US007627781B2

(12) **United States Patent**
**Michaelis et al.**

(10) **Patent No.: US 7,627,781 B2**
(45) **Date of Patent: *Dec. 1, 2009**

(54) **SYSTEM AND METHOD FOR ESTABLISHING A SPARE PROCESSOR FOR RECOVERING FROM LOSS OF LOCKSTEP IN A BOOT PROCESSOR**

(75) Inventors: **Scott L. Michaelis**, Plano, TX (US); **Anurupa Rajkumari**, Round Rock, 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 520 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **10/973,077**

(22) Filed: **Oct. 25, 2004**

(65) **Prior Publication Data**

US 2006/0107112 A1 May 18, 2006

(51) **Int. Cl.**
***G06F 11/00*** (2006.01)

(52) **U.S. Cl.** ..... **714/13**; 714/11

(58) **Field of Classification Search** ..... 714/10–12, 714/13, 31; 712/31
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 4,358,823 | A | 11/1982 | McDonald et al. | |
| 5,226,152 | A | 7/1993 | Klug et al. | |
| 5,249,188 | A | 9/1993 | McDonald | |
| 5,537,655 | A | 7/1996 | Truong | |
| 5,675,794 | A | 10/1997 | Meredith | |
| 5,751,932 | A | 5/1998 | Horst et al. | |
| 5,758,058 | A | 5/1998 | Milburn | |
| 5,764,660 | A | 6/1998 | Mohat | |
| 5,896,523 | A | 4/1999 | Bissett et al. | |
| 5,915,082 | A * | 6/1999 | Marshall et al. | 714/11 |
| 5,991,900 | A | 11/1999 | Garnett | |
| 6,065,135 | A | 5/2000 | Marshall et al. | |
| 6,141,770 | A | 10/2000 | Fuchs | |
| 6,148,348 | A | 11/2000 | Garnett et al. | |
| 6,220,581 | B1 | 4/2001 | Mueller | |
| 6,263,452 | B1 | 7/2001 | Jewett | |
| 6,330,356 | B1 | 12/2001 | Sundareswaran | |
| 6,438,687 | B2 * | 8/2002 | Klein | 713/1 |
| 6,473,869 | B2 | 10/2002 | Bissett et al. | |
| 6,560,682 | B1 | 5/2003 | Miller et al. | |
| 6,604,177 | B1 | 8/2003 | Kondo et al. | |

(Continued)

FOREIGN PATENT DOCUMENTS

| | | |
|---|---|---|
| WO | WO 01 45421 | 6/2001 |

OTHER PUBLICATIONS

"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".

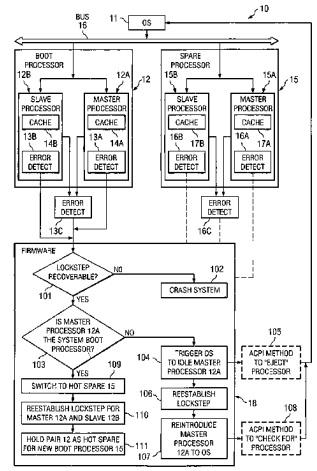
(Continued)

*Primary Examiner*—Dieu-Minh Le

(57) **ABSTRACT**

A system comprises a plurality of processors, and data storage storing information that assigns a role of boot processor to one of the plurality of processors and assigns a role of spare processor to another of the plurality of processors. The system further comprises logic operable, responsive to detecting loss of lockstep for the boot processor, for transferring, during system runtime, the role of boot processor to the spare processor.

**30 Claims, 4 Drawing Sheets**






BMW EXHIBIT 1034

U.S. PATENT DOCUMENTS

| | | | | |
|---|---|---|---|---|
| 6,625,749 | B1 * | 9/2003 | Quach | 714/10 |
| 6,643,387 | B1 | 11/2003 | Sethuraman | |
| 6,675,324 | B2 | 1/2004 | Marisetty et al. | |
| 6,687,851 | B1 | 2/2004 | Somers et al. | |
| 6,735,253 | B1 | 5/2004 | Chang | |
| 6,754,787 | B2 | 6/2004 | Miller et al. | |
| 6,920,581 | B2 * | 7/2005 | Bigbee et al. | 714/10 |
| 6,948,092 | B2 | 9/2005 | Kondo et al. | |
| 6,983,018 | B1 | 1/2006 | Chih-Lung | |
| 7,003,691 | B2 | 2/2006 | Safford et al. | |
| 7,020,800 | B2 | 3/2006 | Fu et al. | |
| 7,042,943 | B2 | 5/2006 | Haskell | |
| 7,085,959 | B2 * | 8/2006 | Safford | 714/11 |
| 7,134,047 | B2 | 11/2006 | Quach | |
| 7,155,721 | B2 | 12/2006 | Safford et al. | |
| 7,191,292 | B2 | 3/2007 | Chaudhry et al. | |
| 7,225,355 | B2 | 5/2007 | Yamazaki et al. | |
| 7,237,144 | B2 | 6/2007 | Safford et al. | |
| 7,251,746 | B2 * | 7/2007 | Fox et al. | 714/13 |
| 2002/0144177 | A1 | 10/2002 | Kondo et al. | |
| 2002/0152420 | A1 | 10/2002 | Chaudhry et al. | |
| 2003/0051190 | A1 | 3/2003 | Marisetty et al. | |
| 2003/0070050 | A1 | 4/2003 | Miller et al. | |
| 2003/0126498 | A1 | 7/2003 | Bigbee | |
| 2003/0135711 | A1 | 7/2003 | Shoemaker | |
| 2003/0140281 | A1 | 7/2003 | Fu et al. | |
| 2004/0006722 | A1 | 1/2004 | Safford | |
| 2004/0019771 | A1 | 1/2004 | Quach | |
| 2004/0078650 | A1 | 4/2004 | Safford et al. | |
| 2004/0078651 | A1 * | 4/2004 | Safford et al. | 714/11 |
| 2004/0153857 | A1 | 8/2004 | Yamazaki et al. | |
| 2005/0172164 | A1 * | 8/2005 | Fox et al. | 714/13 |
| 2005/0240806 | A1 | 10/2005 | Bruckert et al. | |
| 2005/0240811 | A1 * | 10/2005 | Safford et al. | 714/11 |
| 2005/0240829 | A1 | 10/2005 | Safford et al. | |
| 2006/0085677 | A1 | 4/2006 | Safford et al. | |
| 2006/0136672 | A1 | 6/2006 | Chaudhry et al. | |
| 2006/0248384 | A1 * | 11/2006 | Safford | 714/11 |
| 2006/0248684 | A1 | 11/2006 | Vosbikian et al. | |

OTHER PUBLICATIONS

"System and Method for Reintroducing a Processor Module to an Operating System After Lockstep Recovery".
"System and Method for Using Information Relating to a detected Loss of Lockstep for Determining a Responsive Action".
"System and Method for Providing Firmware Recoverable Lockstep Protection".
"System and Method for Configuring Lockstep Mode of a Prcessor Module".
"System and Method for Reestablishing Lockstep for a Processor Module for Which Loss of Lockstep Is Detected".
"System and Method for System Firmware Causing an Operating System to Idle a Processor".
"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".
Notice of Allowance dated Feb. 12, 2009 for U.S. Appl. No. 10/972,588 in the United States.
Notice of Allowance dated Sep. 23, 2008 for U.S. Appl. No. 10/972,588 in the United States.
Office Action dated Mar. 10, 2008 for U.S. Appl. No. 10/972,588 in the United States.
Office Action dated Sep. 12, 2007 for U.S. Appl. No. 10/972,588 in the United States.
Notice of Allowance for U.S. Appl. No. 10/972,835, dated Nov. 20, 2008 in the United States.
Notice of Allowance for U.S. Appl. No. 10/972,835, dated Aug. 1, 2008 in the United States.
Office Action for U.S. Appl. No. 10/972,835, dated Mar. 3, 2008 in the United States.
Office Action for U.S. Appl. No. 10/972,835, dated Jul. 12, 2007 in the United States.
Office Action for U.S. Appl. No. 10/973,003, dated Mar. 5, 2008 in the United States.
Office Action for U.S. Appl. No. 10/973,003, dated Sep. 12, 2007 in the United States.
Notice of Allowance for U.S. Appl. No. 10/973,076, dated Nov. 28, 2008 in the United States.
Notice of Allowance for U.S. Appl. No. 10/973,076, dated Jul. 16, 2008 in the United States.
Notice of Allowance for U.S. Appl. No. 10/973,076, dated Jun. 3, 2008 in the United States.
Final Office Action for U.S. Appl. No. 10/973,076, dated Feb. 27, 2008 in the United States.
Office Action for U.S. Appl. No. 10/973,076, dated Sep. 12, 2007 in the United States.
Notice of Allowance for U.S. Appl. No. 10/972,888, dated Nov. 14, 2007 in the United States.
Office Action for U.S. Appl. No. 10/972,888, dated Jul. 18, 2007 in the United States.
Notice of Allowance for U.S. Appl. No. 10/973,004, dated Aug. 1, 2007 in the United States.
Office Action for U.S. Appl. No. 10/973,004, dated Apr. 10, 2007 in the United States.
Final Office Action for U.S. Appl. No. 10/973,075, dated May 12, 2009 in the United States.
Office Action for U.S. Appl. No. 10/973,075, dated Oct. 29, 2008 in the United State.
Office Action for U.S. Appl. No. 10/973,075, dated Mar. 12, 2008 in the United States.
Office Action for U.S. Appl. No. 10/973,075, dated Jul. 2, 2007 in the United States.
Notice of Allowance for U.S. Appl. No. 10/972,796, dated Dec. 19, 2007 in the United States.
Office Action for U.S. Appl. No. 10/972,796, dated Jul. 12, 2007 in the United States.
Shoichi Araki, Real-Time Tracking of Multiple Moving Object Contours in a Moving Camera Image Sequence, vol. E83-D IEICE Trans. Inf. & Syst, Jul. 7, 2000, 9 pages.
Dong Kyu Kim, Active Object Tracking Using Image Mosaic Background, 4 pages.
Jiang-Bin Zheng, Proceedings of the First International Conference on Machine Learning and Cybernetics, Beijing, Nov. 4-5, 2002, 4 pages.
Jong Ryul Kim, Automatic Localization and Trackign of Moving Objects Using Adaptive Snake Algorithm, ICICS-PCM 2003, Dec. 15-18, 2003, 5 pages, Singapore.
Jianhoa Meng, Cveps-A Compressed Video Editing and Parsing System, Nov. 18, 1996, pp. 1-11.
Final Office Action for U.S. Appl. No. 10/972,835, dated Nov. 21, 2007 in the United States.
Office Action for U.S. Appl. No. 10/973,075, dated Dec. 13, 2007 in the United States.
Notice of Allowance, dated Jul. 14, 2009, U.S. Appl. No. 10/972,588, 12 pages.

* cited by examiner

*FIG. 1*

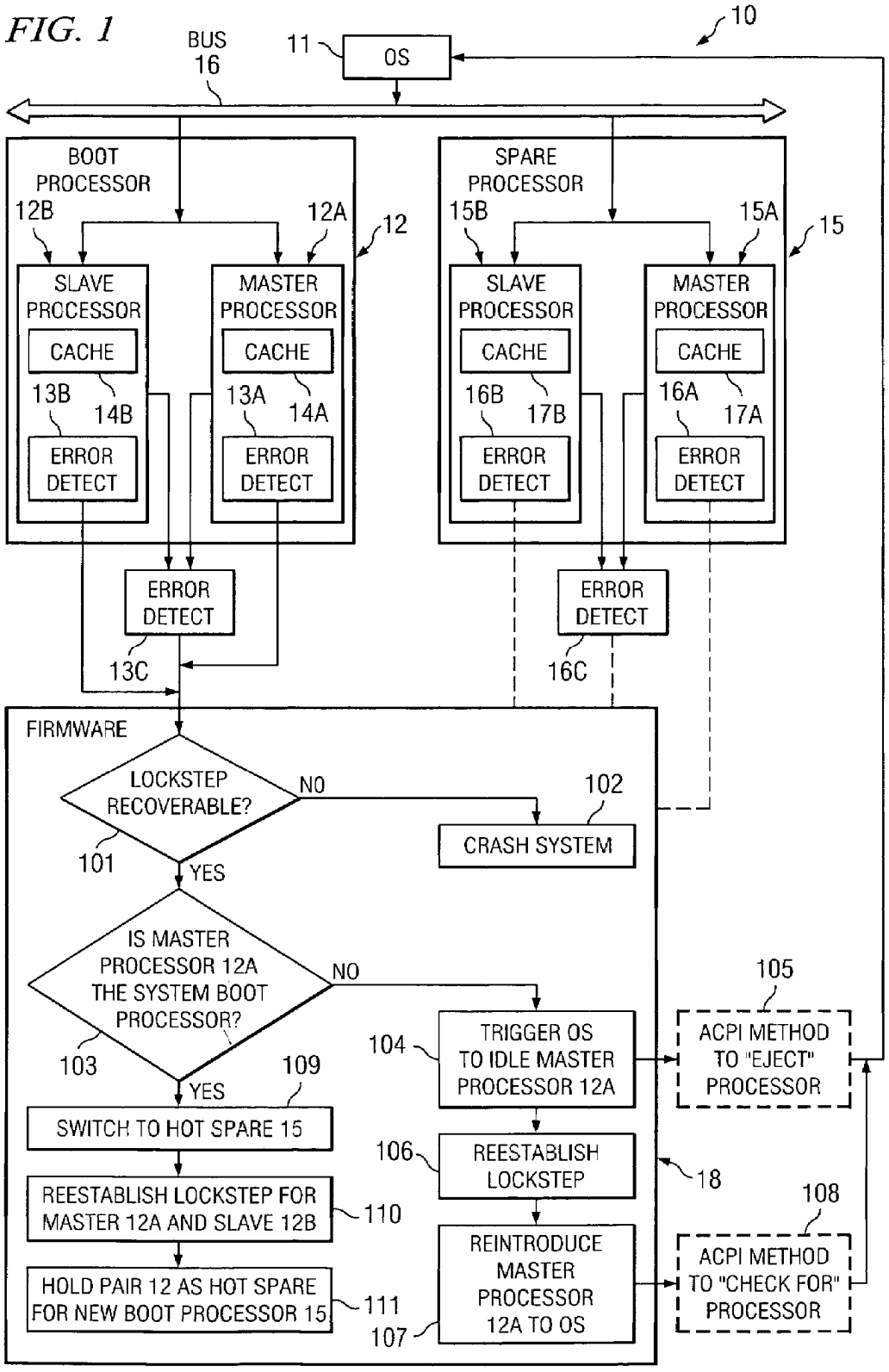

10
BUS
16
11
OS
BOOT PROCESSOR
12B
12A
12
SLAVE PROCESSOR
CACHE
13B
14B
ERROR DETECT
MASTER PROCESSOR
CACHE
13A
14A
ERROR DETECT
SPARE PROCESSOR
15B
15A
15
SLAVE PROCESSOR
CACHE
16B
17B
ERROR DETECT
MASTER PROCESSOR
CACHE
16A
17A
ERROR DETECT
ERROR DETECT
13C
ERROR DETECT
16C
FIRMWARE
LOCKSTEP RECOVERABLE?
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*

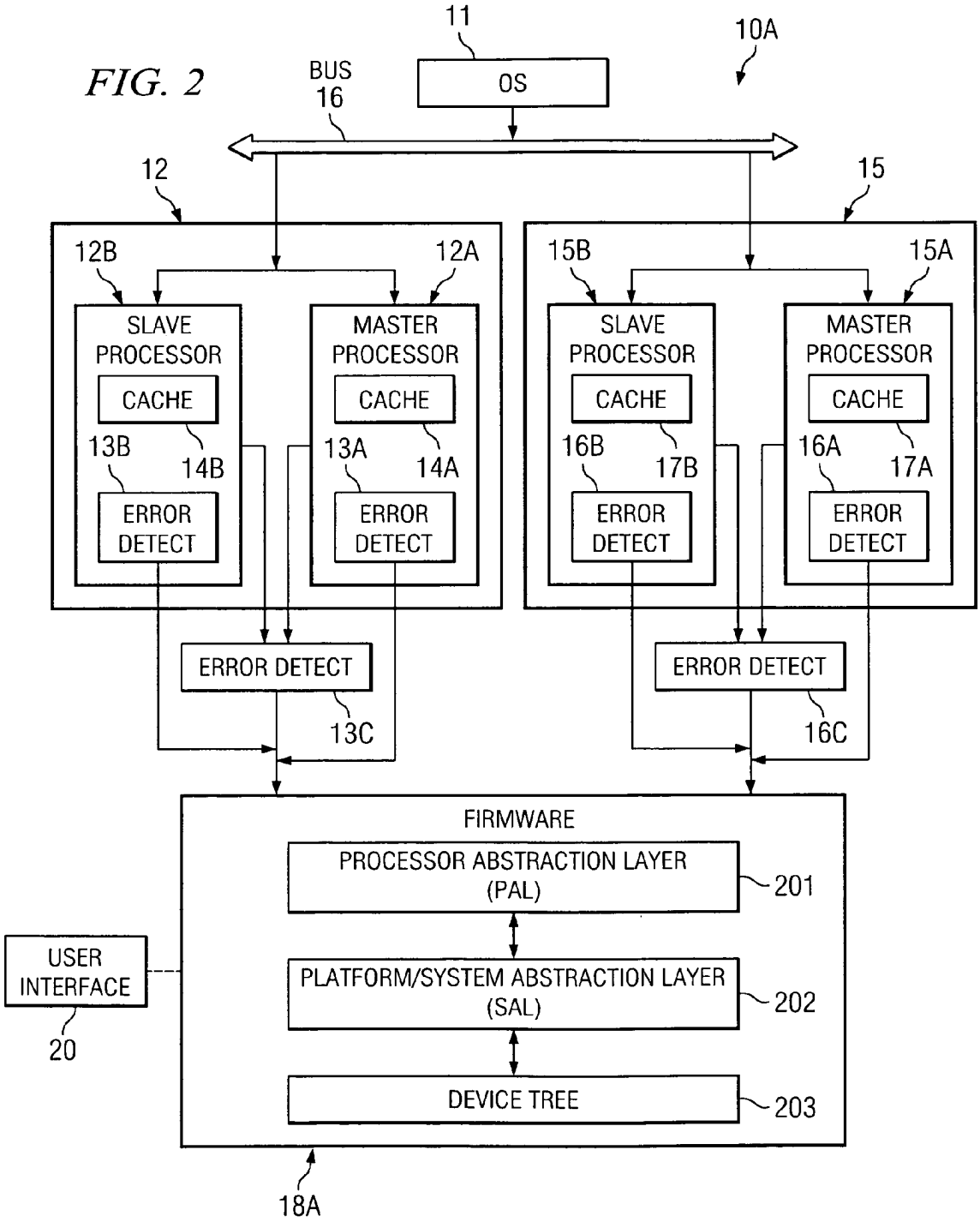

11
10A
OS
BUS
16
12
15
12B
12A
15B
15A
SLAVE PROCESSOR
MASTER PROCESSOR
CACHE
13B
14B
13A
14A
16B
17B
16A
17A
ERROR DETECT
ERROR DETECT
13C
16C
FIRMWARE
PROCESSOR ABSTRACTION LAYER (PAL)
201
PLATFORM/SYSTEM ABSTRACTION LAYER (SAL)
202
DEVICE TREE
203
USER INTERFACE
20
18A

*FIG. 3*

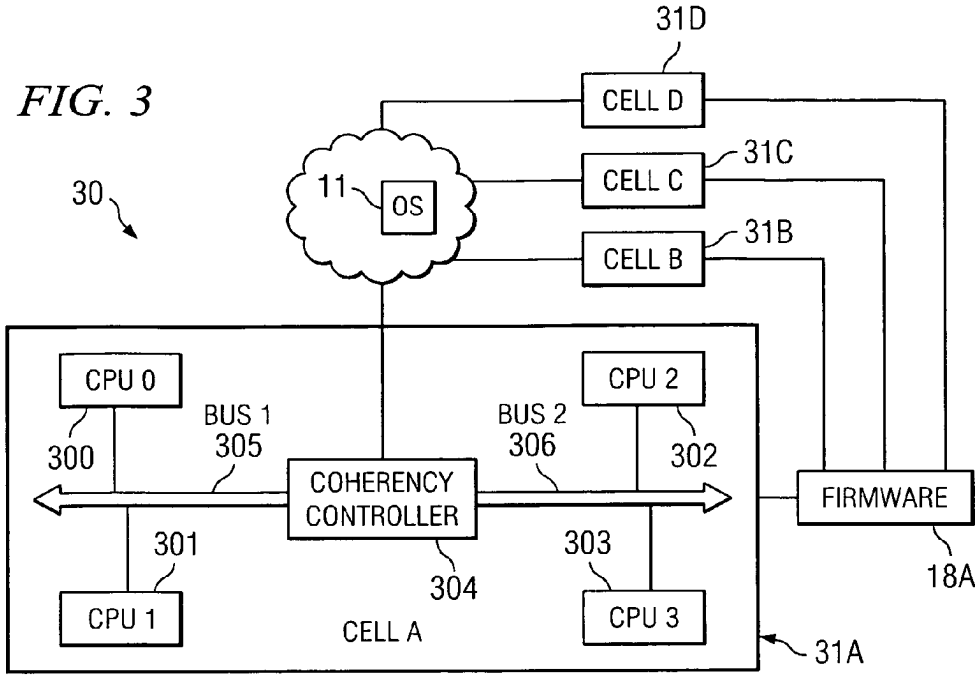



*FIG. 4*

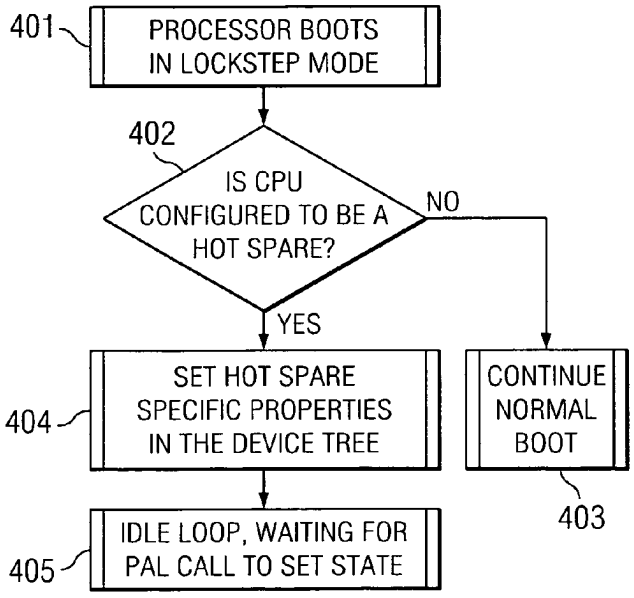



*FIG. 6*

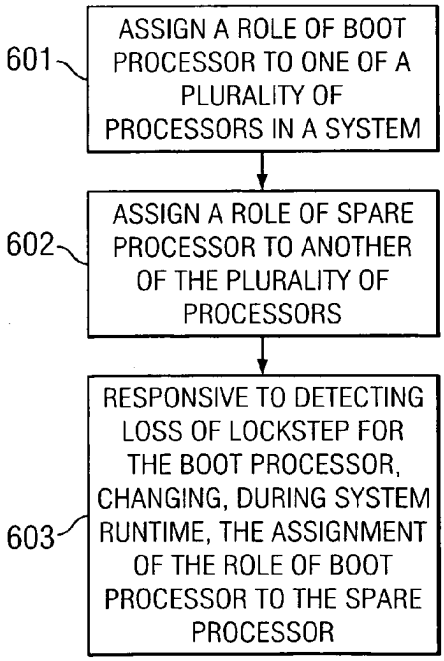
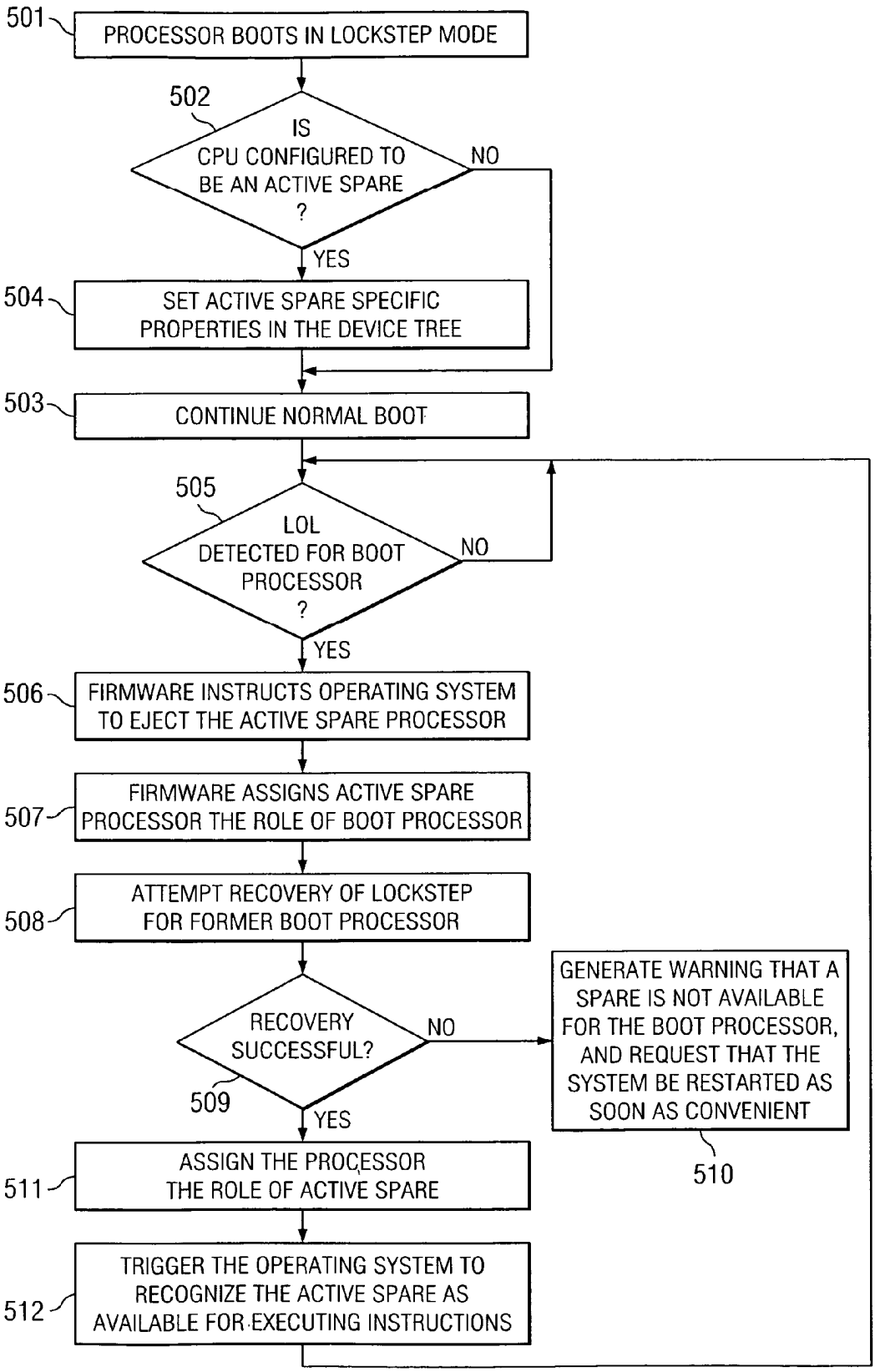


*FIG. 5*


501
PROCESSOR BOOTS IN LOCKSTEP MODE
502
IS CPU CONFIGURED TO BE AN ACTIVE SPARE ?
NO
YES
504
SET ACTIVE SPARE SPECIFIC PROPERTIES IN THE DEVICE TREE
503
CONTINUE NORMAL BOOT
505
LOL DETECTED FOR BOOT PROCESSOR ?
NO
YES
506
FIRMWARE INSTRUCTS OPERATING SYSTEM TO EJECT THE ACTIVE SPARE PROCESSOR
507
FIRMWARE ASSIGNS ACTIVE SPARE PROCESSOR THE ROLE OF BOOT PROCESSOR
508
ATTEMPT RECOVERY OF LOCKSTEP FOR FORMER BOOT PROCESSOR
RECOVERY SUCCESSFUL?
NO
509
YES
GENERATE WARNING THAT A SPARE IS NOT AVAILABLE FOR THE BOOT PROCESSOR, AND REQUEST THAT THE SYSTEM BE RESTARTED AS SOON AS CONVENIENT
510
511
ASSIGN THE PROCESSOR THE ROLE OF ACTIVE SPARE
512
TRIGGER THE OPERATING SYSTEM TO RECOGNIZE THE ACTIVE SPARE AS AVAILABLE FOR EXECUTING INSTRUCTIONS

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

## CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to the following concurrently filed and commonly assigned U.S. Patent Applications: U.S. patent application Ser. No. 10/972,835 (U.S. Published Patent Application 20060107114) titled "SYSTEM AND METHOD FOR USING INFORMATION RELATING TO A DETECTED LOSS OF LOCKSTEP FOR DETERMINING A RESPONSIVE ACTION"; U.S. patent application Ser. No. 10/973,004 (U.S. Published Patent Application 20060107117, now U.S. Pat. No. 7,308,566) titled "SYSTEM AND METHOD FOR CONFIGURING LOCKSTEP MODE OF A PROCESSOR MODULE"; U.S. patent application Ser. No. 10/973,076 (U.S. Published Patent Application 20060107107) titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION"; U.S. patent application Ser. No. 10/972,588 (U.S. Published Patent Application 20060090064) titled "SYSTEM AND METHOD FOR SWITCHING THE ROLE OF BOOT PROCESSOR TO A SPARE PROCESSOR RESPONSIVE TO DETECTTON OF LOSS OF LOCKSTEP IN A BOOT PROCESSOR"; U.S. patent application Ser. No. 10/973,003 (U.S. Published Patent Application 20060107116) titled "SYSTEM AND METHOD FOR REESTABLISHING LOCKSTEP FOR A PROCESSOR MODULE FOR WHICH LOSS OF LOCKSTEP IS DETECTED"; U.S. patent application Ser. No. 10/972,888 (U.S. Published Patent Application 20060107115) titled "SYSTEM AND METHOD FOR SYSTEM FIRMWARE CAUSING AN OPERATING SYSTEM TO IDLE A PROCESSOR"; U.S. patent application Ser. No. 10/973,075 (U.S. Published Patent Application 20060107111) titled "SYSTEM AND METHOD FOR REINTRODUCING A PROCESSOR MODULE TO AN OPERATING SYSTEM AFTER LOCKSTEP RECOVERY"; and U.S. patent application Ser. No. 10/972,796 (U.S. Published Patent Application 20060107106) 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

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).

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.

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.

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.

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.

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.

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

According to at least one embodiment, a system comprises a plurality of processors, and data storage storing information that assigns a role of boot processor to one of the plurality of processors and assigns a role of spare processor to another of the plurality of processors. The system further comprises logic operable, responsive to detecting loss of lockstep for the boot processor, for transferring, during system runtime, the role of boot processor to the spare processor.

According to at least one embodiment, a method comprises 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 the plurality of processors; and responsive to detecting loss of lockstep for the boot processor, changing, during system runtime, the assignment of the role of boot processor to the spare processor.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows an example multi-processor system in which firmware utilizes a spare processor for recovering from LOL for the system's boot processor;

FIG. **2** shows a block diagram of one embodiment implemented for the IA-64 processor architecture;

FIG. **3** shows an example system having multi-processor cells in which an embodiment for using firmware for recovering from LOL may be employed;

FIG. **4** shows an operational flow diagram for establishing a hot spare for the boot processor according to one embodiment;

FIG. **5** shows an operational flow diagram for establishing an active spare for the boot processor and assigning such active spare the role of boot processor responsive to detecting LOL for the boot processor in accordance with one embodiment; and

FIG. **6** shows a general flow diagram for operation of certain embodiments.

## DETAILED DESCRIPTION

As described further herein and in concurrently filed and commonly assigned U.S. patent application Ser. No. 10/973,076 (U.S. Published Patent Application 20060107107) 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. Embodiments are provided herein for establishing a spare processor for the boot processor. That is, embodiments described herein provide techniques for establishing, in a multi-processor system, a spare processor that assumes the role of boot processor upon LOL being detected for the system's boot processor.

In certain embodiments, information (referred to herein as a "device tree") is stored to the 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).

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.

Concurrently filed and commonly assigned U.S. patent application Ser. No. 10/973,076 (U.S. Published Patent Application 20060107107) 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 recognire the lockstep pair of processors as again being available for servicing instructions.

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 establishing/defining such a spare processor 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 Ser. No. 10/973,076 (U.S. Published Patent Application 20060107107) titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," for example.

Various techniques are provided herein for designating the spare for a boot processor. In certain embodiments, a user interface is provided for receiving input specifying the boot processor and/or its spare. In certain embodiments, the system's firmware selects the boot processor and/or its spare. In accordance with one embodiment provided herein, the multiprocessor system includes multiple buses, wherein certain processors are communicatively coupled to a first bus and certain other processors are communicatively coupled to a different bus. In such an architecture, the spare processor is designated as a processor that is coupled to the same bus as the boot processor. Thus, if a user is allowed to select the boot processor and the spare, once the boot processor is selected, the user interface may present the other processor(s) that are coupled to the same bus as the selected boot processor for selection of a spare therefrom. Alternatively, once the boot processor is selected by a user, the system firmware may select a spare as a processor that is coupled to the same bus as the boot processor. In certain embodiments, the system firmware may select the boot processor, as well as a spare for the boot processor.

Turning to FIG. **1**, an example embodiment of a system **10** that establishes a spare processor for use 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 **12**A and slave processor **12**B (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 **12**A is a first core and slave processor **12**B 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 (**12**A and **12**B) 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 **12**B is merely used for redundant processing for checking the output of master processor **12**A. 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.

Master processor **12**A includes cache **14**A, and slave processor **12**B includes cache **14**B. OS **11** and lockstep processor pair **12** are communicatively coupled to bus **16**. Typically, master processor **12**A and slave processor **12**B 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 **12**A back onto bus **16**. The output of slave processor **12**B is used solely for checking the output of mater processor **12**A.

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

Lockstep mismatch is one way of detecting a LOL between the master processor **12**A and slave processor **12**B. A detection of an error by either of error detect logic **13**A and **13**B also provides detection of LOL in the processors **12**A and **12**B. Because the detection of LOL by error detect logic **13**A and **13**B may occur before an actual lockstep mismatch occurs, the detection of LOL by error detect logic **13**A and **13**B 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 **13**A or **13**B, such error may eventually propagate to a lockstep mismatch error that is detectable by error detect logic **13**C.

System **10** further includes a second lockstep processor pair **15** that is communicatively coupled to bus **16**. Lockstep processor pair **15** includes master processor **15**A and slave processor **15**B. Master processor **15**A includes cache **17**A, and slave processor **15**B includes cache **17**B. As with error detect logic **13**A-**13**C implemented for lockstep processor pair **12**, lockstep processor pair **15** has error detect logic **16**A-**16**C. While only two lockstep processor pairs **12** and **15** are shown for simplicity in the example of FIG. **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 FIG. **1**, lockstep processor pair **15** is designated as a spare for boot processor **12**, as described further herein.

In certain embodiments described herein, techniques for designating a spare processor for a boot processor are provided, wherein 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. Thus, in certain embodiments, a "hot" spare is designated for the boot processor.

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. Then, the former boot processor for which the LOL was detected is 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.

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 **13**A-**13**C and **16**A-**16**C. In certain embodiments, processors **12**A, **12**B, **15**A, and **15**B 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.).

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 FIG. **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.

In the example system **10** of FIG. **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.

Suppose that during the system boot-up procedure, master processor **12**A assumes the role of boot processor, as shown in the example of FIG. **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 **12**A. Of course, additional lockstep processor pairs may be included in system **10** (not specifically shown in the example of FIG. **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 **12**A in this example).

Firmware **18** is included in this example, and upon detection of LOL by any of error detect logics **13**A-**13**C, 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 FIG. **1** firmware **18** crashes the system in block **102**.

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 **12**A is the system's boot processor. This can be determined, for example, by accessing the device tree **203** of FIG. **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 **12**A 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**.

In the specific example shown in FIG. **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 FIG. **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 Ser. No. 10/973,076 (U.S. Published Patent Application 20060107107) titled "SYSTEM AND METHOD FOR PROVIDING FIRMWARE RECOVERABLE LOCKSTEP PROTECTION," the disclosure of which is incorporated herein by reference.

In the example embodiment of FIG. **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 **12**A in operational block **104**. In this embodiment, firmware **18** utilizes an ACPI method **105** to "eject" master processor **12**A, thereby triggering OS **11** to idle the master processor **12**A (i.e., stop scheduling tasks for the processor). Of course, by idling master processor **12**A, slave processor **12**B will in turn be idled. Thus, idling master processor **12**A results in idling the lockstep processor pair **12**. In this example embodiment, OS **11** is not aware of the presence of slave processor **12**B, but is instead aware of master processor **12**A. The interface of lockstep processor pair **12** to bus **16** manages copying to slave processor **12**B the instructions that are directed by OS **11** to master processor **12**A. Thus, firmware **18** need not direct OS **11** to eject slave processor **12**B, as OS **11** is not aware of such slave processor **12**B in this example implementation. Again, by idling master processor **12**A, slave processor **12**B is also idled as it merely receives copies of the instructions directed to master processor **12**A. Of course, if in a given implementation OS **11** is aware of slave **12**B, firmware **18** may be implemented to also direct OS **11** to idle such slave processor **12**B in a manner similar to that described for idling master processor **12**A.

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 **12**A to OS **11** in operational block **107**. In this embodiment, firmware **18** updates the ACPI device table information for master processor **12**A to indicate that such master processor **12**A 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 **12**A, thereby reintroducing the master processor **12**A to OS **11**. As a result of checking for master processor **12**A, OS **11** will recognize that such master processor **12**A is again available and will thus begin scheduling tasks for master processor **12**A once again.

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 FIG. **1**, if determined in block **103** that the master processor **12**A 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 **15**A and **15**B in the spare lockstep processor pair **15**. Example techniques that may be utilized by firmware **18** for switching the state of a "good" processor in the lockstep processor pair 12 in accordance with certain embodiments is described further in co-pending U.S. patent application Ser. No. 10/187,833 (published as publication no. US 2004/0006722 A1) filed Jul. 3, 2002 (now U.S. Pat. No. 7,085,959) and titled "METHOD AND APPARATUS FOR RECOVERY FROM LOSS OF LOCK STEP" and in concurrently filed and commonly assigned U.S. patent application Ser. No. 10/972,588 (U.S. Published Patent Application 20060090064) 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", the disclosures of which are each hereby incorporated herein by reference.

Thus, operational block **109** essentially makes the spare processor pair **15** the system's boot processor, and then in block **110** firmware **18** resets the lockstep processor pair **12** and reestablishes its lockstep. 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, tirrnware **18** updates the device tree **203** of FIG. **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 Ser. No. 10/972,796 U.S. Published Patent Application 20060107106) titled "SYSTEM AND METHOD FOR MAINTAINING IN A MULTIPROCESSOR 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.

Embodiments described herein enable a spare processor to be designated for a boot processor, such as spare processor **15** in the above example. In certain implementations, a system administrator can designate through a user interface the boot processor, and, in some cases, further designate the corresponding spare for such boot processor. In other implementations, a system administrator may designate the boot processor and firmware selects the appropriate spare processor. In other embodiments, the firmware may select a boot processor and the corresponding spare processor for such boot processor. The information identifying the boot processor and the spare processor to be used for recovery from LOL detected for the boot processor is stored to the system in a data storage device (e.g., SRAM) that is accessible by the system's firmware to enable the firmware to identify those processors.

FIG. **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 $\mathbf{10}_A$. 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 $\mathbf{10}_A$, firmware **18**, labeled as firmware $\mathbf{18}_A$, 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.

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 **12**A). The boot processor begins executing code out of the system's Read-Only Memory (ROM) (not specifically shown in FIG. **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 $\mathbf{10}_A$ (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.

After it has performed its duty of initializing the processor(s), PAL **201** passes control of system $\mathbf{10}_A$ 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 $\mathbf{18}_A$ 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 $\mathbf{10}_A$ by branching the boot processor into the entry point of such OS loader program.

The OS loader program then uses the standard firmware interfaces to discover and initialize system $\mathbf{10}_A$ 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 $\mathbf{18}_A$ is no longer in control of system $\mathbf{10}_A$.

As OS **11** is initializing, it has to discover from the firmware $\mathbf{18}_A$ 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 **12**A via the supporting firmware $\mathbf{18}_A$, such as querying PAL **201** for the number, size, etc., of the processor's cache **14**A. 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).

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 **18**A 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 **15**A 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.

If a lost lockstep is recovered in accordance with the recovery technique described herein, firmware **18**A 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.

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:

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 |
| ... | ... | ... | ... | ... |

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."

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.

In the example of FIG. **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. User interface **20** may be implemented as a command line interface or as a graphical user interface (GUI), as examples. As one example, the user interface **20** may be implemented as an EFI application that provides a command line interface for inputting commands. Such command line interface may support commands for assigning the boot processor and/or spare processor, as well as performing such tasks as configuring and deconfiguring processors, deconfiguring bad memory, specifying which OS to boot, etc. This command line interface implements a typical command <argument> format and has hooks into the firmware to change settings in NVRAM. As another example, the user interface **20** may be implemented within "parmgr", which is an HP-UX application that controls how partitions are configured, memory interleave, CPU configuration, etc. Such parmgr provides a GUI interface and a command line interface. The GUI interface presents a "worksheet" that has the current values filled in and allows the system administrator to update/change those values and then "apply" the changes. At that time, parmgr pushes the changes into NVRAM so they will be honored on the next reset. Various other implementations of a user interface **20** may be employed in alternative embodiments.

One example system in which firmware $\mathbf{18}_A$ of FIG. **2** may be employed is shown in FIG. **3**. FIG. **3** shows system **30** that includes multi-processor cells A-D, labeled $\mathbf{31}_A$-$\mathbf{31}_D$, respectively. Cell A $\mathbf{31}_A$ is shown in more detail, and should be understood that cells B-D $\mathbf{31}_B$-$\mathbf{31}_D$ have substantially the same architecture as that of cell A $\mathbf{31}_A$ in this example. The architecture of cell A $\mathbf{31}_A$ described hereafter corresponds to that of Hewlett-Packard's Superdome™ systems. As shown, multi-processor cell A $\mathbf{31}_A$ 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 $\mathbf{31}_A$, 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 FIGS. **1**-**3**. Thus, for instance, CPU **0** (**300**) includes both a master processor and a slave processor, as with processors **12**A and **12**B included in the lockstep processor pair **12** of FIGS. **1**-**2**.

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.

As shown, the example firmware $\mathbf{18}_A$ described above with FIGS. **1**-**2** may be employed to manage the recovery from detected LOLs for the processors of cells A-D ($\mathbf{31}_A$-$\mathbf{31}_D$). 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 $\mathbf{18}_A$ 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 $\mathbf{18}_A$ may operate according to the process described above in FIG. **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**).

It should be recognized that in the example system of FIG. **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 FIG. **1** (i.e., operational blocks **104**-**108**) can be utilized without requiring a spare for lockstep recovery for those non-boot processors.

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 FIG. **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.

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.

Turning to FIG. **4**, an operational flow diagram for establishing a hot spare for the boot processor according to one embodiment is shown. The operational loop of FIG. **4** is performed for each processor in a multi-processor system. In block **401**, the processor boots in lockstep mode. As described above, in certain embodiments, whether the processors are to operate in lockstep mode is configurable in the device tree, and thus the firmware may consult the device tree to determine whether to boot each processor in lockstep mode. In operational block **402**, the firmware determines whether the processor is configured to be a hot spare for the system's boot processor. As described above, this determination can be made by accessing the device tree maintained in the firmware. If the processor is not configured to be a hot spare, then operation advances to block **403** whereat the normal boot process continues for this processor. If, on the other hand, the processor is configured to be a hot spare, then operation advances to block **404** whereat hot spare specific properties are set in the device tree. If configured to be a hot spare, the firmware will hide the processor from the system's OS by setting properties on the node associated with this CPU in the firmware device tree. Typically, nodes of type "CPU" are reported to the OS in the ACPI tables. In this case, the "type" will be set in the device tree to "hot_spare" for the designated spare processor so that this processor is not reported in the ACPI namespace. Thus, when the system firmware builds its ACPI tables, it looks for devices of type "CPU", and the hot spare is not placed in the ACPI tables because its type is set to "hot_spare" in this example. Accordingly, the OS will not be aware of the hot spare processor.

In operational block **405**, the hot spare processor executes in an idle loop waiting for a PAL call to set state. Such PAL call to "set state" copies the saved state from the processor that experienced LOL (i.e., the state of the "good" one of the lockstep pair of processors in the processor module that experienced LOL) into the hot spare processor. Specifically, 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. This call does not change the device tree settings. Accordingly, with the spare processor hidden from the OS (i.e., the OS is unaware of the hot spare), the hot spare sits in an idle loop until it is needed for recovering from LOL detected for the boot processor, at which time the system firmware interrupts its loop and assigns it the role of boot processor, as described above.

FIG. **5** shows an operational flow diagram for establishing an active spare for the boot processor and assigning such active spare the role of boot processor responsive to detecting LOL for the boot processor in accordance with one embodiment. Operational blocks **501-504** of FIG. **5** are performed for each processor in a multi-processor system for establishing an active spare in the system. In block **501**, the processor boots in lockstep mode. As described above, in certain embodiments, whether the processors are to operate in lockstep mode is configurable in the device tree, and thus the firmware may consult the device tree to determine whether to boot each processor in lockstep mode. In operational block **502**, the firmware determines whether the processor is configured to be an active spare for the system's boot processor. As described above, this determination can be made by accessing the device tree maintained in the firmware. If the processor is not configured to be an active spare, then operation advances to block **503** whereat the normal boot process continues for this processor. If, on the other hand, the processor is configured to be an active spare, then operation advances to block **504** whereat active spare specific properties are set in the device tree. For instance, in this example, the device tree would include a property called "Active Spare" and this property would be set to TRUE for the active spare processor. If LOL is detected on the boot processor, firmware searches the devices in the device tree for the "Active Spare=TRUE" condition to find the designated active spare processor. Thereafter, operation returns to operational block **503**, whereat the processor continues its normal boot and is introduced to the OS, rather than being hidden from the OS as the above-described hot spare.

Operational block **505** represents the system's normal operation until such time as a LOL is detected for the boot processor. Upon error detect logic detecting a LOL for a processor, the error detect logic invokes the system's firmware. For instance, the LOL may invoke the PAL, which in turn may report the LOL to the SAL. The firmware (e.g., SAL) determines if the processor for which the LOL is detected is designated (e.g., in the device tree) as the boot processor. If so, then the firmware determines in block **505** that LOL is detected for the boot processor. Thus, in operational block **506**, the firmware determines (e.g., from the device tree) the processor designated as the active spare, and the firmware instructs the OS to "eject" the active spare processor. Thus, the active spare is idled and effectively transformed into a hot spare. In operational block **507**, the firmware assigns the active spare processor the role of boot processor. The state of the "good" one of the lockstep processor pair of the boot processor for which the LOL is detected is copied into both of the lockstep processors of the active spare processor. Further, the firmware updates the device tree to identify the active spare as the boot processor, and performs any other configuration changes (e.g., changing the LID of the active spare, etc.) to cause the OS to recognize the active spare as the boot processor.

In block **508**, the firmware attempts to recover lockstep for the former boot processor for which LOL was detected in block **505**. In operational block **509**, the firmware determines whether the lockstep recovery for the former boot processor is successful. In certain embodiments, if the lockstep cannot be recovered in operational block **508** (e.g., the processor fails its self-test when being reset), then that processor is simply not reintroduced back to the OS and in operational block **510** a message is generated by the system to provide notice of the loss of this processor and that the boot processor no longer has a spare. If the lockstep cannot be reestablished in block **508**, then if a LOL occurs in the new boot processor there will not be a spare processor for taking over its role (and thus the system may crash). If desired, the system administrator can schedule an orderly shutdown and re-boot in order to establish a new boot processor that has a hot spare available and/or schedule service of the system (e.g., for replacing the failed processor).

If the lockstep recovery is successful, then the former boot processor is assigned the role of active spare (e.g., by designating it as such in the device tree) in block **511**, and the firmware triggers the OS to recognize the active spare as available for executing instructions. Operation then returns to block **505**, and if a LOL is detected for the current boot processor, the procedure of blocks **506-512** may be repeated.

Turning to FIG. **6**, a general flow diagram for operation of certain embodiments is shown. In block **601**, the role of boot processor is assigned to one of a plurality of processors in a system. As described above, the role of boot processor may be so assigned in a device tree. Such role of boot processor may be assigned by a user (e.g., via a user interface) in certain embodiments, and assigned by the system's firmware in other embodiments.

In block **602**, the role of spare processor is assigned to another one of the processors. As described above, in certain embodiments the spare processor may be a hot spare, while in other embodiments the spare processor may be an active spare. As also described above, the role of spare processor may be so assigned in a device tree. Such role of spare processor may be assigned by a user (e.g., via a user interface) in certain embodiments, and assigned by the system's firmware in other embodiments.

In operational block **603**, the system firmware, responsive to detecting LOL for the boot processor, changes the assignment of the role of boot processor to the spare processor. This change in the role of boot processor occurs dynamically during system runtime, rather than requiring that the system be interrupted (e.g., shutdown and restarted, or crashed).

Certain systems include a plurality of partitions, wherein each partition includes a boot processor and OS. It should be understood that the above techniques for establishing a spare processor for use in assuming the role of boot processor in the event that a LOL is detected for the boot processor may be employed on each partition. Thus, a device tree may designate a boot processor and spare (e.g., active or hot) for each partition.

What is claimed is:

**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 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 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.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

# CERTIFICATE OF CORRECTION



PATENT NO. : 7,627,781 B2
APPLICATION NO. : 10/973077
DATED : December 1, 2009
INVENTOR(S) : Scott L. Michaelis et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

On the Title page, in Item (56), U.S. Patent Documents, in column 2, line 13, below "6,604,177 B1 8/2003 Kondo et al." insert -- 6,615,366 9/2003 Grochowski et al. --.

In the Claims:
In column 19, line 43, in Claim 12, before "processors" insert -- of --.

Signed and Sealed this

Nineteenth Day of October, 2010

David J. Kappos

David J. Kappos
*Director of the United States Patent and Trademark Office*