question of whether the claim at issue is patentably distinct implicates the question of obviousness, it is also proper to consider what has been disclosed by the prior art and what is within the knowledge and skill of the POSITA.

29. I also understand that a claim is invalid under 35 U.S.C. § 103(a) (pre-AIA) if the subject matter described by the claim as a whole would have been obvious to a hypothetical POSITA in view of a prior art reference or in view of a combination of references at the time the claimed invention was made. Therefore, I understand that obviousness is determined from the perspective of a hypothetical POSITA and that the asserted claims of the patent should be read from the point of view of such a person at the time the claimed invention was made. I further understand that a hypothetical POSITA is assumed to know and to have all relevant prior art in the relevant field of endeavor.

30. I understand that, in considering whether an invention for a claimed combination would have been obvious, I may assess whether there are apparent reasons to combine known elements in the prior art in the manner claimed in view of interrelated teachings of multiple prior art references, the effects of demands known to the design community or present in the marketplace, and/or the background knowledge possessed by a POSITA. I understand that other principles may be relied on in evaluating whether a claimed invention would have been obvious, and that these principles include the following:

- A combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results;
- When a device or technology is available in one field of endeavor, design incentives and other market forces can prompt variations of it, either in the same field or in a

different one, so that if a person of ordinary skill can implement a predictable variation, the variation is likely obvious;

- If a technique has been used to improve one device, and a POSITA would recognize that it would improve similar devices in the same way, using the technique is obvious unless its actual application is beyond his or her skill;
- An explicit or implicit teaching, suggestion, or motivation to combine two prior art references to form the claimed combination may demonstrate obviousness, but proof of obviousness does not depend on or require showing a teaching, suggestion, or motivation to combine;
- Market demand, rather than scientific literature, can drive design trends and may show obviousness;
- In determining whether the subject matter of a patent claim would have been obvious, neither the particular motivation nor the avowed purpose of the named inventor controls;
- One of the ways in which a patent's subject can be proved obvious is by noting that there existed at the time of invention a known problem for which there was an obvious solution encompassed by the patent's claims;
- Any need or problem known in the field of endeavor at the time of invention and addressed by the patent can provide a reason for combining the elements in the manner claimed;
- "Common sense" teaches that familiar items may have obvious uses beyond their primary purposes, and in many cases a person of ordinary skill will be able to fit the teachings of multiple patents together like pieces of a puzzle;
- A POSITA is also a person of ordinary creativity, and is not an automaton;
- A patent claim can be proved obvious by showing that the claimed combination of elements was "obvious to try," particularly when there was a design need or market pressure to solve a problem and there were a finite number of identified, predictable solutions such that a POSITA would have had good reason to pursue the known options within his or her technical grasp; and
- One should be cautious of using hindsight in evaluating whether a claimed invention would have been obvious.

31. I further understand that, in making a determination as to whether or not the claimed invention would have been obvious to a person of ordinary skill, certain objective factors if they are present should be considered, such as: commercial success of products practicing the claimed invention; long-felt but unsolved need; teaching away; failure of others; unexpected results; copying; and praise by others in the field. These factors are generally referred to as "secondary considerations" or "objective indicia" of non-obviousness. I understand, however, that for such objective evidence to be relevant to the obviousness of a claim, there must be a causal relationship (called a "nexus") between the claim and the evidence, and that this nexus must be based on a novel element of the claim rather than something in the prior art. I also understand that, even when they are present, secondary considerations may be unable to overcome primary evidence of obviousness (e.g., motivation to combine with predictable results) that is sufficiently strong.

32. I understand that a claim is invalid under 35 U.S.C. § 102 if the subject matter recited in the claim was known by others or described in a printed publication before the claimed invention. I also understand that evidence of a claim's invalidity under 35 U.S.C. § 102 may be demonstrated by the descriptions of the claimed invention in the printed publication. I further understand that, pursuant to M.P.E.P. § 2131.01(B), a clarifying reference may be used to explain the meaning of a term in the printed publication for the purpose of invalidating a claim under 35 U.S.C. § 102.

2. **Analogous Art**

33. I am informed that for a prior art reference to be proper for use in an obviousness analysis, the reference must be "analogous art" to the claimed invention. I am informed that a reference is analogous art to the claimed invention if: (1) the reference is from the same field of endeavor as the claimed invention (even if it addresses a different problem), or (2) the reference is reasonably pertinent to the problem faced by the inventor (even if it is not in the same field of endeavor as the claimed invention). In order for a reference to be "reasonably pertinent" to the problem, it must logically have commended itself to an inventor's attention in considering his problem. In determining whether a reference is reasonably pertinent, one should consider the problem faced by the inventor, as reflected either explicitly or implicitly, in the specification. I believe that all of the references that I considered in forming my opinions in this reexamination request are well within the range of references a

person having ordinary skill in the art would have consulted by the time of the invention to address the type of problems described in the Challenged Claims.

3. Combining Known Elements in the Art, Common Sense

34. I am informed that, in order to establish that a claimed invention was obvious based on a combination of prior art elements, a clear articulation of the reason(s) why a claimed invention would have been obvious must be provided. Specifically, I am informed that, under the U.S. Supreme Court's **KSR** decision, a combination of multiple items of prior art can render a patent claim obvious when there was an apparent reason for a person having ordinary skill at the time of the invention to combine or modify the prior art. I am informed that such reason may include, but is not limited to, one or more of the following rationales: (A) combining prior art methods according to known methods to yield predictable results; (B) substituting one known element for another to obtain predictable results; (C) using a known technique to improve a similar device in the same way; (D) applying a known technique to a known device ready for improvement to yield predictable results; (E) trying a finite number of identified, predictable potential solutions, with a reasonable expectation of success; (F) identifying that known work in one field of endeavor may prompt variations of it for use in either the same field or a different one based on design incentives or other market forces if the variations are predictable to one of ordinary skill in the art; or (G) identifying an explicit teaching, suggestion, or motivation in the prior art that would have led one of ordinary skill to modify the prior art reference or to combine the prior art references to arrive at the claimed invention.

35. I am informed that the test for obviousness is not whether the features of a secondary reference may be bodily incorporated into the structure of the primary reference, but rather whether one of skill would have been motivated to combine the teachings of the prior art references to achieve the claimed invention. Nevertheless, I am informed that an invention that is a combination of prior art must do more than yield predictable results to be non-obvious and that, even where one or more motivations to combine exist, claims are not necessarily obvious unless a person having ordinary skill in the art would have had a reasonable expectation of success in implementing or applying the proposed combination at the time of the invention.

36. I also am informed that in conducting an obviousness analysis, a precise teaching directed to the specific subject matter of the challenged claim need not be sought out

because it is appropriate to take account of the inferences and creative steps that a person having ordinary skill in the art at the time of the invention would employ. In other words, the prior art need not be directed towards solving the same specific problem as the problem addressed by the patent. Further, the individual prior art references themselves need not all be directed towards solving the same problem. The prior art considered can be directed to any need or problem known in the field of endeavor at the time of invention and can provide a reason for combining the elements of the prior art in the manner claimed. Common sense teaches that familiar items may have obvious uses beyond their primary purposes, and a person having ordinary skill in the art will often have been able to fit the teachings of multiple prior art references like pieces of a puzzle. I am informed that, under the KSR standard, common sense is important and should be considered in assessing obviousness.

37. I also am informed that the fact that a particular combination of prior art elements was "obvious to try" may indicate that the combination was obvious even if no one attempted the combination. If the combination was obvious to try (regardless of whether it was actually tried) or leads to anticipated success, then it is likely the result of ordinary skill and common sense rather than innovation. I am further informed that in many fields it may be that there is little discussion of obvious techniques or combinations, and it often may be the case that market demand, rather than scientific literature or knowledge, will drive the design of an innovation. When there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person having ordinary skill in the art has good reason to pursue the known options within his or her technical grasp. If this leads to the anticipated success, it is likely the product not of innovation, but of ordinary skill and common sense.

4. **Teaching Away**

38. I am informed that it is improper to combine references where the references teach away from their combination. I am informed that a reference may be said to teach away when a person of skill, upon reading the reference, would be discouraged from following the path set out in the reference, or would be led in a direction divergent from the path that was taken by the patent applicant. In general, a reference will teach away if it suggests that the line of development flowing from the reference's disclosure is unlikely to be productive of the result sought by the patentee. I am informed that a reference teaches away, for example, if (1)

the combination would produce a seemingly inoperative device, or (2) the references leave the impression that the product would not have the property sought by the patentee. However, I am informed that a reference does not teach away if it merely expresses a general preference for an alternative invention but does not criticize, discredit, or otherwise discourage investigation into the invention claimed. Further, a person of skill will often appreciate that different courses of action often have simultaneous advantages and disadvantages, and I am informed that the existence of such does not necessarily obviate a motivation to combine.

5. **Secondary Considerations**

39. I am informed that even if a prima facie case of obviousness is established, the final determination of obviousness must also consider "secondary considerations" if presented. I am informed that, in most instances, the patentee raises these secondary considerations of non-obviousness. In that context, the patentee argues an invention would not have been obvious in view of these considerations, which include: (a) commercial success of a product due to the merits of the claimed invention; (b) a long-felt, but unsatisfied need for the invention; (c) failure of others to find the solution provided by the claimed invention; (d) deliberate copying of the invention by others; (e) unexpected results achieved by the invention; (f) praise of the invention by others skilled in the art; (g) lack of independent simultaneous invention within a comparatively short space of time; (h) teaching away from the invention in the prior art.

40. I am further informed that secondary-considerations evidence is only relevant if the offering party establishes a connection or nexus between the evidence and the claimed invention. The nexus cannot be based on prior art features. For example, if the commercial success of products allegedly embodying the claims is presented as a secondary indicium of non-obviousness, the patent owner must show that such success is due to features of the claimed invention and not features unrelated to the claimed invention or that already existed in the prior art. The establishment of a nexus is a question of fact. I have considered the secondary considerations that I am aware of at this time and determined that they do not impact my opinions.

B. Claim Construction

41. I am informed that, in performing an obviousness analysis, it is necessary to understand the scope of the claims. I am also informed that the first step in an unpatentability

analysis, therefore, often involves construing the claims. Second, the construed claim language is then compared to the disclosures of the prior art.

42. I have also been informed that the claims must be given their broadest reasonable interpretation consistent with the patent's specification. It is my understanding that what is to be considered includes the claims, the patent specifications and drawings, and the prosecution history, including any art listed by the examiner or the applicant. I understand that the broadest reasonable interpretation may differ from the Phillips standard, where claim terms are given their ordinary and custom meaning.

43. Counsel has informed me that the specification of the '781 Patent provides definitions for certain terms and phrases used in the claims. Therefore, I have applied these definitions here.

III. OPINION

A. Summary of My Opinions

44. It is my understanding that Requester submits that the prior art references raise substantial new questions of patentability because "the teaching of the (prior art) patents and printed publications is such that a reasonable examiner would consider the teaching to be important in deciding whether or not the claim is patentable." *See* Manual of Patent Examining Procedure (M.P.E.P.) § 2242. For at least the reasons discussed below, I agree with Requester.

B. Level of Ordinary Skill in the Art

45. I agree that a POSITA for the '781 Patent would have had, as of October 25, 2004: (1) at least a Bachelor's Degree in electrical engineering, electronics engineering, computer engineering, or a related scientific field; and (2) at least one to two years of experience in the computer science field, including chip design and software engineering, with additional education substituting for less experience and vice versa.

C. Background Technology

46. I understand that claims 1, 3, 7, 9, 10, 12-14, 16, 17, 19, 21-23, 25-27, 29, and 30 recite a "boot processor". In my opinion, a boot processor is any processor designated for handling bootstrap functions. In my experience, a boot processor can also be referred as a bootstrap

processor (BSP) or by a computer-readable designator (e.g., logical CPU 0). I agree that a boot processor is likely a part of any multiprocessor computer system.

1. **Boot Processor**

47. I agree that a boot processor is a critical device for any computing system and is likely included in any multiprocessor computing system. I agree that, in multi-processor (MP) system architectures, a boot processor may either be selected by default or through a selection process during bootup. I agree that, when all processors in an MP-compliant system are functionally identical, one of the processors will be designated as the bootstrap processor (BSP) at system initialization by the system hardware or by the system hardware in conjunction with the basic input/output system (BIOS).

48. I agree that, in some MP system architectures, a boot processor may be referred to by a logical, rather than a physical, position (e.g., logical CPU 0 or CPU L0 rather than physical CPU 0 or CPU P0) that is indicative of its designation for controlling other system hardware since the role of boot processor may often be assigned to a processor regardless of its physical position within a system. I agree that this interpretation of boot processor is consistent with the context provided by the '781 Patent.

49. In my opinion, the broadest reasonable interpretation for boot processor that is both "consistent with the ordinary and customary meaning of the term" and "consistent with the use of the term in the specification and drawings" is any processor designated for handling bootstrap functions. To the extent that the PTO does not agree with this, my opinions may also be applicable under similar interpretations.

50. I have applied the plain and ordinary meaning of all other claim terms in the Challenged Claims.

D. Challenged Claims Are Unpatentable

51. It is my opinion that claims 1-30 of the '781 Patent are unpatentable because they recite subject matter that would have been rendered obvious to a POSITA in view of Exhibits 1004-1008. The Request reflects my reasons for this opinion. I also elaborate on certain points that informed my reasons for this opinion below.

IV. GROUND 1: KALYANASUNDHARAM IN VIEW OF KENNEDY PRESENTS SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

A. Kalyanasundharam in view of Kennedy Renders Claims 1-6 and 9-15 of the '781 Patent Obvious to a POSITA

52. It is my opinion that Kalyanasundharam in view of Kennedy, together with the knowledge of a POSITA, discloses or renders obvious claims 1-6 and 9-15 of the '781 Patent. In addition to the opinions expressed in this Declaration, I have reviewed and agree with the corresponding obviousness contentions set forth in the Request.

1. Overview of Kalyanasundharam

53. An overview of Kalyanasundharam is provided in the Request. I have reviewed and agree with that overview of Kalyanasundharam.

54. For example, I agree that Kalyanasundharam is analogous art to the '781 Patent because Kalyanasundharam is from the same field of endeavor (*e.g.*, providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 (*e.g.*, processor failures and high availability in multiprocessor systems).

2. Overview of Kennedy

55. An overview of Kennedy is provided in the Request. I have reviewed and agree with that overview of Kennedy.

56. In my opinion, Kennedy is analogous art to the '781 Patent because Kennedy is from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (e.g., detecting and/or recovering from processor failures in multiprocessor systems). *See* Ex. 1005 at 2:15-21, 10:4-7; *see also* Ex. 1001 at 4:34-53, 3:16-18.

3. Overview of Motivations to Combine Kalyanasundharam and Kennedy

57. An overview of motivations to combine Kalyanasundharam and Kennedy are provided in the Request. I have reviewed and agree with those motivations to combine Kalyanasundharam and Kennedy.

58. I agree that a POSITA would have been motivated to combine the teachings of Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system (*e.g.*, in the event that the boot processor fails).

59. For example, in my opinion, a POSITA would have been motivated to modify Kalyanasundharam's system, which implements at least general processor failover, by Kennedy's specific teachings about boot processor failover.

60. In my opinion, a POSITA would have been motivated to do so because Kennedy's failover capabilities would provide Fox's system with additional fault tolerance and high availability (*e.g.*, in the event that the boot processor fails).

61. I also agree that a POSITA would have recognized the obvious benefits and advantages of enabling Kalyanasundharam's system to dynamically replace a failed boot processor, especially in view of Kennedy's express suggestion that doing so "provides fault tolerance and high availability[.]" *Kennedy* (Ex. 1005) at 2:21-22.

4. **Reasonable Expectation of Success**

62. The Request sets forth a reasonable expectation of success for the proposed combination of Kalyanasundharam and Kennedy. I have reviewed and agree with that reasonable expectation of success.

63. For example, I agree that Kalyanasundharam's failed primary processor could have been the boot processor of Kalyanasundharam's fault-tolerant system even though that detail is not explicitly stated in Kalyanasundharam.

64. In my opinion, because a boot processor was a common component in computer systems (*e.g.*, Kalyanasundharam's system) and Kennedy's specific teachings are merely directed to failing over the boot processor, the proposed combination of Kalyanasundharam and Kennedy would have been expected to succeed. In my opinion, a POSITA would have expected any multiprocessor system to include one or more boot processors.

65. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in the manner proposed, and Kalyanasundharam did not teach away from or discourage applying such a modification.

5. ***KSR* Rationales**

66. The Request sets forth various *KSR* rationales based on the combination of Kalyanasundharam and Kennedy in support of the conclusion of obviousness. I have reviewed and agree with those *KSR* rationales based on the combination of Kalyanasundharam and Kennedy.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

67. I agree that a POSITA would have combined the teachings of Kalyanasundharam and Kennedy according to known methods to yield predictable results.

68. For example, I agree that the combination of Kalyanasundharam and Kennedy disclosed each element recited in the applicable claims.

69. I agree that Kennedy disclosed a known method for transferring the role of boot processor to a spare processor. For example, in my opinion, a POSITA would have understood Kalyanasundharam as disclosing a boot processor and failing over the same to an auxiliary processor. However, to the extent that Kalyanasundharam is interpreted to not explicitly disclose a "boot processor", I agree that a POSITA would have combined Kalyanasundharam's fault-tolerant system with Kennedy's boot processor by the method disclosed in Kennedy to transfer the role of boot processor to Kalyanasundharam's auxiliary processor.

70. I also agree that each element from Kalyanasundharam and Kennedy performs the same function in isolation and in the combination. For example, I agree that Kalyanasundharam's fault-tolerant system in isolation replaces a failed primary processor with an auxiliary processor and that Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure. I further agree that a POSITA would have recognized the results of the proposed combination of Kalyanasundharam and Kennedy as being predictable. For example, in my opinion, a POSITA would have recognized that, by replacing Kalyanasundharam's failed primary processor with Kennedy's failed boot processor, Kalyanasundharam's system would have been able to predictably provide boot processor failover, as opposed to just primary processor failover.

Simple Substitution of One Known Element for Another to Obtain Predictable Results

71. I agree that a POSITA would have performed a simple substitution of Kalyanasundharam's processor with Kennedy's boot processor to obtain predictable results.

72. For example, I agree that, to the extent that Kalyanasundharam is not interpreted as explicitly disclosing a "boot processor," the substitution of Kalyanasundharam's failed primary processor with a failed boot processor would obviate the difference between Kalyanasundharam and the claimed device.

73. I also agree that the substituted components from Kennedy and their functions were known in the art. For example, in my opinion, the substituted boot processor from Kennedy and its functions were known in the art.

74. I further agree that a POSITA would have substituted one known element in Kalyanasundharam for another in Kennedy. For example, in my opinion, it was well within the skill of a POSITA to merely substitute one type of processor for another (*e.g.,* Kalyanasundharam's failed primary processor for Kennedy's failed boot processor).

75. In addition, I agree that a POSITA would have recognized the results of this substitution as being predictable. For example, in my opinion, a POSITA would have recognized that, by substituting Kalyanasundharam's failed primary processor for Kennedy's failed boot processor, Kalyanasundharam's device would have been able to predictably provide boot processor failover.

Use of Known Technique to Improve Similar Devices in the Same Way

76. I agree that, in the context of the proposed combination of Kalyanasundharam and Kennedy, a known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA.

77. I agree that Kalyanasundharam disclosed a base device upon which the claimed invention can be seen as an improvement. For example, I agree that, to the extent that Kalyanasundharam does not explicitly disclose applying its techniques specifically to a "boot processor," an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed primary processor.

78. I agree that the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention. For example, in my opinion, Kennedy disclosed a comparable device, method, or product that transfers control from a failed boot processor to an alternate processor. I agree that, because Kennedy's device transfers control from the failed boot processor to the alternate processor, Kennedy's device has been improved relative to Kalyanasundharam's device in the same way as the claimed invention.

79. I agree that a POSITA would have applied the known improvement technique in the same way to the base device. For example, in my opinion, a POSITA would have improved Kalyanasundharam's device by applying Kennedy's technique to fail over a boot processor, as opposed to failing over a primary processor.

80. I agree that a POSITA would have recognized the results of the applied improvement technique as being predictable. For example, in my opinion, a POSITA would have

recognized that, by applying Kennedy's known technique, Kalyanasundharam's device would have been able to predictably provide boot processor failover.

Applying a Known Technique to a Known Device Ready for Improvement

81. I agree that, in the context of the proposed combination of Kalyanasundharam and Kennedy, a known technique recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results.

82. I agree that Kalyanasundharam disclosed a base device upon which the claimed invention can be seen as an improvement (if Kalyanasundharam is interpreted to somehow limit its disclosure to processors other than "boot processors"). For example, in my opinion, an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed general processor, with a hot spare.

83. I agree that the prior art also disclosed a known technique that is applicable to the base device. For example, in my opinion, Kennedy disclosed a known technique for transferring control of a device from a failed boot processor to an alternate processor. I agree that, because Kennedy's known technique involves transferring control of the device from the failed boot processor to the alternate processor and Kennedy's device is very similar to and includes the same components and functionality as Kalyanasundharam's device, Kennedy's known technique is applicable to Kalyanasundharam's device.

84. I agree that a POSITA would have recognized that applying the known technique to the base device would have yielded predictable results. For example, in my opinion, a POSITA would have recognized that applying Kennedy's known technique to Kalyanasundharam's device would have yielded predictable results. More specifically, I agree that a POSITA would have recognized that, by applying Kennedy's known technique, Kalyanasundharam's device would have been able to predictably provide boot processor failover. Moreover, I agree that, because Kalyanasundharam already disclosed applying the same failover technique to processors in general, a POSITA would have reasonably expected the application of Kennedy's technique on Kalyanasundharam's device to be successful.

85. I agree that the predictable results would have improved the base device. For example, in my opinion, the predictable results would have improved the reliability of Kalyanasundharam's device (*e.g.,* in the event that the boot processor fails).

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to

Combine Prior-Art Teachings to Arrive at the Claimed Invention

86. I agree that one or more teachings, suggestions, or motivations in Kennedy would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy to arrive at the claimed invention.

87. I agree that Kennedy emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy in this way. The Request cites several examples of such teachings, suggestions, and/or motivations from Kennedy. I agree that any or all of those teachings, suggestions, and/or motivations cited in the Request would have led a POSITA to combine the teachings of Kalyanasundharam and Kennedy in this way.

88. I also agree that a POSITA would have had the knowledge to combine the teachings of Kalyanasundharam and Kennedy so as to enable Kalyanasundharam's system to dynamically replace a failed boot processor, as opposed to dynamically replacing a failed primary processor. I agree further that there was a reasonable expectation of success for the proposed combination of Kalyanasundharam and Kennedy. For example, in my opinion, Kalyanasundharam's general disclosure of a failed primary processor would have been understood to include the boot processor of Kalyanasundharam's system even though that detail is not explicitly stated in Kalyanasundharam. Moreover, in my opinion, Kalyanasundharam does not discourage or otherwise limit its disclosure to processors other than "boot processors," and it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in the manner proposed. For at least these reasons, I agree that the proposed combination of Kalyanasundharam and Kennedy, in which Kalyanasundharam's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed.

V. GROUND 2: SAFFORD IN VIEW OF KENNEDY PRESENTS SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

A. Safford in view of Kennedy Renders Claims 1-6 and 9-15 of the '781 Patent Obvious to a POSITA

89. It is my opinion that Safford in view of Kennedy, together with the knowledge of a POSITA, discloses or renders obvious claims 1-6 and 9-15 of the '781 Patent. In addition to the

opinions expressed in this Declaration, I have reviewed and agree with the corresponding obviousness contentions set forth in the Request.

1. Overview of Safford

90. An overview of Safford is provided in the Request. I have reviewed and agree with that overview of Safford.

91. In my opinion, Safford is analogous art to the '781 Patent because Safford is both from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) as the '781 Patent and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (e.g., system crashes caused by loss of lockstep). *See Safford* (Ex. 1004) at 3:21-22, 1:45-47; ***see also '781 Patent*** (Ex. 1001) at 3:16-18.

2. Overview of Motivations to Combine Safford and Kennedy

92. An overview of motivations to combine Safford and Kennedy are provided in the Request. I have reviewed and agree with those motivations to combine Safford and Kennedy.

93. I agree that a POSITA would have been motivated to combine the teachings of Safford and Kennedy to improve the reliability of Safford's system (*e.g.,* to provide high availability in the event that the boot processor fails). For example, in my opinion, a POSITA would have been motivated to instantiate Safford's system, which implements at least general processor failover, in a manner that specifically embodies Kennedy's teachings about boot processor failover. In my opinion, a POSITA would have been motivated to do so because Kennedy's boot-processor failover capabilities would provide Safford's system with additional fault tolerance and high availability (*e.g.,* in the event that the boot processor fails). I also agree that a POSITA would have recognized the obvious benefits and advantages of enabling Safford's system to dynamically replace a failed boot processor, especially in view of Kennedy's express suggestion that doing so "provides fault tolerance and high availability[.]" *Kennedy* (Ex. 1005) at 2:21-22.

3. Reasonable Expectation of Success

94. The Request sets forth a reasonable expectation of success for the proposed combination of Safford and Kennedy. I have reviewed and agree with that reasonable expectation of success.

95. For example, I agree that Safford's failed processor could have been the boot processor of Safford's system even though that detail is not explicitly stated in Safford since Safford's teachings apply to any processor in the system. In my opinion, because a boot processor was a common component in computer systems (*e.g.*, Safford's system) and Kennedy's teachings are specifically directed to failing over the boot processor, the proposed combination of Safford and Kennedy would be reasonably expected to succeed. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in the manner proposed, and Safford did not teach away from or discourage applying such a modification.

4. ***KSR* Rationales**

96. The Request sets forth various *KSR* rationales based on the combination of Safford and Kennedy in support of the conclusion of obviousness. I have reviewed and agree with those *KSR* rationales based on the combination of Safford and Kennedy.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

97. I agree that a POSITA would have combined the teachings of Safford and Kennedy according to known methods to yield predictable results. For example, I agree that the combination of Safford and Kennedy disclosed each element recited in the applicable claims. I agree that Kennedy disclosed a known method for transferring the role of boot processor to a spare processor. In my opinion, a POSITA would have understood Safford as disclosing a boot processor and failing over the same to a hot spare processor. However, to the extent that Safford is interpreted to not explicitly disclose a "boot processor", I agree that a POSITA would have combined Safford's system with Kennedy's teachings about boot processors using the method disclosed in Kennedy to transfer the role of boot processor to Safford's hot spare processor.

98. I also agree that each element from Safford and Kennedy performs the same function in isolation and in the combination. For example, I agree that Safford's system in isolation replaces a failed processor with a hot spare processor and that Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure.

99. I further agree that a POSITA would have recognized the results of the proposed combination of Safford and Kennedy as being predictable. For example, in my opinion, a POSITA would have recognized that, by replacing Safford's failed processor with Kennedy's failed boot processor, Safford's system would have been able to predictably provide boot processor failover, as opposed to being limited to non-boot processor failover.

Simple Substitution of One Known Element for Another to Obtain Predictable Results

100. I agree that a POSITA would have performed a simple substitution of Safford's processor with Kennedy's boot processor to obtain predictable results.

101. For example, I agree that, to the extent that Safford is not interpreted as explicitly disclosing a "boot processor," the substitution of Safford's failed processor with a failed boot processor would obviate the difference between Safford and the claimed device.

102. I also agree that the substituted components from Kennedy and their functions were known in the art. For example, in my opinion, the substituted boot processor from Kennedy and its functions were known in the art.

103. I further agree that a POSITA would have substituted one known element in Safford for another in Kennedy. For example, in my opinion, it was well within the skill of a POSITA to merely substitute one type of processor for another (e.g., Safford's failed processor for Kennedy's failed boot processor). I also agree that the teachings in Safford are not limited to non-boot processors, and when viewed in light of Kennedy's teachings about boot processors, it would be obvious to use a boot processor as taught in Kennedy in the system taught in Safford.

104. In addition, I agree that a POSITA would have recognized the results of this substitution as being predictable. For example, in my opinion, a POSITA would have recognized that, by substituting Safford's failed processor for Kennedy's failed boot processor, Safford's device would have been able to predictably provide boot processor failover using the mechanisms taught in Safford.

Use of Known Technique to Improve Similar Devices in the Same Way

105. I agree that, in the context of the proposed combination of Safford and Kennedy, a known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA.

106. I agree that Safford disclosed a base device upon which the claimed invention can be seen as an improvement. For example, I agree that, to the extent that Safford does not explicitly disclose applying its techniques specifically to a "boot processor," an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed general processor, with a hot spare.

107. I agree that the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention. For example, in my opinion, Kennedy disclosed a

comparable device, method, or product that transfers control from a failed boot processor to an alternate processor. I agree that, because Kennedy's device transfers control from the failed boot processor to the alternate processor, Kennedy's device has been improved relative to Safford's device in the same way as the claimed invention.

108. I agree that a POSITA would have applied the known improvement technique in the same way to the base device. For example, in my opinion, a POSITA would have improved Safford's device by applying Kennedy's technique to fail over a boot processor, as opposed to failing over a general processor.

109. I agree that a POSITA would have recognized the results of the applied improvement technique as being predictable. For example, in my opinion, a POSITA would have recognized that, by applying Kennedy's known technique, Safford's device would have been able to predictably provide boot processor failover.

Applying a Known Technique to a Known Device Ready for Improvement

110. I agree that, in the context of the proposed combination of Safford and Kennedy, a known technique recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results.

111. I agree that Safford disclosed a base device upon which the claimed invention can be seen as an improvement (if Safford is interpreted to somehow limit its disclosure to processors other than "boot processors"). For example, in my opinion, an improvement appears to be that the claimed invention replaces a failed boot processor, as opposed to a failed general processor, with a hot spare.

112. I agree that the prior art also disclosed a known technique that is applicable to the base device. For example, in my opinion, Kennedy disclosed a known technique for transferring control of a device from a failed boot processor to an alternate processor. I agree that, because Kennedy's known technique involves transferring control of the device from the failed boot processor to the alternate processor and Kennedy's device is very similar to and includes the same components and functionality as Safford's device, Kennedy's known technique is applicable to Safford's device.

113. I agree that a POSITA would have recognized that applying the known technique to the base device would have yielded predictable results. For example, in my opinion, a POSITA would have recognized that applying Kennedy's known technique to Safford's device would have

yielded predictable results. More specifically, I agree that a POSITA would have recognized that, by applying Kennedy's known technique, Safford's device would have been able to predictably provide boot processor failover. Moreover, I agree that, because Safford already disclosed applying the same failover technique to processors in general, a POSITA would have reasonably expected the application of Kennedy's technique on Safford's device to be successful.

114. I agree that the predictable results would have improved the base device. For example, in my opinion, the predictable results would have improved the reliability of Safford's device (*e.g.,* in the event that the boot processor fails).

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to Combine Prior-Art Teachings to Arrive at the Claimed Invention

115. I agree that one or more teachings, suggestions, or motivations in Kennedy would have led a POSITA to combine the teachings of Safford and Kennedy to arrive at the claimed invention.

116. I agree that Kennedy emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Safford and Kennedy in this way. The Request cites several examples of such teachings, suggestions, and/or motivations from Kennedy. I agree that any or all of those teachings, suggestions, and/or motivations cited in the Request would have led a POSITA to combine the teachings of Safford and Kennedy in this way. In my opinion, a POSITA would have used the teachings in Kennedy that suggest replacement of a failed boot processor as a motivation to use known art such as Safford as a specific mechanism and structure for implementing the replacement of a failed processor.

117. I also agree that a POSITA would have had the knowledge to combine the teachings of Safford and Kennedy so as to enable Safford's system to dynamically replace a failed boot processor, as opposed to dynamically replacing a failed general processor.

118. I agree further that there was a reasonable expectation of success for the proposed combination of Safford and Kennedy. For example, in my opinion, Safford's general disclosure of failed processors would have been understood to include the boot processor of Safford's system even though that detail is not explicitly stated in Safford. Moreover, in my opinion, Safford does not discourage or otherwise limit its disclosure to processors other than "boot processors," and it was well within the skill of a POSITA to modify Safford's system by Kennedy in the manner proposed. For at least these reasons, I agree that the proposed combination of Safford and Kennedy,

in which Safford's system is modified by Kennedy to achieve dynamic replacement of a failed boot processor, would have been reasonably expected to succeed.

VI. GROUND 3: KENNEDY IN VIEW OF NGUYEN AND FOX PRESENTS SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

A. Kennedy in view of Nguyen and Fox Renders Claims 1-30 of the '781 Patent Obvious to a POSITA

119. It is my opinion that Kennedy in view of Nguyen and Fox, together with the knowledge of a POSITA, discloses or renders obvious claims 1-30 of the '781 Patent. In addition to the opinions expressed in this Declaration, I have reviewed and agree with the corresponding obviousness contentions set forth in the Request.

1. Overview of Nguyen

120. An overview of Nguyen is provided in the Request. I have reviewed and agree with that overview of Nguyen.

121. For example, I agree that Nguyen is analogous art to the '781 Patent. For example, Nguyen is from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (*e.g.*, silent data corruption). ***See '781 Patent*** (Ex. 1001), 1:56-64, 3:16-18; *see also Nguyen* (Ex. 1007) ¶ [0004], [0006].

2. Overview of Fox

122. An overview of Fox is provided in the Request. I have reviewed and agree with that overview of Fox.

123. For example, I agree that Fox is analogous art to the '781 Patent because Fox is from the same field of endeavor (e.g., providing recovery from failure in a high-availability computing system) and is reasonably pertinent to a problem faced by the inventors of the '781 Patent (e.g., suspension of processing during the replacement of a failed processor).

3. Overview of Motivations to Combine Kennedy, Nguyen, and Fox

124. An overview of motivations to combine Kennedy, Nguyen, and Fox are provided in the Request. I have reviewed and agree with those motivations to combine Kennedy, Nguyen, and Fox.

125. In my opinion, a POSITA would have been motivated to modify Kennedy to fail over from the boot processor to the alternate processor in response to a loss of lockstep in the boot processor.

126. I believe that a POSITA would have recognized that "processors" and "cores" are corresponding components with respect to these references whose differences, if any exist, do not implicate the combination.

127. I agree that a POSITA would have been motivated to combine the teachings of Kennedy, Nguyen, and Fox to improve the reliability of Kennedy's system (*e.g.*, by failing over the boot processor to a hot spare processor in response to a loss of lockstep).

128. For example, in my opinion, a POSITA would have been motivated to modify Kennedy's system, which implements boot processor failover, by Nguyen's teachings about detecting a loss of lockstep.

129. In my opinion, a POSITA would have been motivated to do so because Nguyen's capabilities of initiating failover in response to a loss of lockstep would provide Kennedy's system with additional fault tolerance and high availability (*e.g.*, in the event that the boot processor fails during runtime).

130. I also agree that a POSITA would have recognized the obvious benefits and advantages of enabling Kennedy's system to dynamically fail over boot processors in response to a loss of lockstep, especially in view of Kennedy's express objective of "provid[ing] fault tolerance and high availability." *Kennedy* (Ex. 1005) at 2:21-22.

131. I believe that a POSITA would have been motivated to further modify Kennedy's system by Fox's teachings about a hot spare processor that is held idly in reserve.

132. In my opinion, a POSITA would have been motivated to do so because Fox's hot spare processor would enable Kennedy's system to seamlessly transfer executing processes from the failing master execution core to a hot spare processor without affecting the operating system (OS), the applications running on Nguyen's system, or the high availability of Kennedy's system.

133. I further agree that a POSITA would have recognized the obvious benefits and advantages of providing Kennedy's system with a hot spare processor, especially in view of Fox's express suggestion that "a failing processor is able to automatically fail-over to a hot-spare processor without affecting the OS or executing applications." *Fox* (Ex. 1006) at 4:20-23.

4. **Reasonable Expectation of Success**

134. The Request sets forth a reasonable expectation of success for the proposed combination of Kennedy, Nguyen, and Fox. I have reviewed and agree with that reasonable expectation of success.

135. I agree that the proposed combination of Kennedy and Nguyen would have been reasonably expected to succeed.

136. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen in the manner proposed, and Kennedy did not teach away from or discourage applying such a modification.

137. I agree that the proposed combination of Kennedy, Nguyen, and Fox would have been reasonably expected to succeed.

138. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in the manner proposed, and Kennedy did not teach away from or discourage applying such a modification.

5. ***KSR* Rationales**

139. The Request sets forth various *KSR* rationales based on the combination of Kennedy, Nguyen, and Fox in support of the conclusion of obviousness. I have reviewed and agree with those *KSR* rationales based on the combination of Kennedy, Nguyen, and Fox.

Combining Prior-Art Elements According to Known Methods to Yield Predictable Results

140. I agree that a POSITA would have combined the teachings of Kennedy, Nguyen, and Fox according to known methods to yield predictable results.

141. For example, I agree that the combination of Kennedy, Nguyen, and Fox disclosed each element recited in the applicable claims.

142. I agree that Kennedy disclosed a known method for transferring the role of boot processor to a spare processor, Nguyen disclosed a known method for detecting a loss of lockstep in a master execution core, and Fox disclosed a known method for replacing a failed processor with a hot spare processor. In my opinion, a POSITA would have combined Kennedy's system with Nguyen's known methods to transfer the role of boot processor to the alternate processor in response to a loss of lockstep for the boot processor. In my opinion, a POSITA would have further combined Kennedy's modified system with Fox's hot spare processor by the methods disclosed in Fox to perform the boot processor failover without affecting Kennedy's OS or corresponding applications.

143. I also agree that each element from Kennedy, Nguyen, and Fox performs the same function in isolation and in the combination. For example, I agree that, in isolation and in combination, Kennedy's system automatically transfers the role of boot processor to a spare processor in response to a detected failure, Nguyen's system detects a loss of lockstep in a master execution core, and Fox's system replaces a failed processor with a hot spare processor.

144. I further agree that a POSITA would have recognized the results of the proposed combination of Kennedy, Nguyen, and Fox as being predictable. For example, in my opinion, a POSITA would have recognized that, by failing over Kennedy's boot processor to Fox's hot spare processor in response to Nguyen's detection of a loss of lockstep, Kennedy's modified system would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep.

Simple Substitution of One Known Element for Another to Obtain Predictable Results

145. I agree that a POSITA would have performed a simple substitution of Kennedy's detection of a boot processor failure with Nguyen's detection of a loss of lockstep and a simple substitution of Kennedy's alternate processor with Fox's hot spare processor to obtain predictable results.

146. I agree that the substitution of Kennedy's detection of a boot processor failure with a detection of a loss of lockstep and the substitution of Kennedy's alternate processor with a hot spare processor would obviate the difference between Kennedy and the claimed device.

147. I also agree that the substituted components from Nguyen and Fox and their functions were known in the art. For example, in my opinion, the substituted detection of a loss of lockstep from Nguyen, the substituted hot spare processor from Fox, and their corresponding functions were known in the art.

148. I further agree that a POSITA would have substituted known elements in Kennedy for others in Nguyen and Fox. For example, in my opinion, it was well within the skill of a POSITA to merely substitute one type of trigger for boot processor failover for another (*e.g.,* Kennedy's boot processor failure for Nguyen's loss of lockstep) and to merely substitute one type of spare processor for another (*e.g.,* Kennedy's alternate processor for Fox's hot spare processor).

149. In addition, I agree that a POSITA would have recognized the results of this substitution as being predictable. For example, in my opinion, a POSITA would have recognized that, by substituting Kennedy's boot processor failure for Nguyen's loss of lockstep and Kennedy's

alternate processor for Fox's hot spare processor, Kennedy's modified device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep to ensure that a boot processor is always selected in case of a reboot and to decrease delays in the failover process.

Use of Known Technique to Improve Similar Devices in the Same Way

150. I agree that, in the context of the proposed combination of Kennedy, Nguyen, and Fox, a known technique had been used to improve similar devices in the same way, thereby making this technique part of the ordinary capabilities of a POSITA.

151. I agree that Kennedy disclosed a base device upon which the claimed invention can be seen as an improvement. For example, I agree that Kennedy disclosed a device that replaces the boot processor with an alternate processor in response to a failure of the boot processor. In my opinion, an improvement appears to be that the claimed invention in the '781 Patent fails over from the boot processor to the hot spare processor.

152. I agree that the prior art also disclosed a comparable device that has been improved in the same way as the claimed invention. For example, in my opinion, Nguyen disclosed a comparable device that detects a loss of lockstep, and Fox disclosed a comparable device that replaces a failed processor with a hot spare processor. I agree that, because Nguyen's device detects a loss of lockstep and Fox's device replaces the failed processor with a hot spare processor, Nguyen's and Fox's devices have been improved relative to Kennedy's device in the same ways as the claimed invention.

153. I agree that a POSITA would have applied the known improvement technique in the same way to the base device. For example, in my opinion, a POSITA would have improved Kennedy's device by applying Nguyen's technique to perform the failover in response to a loss of lockstep and by applying Fox's technique to fail over to a hot spare processor.

154. I agree that a POSITA would have recognized the results of the applied improvement technique as being predictable. For example, in my opinion, a POSITA would have recognized that, by applying Nguyen's and Fox's known techniques, Kennedy's device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep.

Applying a Known Technique to a Known Device Ready for Improvement

155. I agree that, in the context of the proposed combination of Kennedy, Nguyen, and Fox, a known technique recognized as part of the ordinary capabilities of a POSITA would have been applied to a known device ready for improvement to yield predictable results.

156. I agree that Kennedy disclosed a base device upon which the claimed invention can be seen as an improvement. In my opinion, an improvement appears to be that the claimed invention fails over from the boot processor to the hot spare processor in response to a loss of lockstep.

157. I agree that the prior art also disclosed a known technique that is applicable to the base device. For example, in my opinion, Nguyen described a known technique for detecting a functional redundancy checking (FRC) error (*e.g.,* loss of lockstep) in a master execution core. I also agree that Fox disclosed a known technique for replacing a failed processor with a hot spare processor. In my opinion, because Nguyen's known technique involves detecting a loss of lockstep and Fox's known technique involves replacing the failed processor with the hot spare processor, Nguyen's and Fox's known techniques would have been applicable to Kennedy's device, which is very similar to and includes the same components and functionality as Nguyen's and Fox's devices.

158. I agree that a POSITA would have recognized that applying the known technique to the base device would have yielded predictable results. For example, in my opinion, a POSITA would have recognized that applying Nguyen's and Fox's known techniques to Kennedy's device would have yielded predictable results. More specifically, I agree that a POSITA would have recognized that, by applying Nguyen's and Fox's known techniques, Kennedy's device would have been able to predictably fail over the boot processor to the hot spare processor in response to a loss of lockstep.

159. I agree that the predictable results would have improved the reliability of Kennedy's device (*e.g.,* by failing over the boot processor to a hot spare processor in response to a loss of lockstep). For example, in my opinion, Kennedy's device would become more reliable by failing over the boot processor to a hot spare processor such that the hot spare processor immediately assumes responsibility for booting the underlying system in case a reboot is needed. I believe that, by doing so, Kennedy's device as modified by Nguyen and Fox would avoid a situation in which the underlying system attempts to reboot without a designated, functional boot processor after a loss of lockstep.

Some Teaching, Suggestion, or Motivation in the Prior Art Would Have Led a POSITA to Combine Prior-Art Teachings to Arrive at the Claimed Invention

160. I agree that one or more teachings, suggestions, or motivations in Nguyen and Fox would have led a POSITA to combine the teachings of Kennedy, Nguyen, and Fox to arrive at the claimed invention.

161. I agree that Nguyen emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kennedy and Nguyen. I also agree that Fox emphasized and/or reiterated certain teachings, suggestions, and/or motivations that would have led a POSITA to combine the teachings of Kennedy and Fox. The Request cites several examples of such teachings, suggestions, and/or motivations from Nguyen and Fox. I agree that any or all of those teachings, suggestions, and/or motivations cited in the Request would have led a POSITA to combine the teachings of Kennedy, Nguyen, and Fox in this way.

162. I also agree that a POSITA would have had the knowledge to combine the teachings of Kennedy, Nguyen, and Fox so as to enable Kennedy's system to fail over the boot processor to a hot spare processor in response to a loss of lockstep.

163. I agree further that there was a reasonable expectation of success for the proposed combination of Kennedy, Nguyen, and Fox. For example, in my opinion, Kennedy as modified by Nguyen and Fox would have rendered obvious the technique of failing over a boot processor to a hot spare processor in response to a loss of lockstep. Moreover, in my opinion, Kennedy does not discourage or otherwise limit its disclosure to replacing the boot processor with an active spare processor in response to a non-lockstep error, and it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in the manner proposed. For at least these reasons, I agree that the proposed combination of Kennedy, Nguyen, and Fox—in which Kennedy's system is modified by Nguyen and Fox to achieve dynamically replacing a failed boot processor with a hot spare processor in response to a loss of lockstep—would have been reasonably expected to succeed.

VII. GROUND 4: SAFFORD IN LIGHT OF CEPULIS PRESENTS SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

A. Safford Discloses Each and Every Limitation of Claims 1 and 6 of the '781

164. It is my opinion that Safford discloses each and every limitation of claims 1 and 6 of the '781 Patent, especially in light of Cepulis' clarifying explanation of "logical CPU 0" disclosed in Safford. In addition to the opinions expressed in this Declaration, I have reviewed and agree with the contentions about Safford's disclosures under 35 U.S.C. § 102 as set forth in the Request.

1. Overview of Cepulis

165. An overview of Cepulis is provided in the Request. I have reviewed and agree with that overview of Cepulis.

166. In my opinion, Cepulis is analogous art to the '781 Patent because Cepulis is from the same field of endeavor (e.g., providing recovery in a high-availability multi-processor system) and is reasonably pertinent to the problem faced by the inventors of the '781 Patent. For example, I agree that the '781 Patent is directed to recovering from processor errors in multiprocessor systems in a manner that avoids crashing an operating system. *'781 Patent* (Ex. 1001) at 3:16-18. Similarly, I agree that Cepulis is directed to "a multiprocessor computer system [that] handles the failure of one or more of its processors without totally disabling the system." *Cepulis* (Ex. 1013), Abstract.

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

A. Proposed § 103 Rejections for the Challenged Claims

1. Ground 1: Kalyanasundharam in view of Kennedy

167. I agree that Kalyanasundharam in view of Kennedy renders claims 1-6 and 9-15 obvious to a POSITA for the reasons set forth in the Request. I identify evidence supporting my opinions regarding the obviousness of certain claims and/or limitations below. It is worth noting that certain claim limitations analyzed for obviousness in the Request are omitted from the claim set below for the purpose of brevity. In my opinion, the Request's obviousness analysis of those

claim limitations is sufficient and does not require my support to demonstrate a POSITA's understanding of the prior art relative to those claim limitations.

168. My opinions should be evaluated in view of the corresponding motivations to combine discussed above and the citations referenced in the Request. I have also reviewed and assisted in the preparation of Appendix A, which provides a detailed mapping of Kalyanasundharam and Kennedy to claims 1-6 and 9-15 that accurately reflects my opinions regarding obviousness.

169. **Claim 1:** ***[1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors*** (I agree that Kalyanasundharam described assigning the roles of primary processor core and auxiliary processor core to certain processors. *See Kalyanasundharam* (Ex. 1007) at 8:34-44, 4:66-5:2. For at least the reasons provided in this Declaration above and in the Request, I agree that a POSITA would have understood Kalyanasundharam as disclosing a boot processor and failing over the same to a hot spare processor. I agree that, even if Kalyanasundharam did not explicitly identify which processor was the boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In my opinion, Kalyanasundharam taught data storage that stores role assignments (*e.g.,* primary core and auxiliary core) of the processors, and Kennedy taught that one such assignment could be the role of boot processor and that data storage holds configuration information (*e.g.,* boot processor assignment). I agree that a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by storing information that assigns the roles of boot processor and spare processor in case of a reboot. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors.).

170. ***[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor*** (I agree that Kalyanasundharam described transferring the role of primary processor to

an auxiliary processor in response to a mismatch involving the primary processor. *See Kalyanasundharam* (Ex. 1007) at 13:15-29. I agree that, even if Kalyanasundharam did not explicitly identify the mismatched processor as having been the boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In view of the above citations, I agree that Kalyanasundharam taught logic that causes its system to replace the primary processor with an auxiliary processor in response to a mismatch involving the primary processor and that Kennedy taught logic that causes its system to fail over the boot processor to an alternate processor. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor.).

171. ***[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor*** ("In other embodiments, a fixed or **programmable number of the processor cores (less than the total number in the node) may participate** in redundant execution." *Kalyanasundharam* (Ex. 1007) at 11:35-38. "The configuration register 42 may further include a set of **core enables** which indicate **which cores are participating in redundant execution mode.**" *Id.* at 12:12-15. "The control circuit 84 may…**optionally disable a processor core or cores** from comparison in the redundant execution mode." *Id.* at 12:29-32; "Additionally, the core enables in the configuration register 42 may be used to **selectively enable various pairs** for redundant execution." *Id.* at 15:7-10; "If the primary processor core is programmably selected, **a different processor core may be selected as the primary processor core** and **the previous primary processor core may be disabled.**" *Id.* at 11:45-48. "For example, servers are implemented on **fault-tolerant systems** so that the servers may be available as close to 24 hours a day as possible. Any **mission-critical computing functions** may benefit from being assigned to

fault-tolerant computer systems." *Id.* at 1:14-19. I agree that, even if Kalyanasundharam did not explicitly use the term "hot spare" or the phrase "held in reserve" in connection with its processors, Kalyanasundharam described its processors as having the same functionality as the hot spare recited in claim 1. For example, in view of the above citations, I agree that Kalyanasundharam taught selectively enabling processors for participation, selectively disabling processors from participation, and maintaining configuration register 42 that tracks which processors are participating and which processors are not participating. For at least these reasons, I agree that a POSITA would have understood Kalyanasundharam as having taught that the auxiliary processor that assumes the role of primary processor could have been disabled prior to the primary processor's mismatch. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught that the processor assigned the role of spare could be held in reserve until a loss of lockstep is detected, thereby rendering this subject matter obvious to a POSITA.).

172. **Claim 2**: ***The system of claim 1 wherein said data storage is non-volatile data storage*** ("In the redundant execution mode, the communications from the primary processor core may be propagated and the communications from the auxiliary processor core(s) may be discarded…In this embodiment, **the PC bit from the configuration register 42** may identify the primary processor core." *Kalyanasundharam* (Ex. 1007) at 8:34-44. **"The configuration register 42** includes **a primary core (PC) field** encoded to **identify which of the processor cores 18A-18N is designated as the primary core."** *Id.* at 8:34-44. I agree that, even if Kalyanasundharam did not explicitly identify the configuration register as being non-volatile data storage, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 2 obvious to a POSITA. *See Kennedy* (Ex. 1005) at 5:15-18. From these teachings, I agree that a POSITA would have understood that the data storage that stores the information assigning the roles of boot processor and spare processor could have been non-volatile data storage. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 2 obvious to a POSITA.).

173. **Claim 3**: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor*** (I agree that, even if Kalyanasundharam did not explicitly teach a user interface that

receives input specifying the processor assigned the role of boot processor, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 3 obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with **a corresponding change in the boot ROM 150, as well known in the art.**" *Id.* In my opinion, it was well known in the art that the act of selecting in computer systems is often performed via user interfaces. Based on this common knowledge in the art, I agree that a POSITA would have understood Kennedy's teaching about selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. I agree that a POSITA would have been motivated to modify Kalyanasundharam by Kennedy and this this common knowledge in the art to enable a user to select which processor is to boot the system via a user interface. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA.).

174. **Claim 4**: ***The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor*** (I agree that, even if Kalyanasundharam did not explicitly teach a user interface that receives input specifying the processor assigned the role of boot processor, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 4 obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, "other **processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art." *Id.* In my opinion, it was well known in the art that the act of selecting in computer systems is often performed via user interfaces. Based on this common knowledge in the art, I agree that a POSITA would have understood Kennedy's teaching about selecting the processor to boot the system as including a spare processor selection (*e.g.,* one of the alternate processors not selected as the boot processor) made by input received via a user interface. I agree that a POSITA would have been motivated to modify Kalyanasundharam by Kennedy and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the

boot processor. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA.).

175. **Claim 5**: ***The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor*** ("The **nodes 12A-12B are coupled[.]**" *Kalyanasundharam* (Ex. 1007) at 2:30. "**The node 12A comprises processor cores 18A-18B coupled to a node controller 20A[.]**" *Id.* at 2:33-34. "**The node 12B similarly comprises processor cores 18C-18D coupled to a node controller 20B[.]**" *Id.* at 2:36-37. The node controller 20A may be configured to **propagate communications from one of the processor cores 18A-18B (e.g., from the primary processor core)** and to **discard communications from the other processor core 18A-18B (e.g., from the auxiliary processor core).**" *Id.* at 4:36-40. "**In** one embodiment, **each of the nodes 12A-12B may be a single integrated circuit chip** comprising the circuitry shown therein in FIG. 1. That is, **each node 12A-12B may be a chip multiprocessor (CMP).**" *Id.* at 2:46-49. In view of these citations, I agree that Kalyanasundharam taught nodes that each included a primary processor and an auxiliary processor, but only the primary processor propagates communications because the auxiliary processor's communications are discarded. Therefore, I agree that, under the broadest reasonable interpretation consistent with the '781 Patent, a POSITA would have understood Kalyanasundharam's nodes as each constituting a processor module, Kalyanasundharam's primary processor core as constituting a master processor, and Kalyanasundharam's auxiliary processor core as constituting a slave processor. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught processor modules that each include a master processor and a slave processor.).

176. **Claim 6**: ***The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor*** ("The control circuit 84 may use any selection mechanism. For example, the processor cores may be numbered (e.g., 0 to N-1), and the control circuit 84 may **increment** the number of the presently selected primary processor core **to select a new primary processor core**." *Kalyanasundharam* (Ex. 1007) at 13:29-33. "If one or more processor cores are disabled in the configuration register 42, the control circuit 84 may increment until **an enabled**

processor core is selected." *Id.* at 13:33-36. In view of these citations, I agree that Kalyanasundharam taught that the spare to the primary processor may be an enabled (*e.g.,* active) processor core selected by control circuit 84. I agree that, although Kalyanasundharam did not explicitly use the term "boot processor" when describing the primary processor, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 6 obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. Based on the supplementation of Kennedy's teachings, I agree that a POSITA would have understood Kalyanasundharam's primary processor to have constituted a boot processor for at least the same reasons provided above in connection with claim 1. Accordingly, Kalyanasundharam as modified by Kennedy taught that the spare processor is an active spare before being transferred the role of boot processor.).

177. **Claim 9**: ***[9.1] assigning a role of boot processor to one of a plurality of processors in a system*** (I agree that Kalyanasundharam described assigning the role of primary processor to one of processor cores 18A-18N. *See Kalyanasundharam* (Ex. 1007) at 8:34-44. For at least the reasons provided in this Declaration above and in the Request, I agree that a POSITA would have understood Kalyanasundharam as disclosing a boot processor and failing over the same to a hot spare processor. I agree that, even if Kalyanasundharam did not explicitly identify which processor was the boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In my opinion, Kalyanasundharam taught assigning the role of primary processor to a certain processor core, and Kennedy taught assigning the role of boot processor to one of its processors. I agree that a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by assigning the roles of boot processor and spare processor in case of a reboot. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught assigning the role of boot processor to one of its processor cores.).

178. ***[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor*** ("If the masked comparison results indicate that the communication from the primary processor core **mismatches** with all auxiliary processor cores (that are enabled)…the control circuit 84 may

disable the primary processor core and **select one of the auxiliary processor cores to be the primary processor core** (block 102)." *Kalyanasundharam* (Ex. 1007) at 13:15-22. "As with the discussion above for **disabling the auxiliary processor core**, the control circuit 83 may simply **disable the primary processor core from comparison** by clearing the enable bit in the configuration register 42, or may **disable the primary processor core from executing at all**." *Id.* at 13:22-27. "The control circuit 84 may also **update the PC field of the configuration register 42** to reflect the **newly selected primary processor core.**" *Id.* at 13:27-29. Accordingly, I agree that Kalyanasundharam described changing the role of primary processor to an auxiliary processor in response to a mismatch (*e.g.*, loss of lockstep) involving the primary processor. I also agree that, even if Kalyanasundharam did not explicitly identify the mismatched processor as having been the boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In view of the above citations, I agree that Kalyanasundharam taught changing the role of primary processor to the spare processor in response to a loss of lockstep involving the primary processor and that Kennedy taught changing the role of boot processor to an alternate processor. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught changing the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor.).

179. ***[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system*** (I agree that, even if Kalyanasundharam did not explicitly teach an operating system that recognizes the spare processor as the boot processor after the role change, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "a **central system boot ROM 150** accessible by all processors…may be **used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he CPU1 120 controls the system again and begins the

operating system boot process." *Id.* at 9:53-55. "**The alternate boot processors**…can execute the **boot code** in the **central boot ROM 150**". *Id.* at 10:42-45. I agree that a POSITA would have understood that, because Kennedy's central boot ROM 150 is used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system, the operating system must recognize the spare processor as the boot processor after failover. In my opinion, at the very least, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 7 obvious to a POSITA due at least in part to Kennedy's central boot ROM 150 being used by the operating system and storing the boot code executed in conjunction with the operating system to boot the system. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy would have rendered obvious to a POSITA an operating system that recognizes the spare processor as the boot processor without requiring shutdown.).

180. **Claim 12**: ***The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor*** (I agree that, even if Kalyanasundharam did not explicitly teach determining a processor that is coupled to a common communication bus with the boot processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, I agree that Kennedy taught that the boot processor and the alternate boot processors are installed in specific slots along the same backplane bus. Based on this teaching from Kennedy, I agree that a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. As a result, I agree that any form of identifying such an alternate processor would constitute a determination of an alternate processor coupled to a common communication bus with the boot processor. In my opinion, a POSITA would have been motivated to combine these teachings from Kennedy with Kalyanasundharam to improve the communication capabilities of Kalyanasundharam's system for fault tolerance and high availability. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to