UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

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

v.

FORAS TECHNOLOGIES LTD.,
Patent Owner

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

Inter Partes Review No.: IPR2024-00969

DECLARATION OF R. JACOB BAKER, PH.D., P.E.

TABLE OF CONTENTS

Page

I, R. Jacob Baker, declare that I have personal knowledge of the facts set forth in this declaration and, if called to testify as a witness, could and would do so competently.

I. INTRODUCTION

1. My name is R. Jacob Baker Ph.D., P.E., and I am a Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas. I have been retained as an expert witness and have prepared this declaration on behalf of Petitioners ZF Friedrichshafen AG, ZF Active Safety and Electronics US LLC, and Nissan Motor Company, Ltd., for the above-referenced *inter partes* review proceeding.

2. In this declaration, I will give my opinion as to whether claims 9-12 and 23-25 of U.S. Patent No. 7,502,958 ("the '958 Patent") (Ex. 1001) are valid. I also provide herein the technical bases for these opinions, as appropriate. I understand that the '958 Patent is owned by Foras Technologies Ltd.

3. This declaration contains statements of my opinions formed to date, and the bases and rationale for these opinions. I may offer additional opinions based on further review of materials in this case, including opinions and/or testimony of other expert witnesses.

4. For my efforts in connection with the preparation of this declaration, I have been compensated at my usual and customary rate for this type of consulting

activity. My compensation is in no way contingent on the results of these or any other proceedings related to the '958 Patent.

II. PROFESSIONAL BACKGROUND AND QUALIFICATIONS

5. My qualifications generally are set forth in my *curriculum vitae*, which is submitted as Ex. 1003 to this declaration.

6. I have been working as an Engineer since 1985 and I have been teaching Electrical and Computer Engineering courses since 1991. I am currently a Professor of Electrical and Computer Engineering at the University of Nevada, Las Vegas ("UNLV"). I am also currently an industry consultant for Freedom Photonics.

7. I received B.S. and M.S. degrees in Electrical Engineering from UNLV in 1986 and 1988, respectively. I received my Ph.D. in Electrical Engineering from the University of Nevada, Reno, in 1993.

8. My doctoral research, culminating in the award of a Ph.D., investigated the use of power MOSFETs in the design of very high peak power, and high-speed, instrumentation.

9. I am the named inventor on over 150 U.S. patents resulting from my industry work. Many of these patents relate to processor technology including redundant system design and error correction.

10. I have done technical and expert witness consulting for over 200 companies since I started working as an engineer in 1985. From 1985 to 1993 I

worked for EG&G Energy Measurements and the Lawrence Livermore National Laboratory designing nuclear diagnostic instrumentation for underground nuclear weapon tests at the Nevada test site. During this time, I designed, and oversaw the fabrication of, over 30 electronic and electro-optic instruments including high-speed cable and fiber-optic receiver/transmitters, PLLs, frame and bit-syncs, data converters, streak-camera sweep circuits, Pockel's cell drivers, micro-channel plate gating circuits, charging circuits for battery backup of equipment for recording test data, and analog oscilloscope electronics. Many of these electronic instruments included redundant system design features including redundant image processing. This was required since the cost of the nuclear tests were significant and the Physicists performing the experiments did not want lose data for any reason.

11. From 1994 to 1996 I worked at Micron Display and Micron Technology, Inc. ("Micron") in Boise, Idaho. The image processing work I performed at Micron often involved redundant designs to ensure manufacturability and improve product yield.

12. I am a licensed Professional Engineer and have extensive industry experience in circuit design, fabrication, and manufacture of Dynamic Random Access Memory (DRAM) semiconductor integrated circuit chips, Phase-Change Random Access Memory (PCRAM) chips, and CMOS Image Sensors (CISs) at Micron Technology, Inc. ("Micron") in Boise, Idaho. I spent considerable time

working on the development of Flash memory chips while at Micron. My efforts resulted in more than a dozen patents relating to Flash memory. One of my projects at Micron included the development, design, and testing of circuit design techniques for a multi-level cell (MLC) Flash memory using signal processing. Another project focused on the design of buffers for high-speed double-data rate DRAM which resulted in around 10 US patents in buffer design. Among many other experiences, I led the development of the delay locked loop (DLL) in the late 1990s so that Micron DRAM products could transition to the DDR memory protocol, used in mobile and non-mobile (server, desktop, cell phones, tablets, etc.) computing systems as main computer memory, for addressing and controlling accesses to memory via interprocess communications (IPC) with the memory controller (MC). I provided technical assistance with Micron's acquisition of Photobit during 2001 and 2002, including transitioning the manufacture of CIS products into Micron's process technology.

13. I have extensive experience in the development of instrumentation and commercial products in a multitude of areas including: integrated electrical/biological circuits and systems, array (memory, imagers, and displays) circuit design, CMOS analog and digital circuit design, diagnostic electrical and electro-optic instrumentation for scientific research, CAD tool development and online tutorials, low-power interconnect and packaging techniques, design of

communication/interface circuits (to meet commercial standards such as USB, firewire, DDR, PCIe, SPI, etc.), circuit design for the use and storage of renewable energy, and power electronics.

14. I was an adjunct faculty member in the Electrical Engineering department of the University of Nevada, Las Vegas in 1991 and 1992. From 1993 to 2000, I served on the faculty at the University of Idaho as an Assistant Professor and then as a tenured Associate Professor of Electrical Engineering. In 2000, I joined a new Electrical and Computer Engineering program at Boise State University ("BSU") where I served as department chair from 2004 to 2007. At BSU, I helped establish graduate programs in Electrical and Computer Engineering including, in 2006, the university's second Ph.D. degree. In 2012, I re-joined the faculty at UNLV. Over the course of my career as a professor I have advised over 100 masters and doctoral students.

15. On numerous occasions, I have been recognized for my contributions as an educator in the field. While at Boise State University, I received the President's Research and Scholarship Award (2005), Honored Faculty Member recognition (2003), and Outstanding Department of Electrical Engineering Faculty recognition (2001). In 2007, I received the Frederick Emmons Terman Award (the "Father of Silicon Valley"). The Terman Award is bestowed annually upon an outstanding young electrical/computer engineering educator in recognition of the educator's

contributions to the profession. In 2011, I received the IEEE Circuits and Systems Education Award. I received the Tau Beta Pi Outstanding Electrical and Computer Engineering Professor Award every year it was awarded while I have been back at UNLV.

16. I have authored several books and papers in the electrical and computer engineering area. My published books include *CMOS Circuit Design, Layout, and Simulation* (Baker, R.J., *Wiley-IEEE*, ISBN: 9781119481515 (4th ed., 2019)) and *CMOS Mixed-Signal Circuit Design* (Baker, R.J., *Wiley-IEEE*, ISBN: 9780470290262 (2nded., 2009) and ISBN: 978-0471227540 (1st ed., 2002)). I co-authored *DRAM Circuit Design: Fundamental and High-Speed Topics* (Keeth, B., Baker, R.J., Johnson, B., and Lin, F., *Wiley-IEEE*, ISBN: 9780470184752 (2008)), *DRAM Circuit Design: A Tutorial* (Keeth, B. and Baker, R.J., *Wiley-IEEE*, ISBN: 0780360141 (2001)), and *CMOS Circuit Design, Layout and Simulation* (Baker, R.J., Li, H.W., and Boyce, D.E., *Wiley-IEEE*, ISBN: 9780780334168 (1998)). I contributed as an editor and co-author on several other electrical and computer engineering books.

17. I have performed technical and expert witness consulting for over 125 companies and laboratories and given more than 50 invited talks at conferences, companies, and universities. Further, I am the author or co-author of more than 100

papers and presentations in the areas of electrical and computer engineering design, fabrication and packaging.

18. I currently serve, or have served, as a volunteer on: the IEEE Press Editorial Board (1999-2004); as editor for the Wiley-IEEE Press Book Series on Microelectronic Systems (2010-2018); as the Technical Program Chair of the 2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS 2015); on the IEEE Solid-State Circuits Society (SSCS) Administrative Committee (2011-2016); as a Distinguished Lecturer for the SSCS (2012-2015); as the Technology Editor (2012-2014) and Editor-in-Chief (2015-2020) for the *IEEE Solid-State Circuits Magazine*; IEEE Kirchhoff Award Committee (2020-present); and advisor for the student branch of the IEEE at UNLV (2013-present). These meetings, groups, and publications are intended to allow researchers to share and coordinate research. My active participation in these meetings, groups, and publications allowed me to see what other researchers in the field have been doing. In addition to the above, I am an IEEE Fellow and a member of the honor societies Eta Kappa Nu and Tau Beta Pi.

III. MATERIALS REVIEWED

19. I have considered information from various sources in forming my opinions. For example, a list of materials I have considered is included in a list of exhibits attached as Appendix A to this declaration. That list includes the ’958

Patent itself (Ex. 1001), the prosecution history of the '958 Patent (Ex. 1004), and numerous other publications. I may review additional documents filed in connection with this proceeding as they become available.

20. I have also reviewed the petition for *inter partes* review of the '958 Patent, the Declaration of Dr. Michael C. Brogioli, and the Decision – Granting Institution of *Inter Partes* Review, for IPR2023-1373. I have also reviewed the prosecution history of the re-examination of the '958 Patent (Appl. No. 90/019,245). I have analyzed the following sections of Dr. Brogioli's declaration with which I generally agree: overview of The State of Art, Overview of the '958 Patent, Overview of Bigbee (Ex. 1005), Overview of Nguyen (Ex. 1006), discussion of Bigbee and Nguyen as analogous art to the '958 Patent, Claim Construction analysis, and analysis of claim 1 (from ground A explaining that claim 1 is obvious over Bigbee in view of Nguyen). Sections of my declaration herein follow language contained in these sections from Dr. Brogioli's declaration, but all portions of this declaration are my own opinions and I am not relying on Dr. Brogioli's opinions as support for my opinions.

IV. UNDERSTANDING OF APPLICABLE LEGAL STANDARDS

21. I have applied the following legal principles provided to me by counsel in arriving at the opinions set forth in this declaration.

A. Legal Standard for Prior Art

22. I understand that a patent or other publication must first qualify as prior art before it can be used to invalidate a patent claim.

23. I understand that a U.S. or foreign patent or patent publication qualifies as prior art, under pre-AIA 35 U.S.C. § 102(a), to an asserted patent if the date of issuance or publication of the patent or patent publication is prior to the alleged invention of the asserted patent. I further understand that a printed publication, such as a book or an article published in a magazine or trade publication, qualifies as prior art, under pre-AIA 35 U.S.C. § 102(a), to an asserted patent if the date of publication (e.g., the date it was made publicly accessible) is prior to the alleged invention of the asserted patent.

24. I understand that a U.S. or foreign patent or patent publication qualifies as prior art, under pre-AIA 35 U.S.C. § 102(b), to an asserted patent if the date of issuance or publication of the patent or patent publication is more than one year before the filing date of the asserted patent. I further understand that a printed publication, such as a book or an article published in a magazine or trade publication, qualifies as prior art, under pre-AIA 35 U.S.C. § 102(b), to an asserted patent if the date of publication (e.g., the date it was made publicly accessible) is more than one year before the filing date of the asserted patent.

25. I understand that a U.S. patent or patent publication qualifies as prior art, under pre-AIA 35 U.S.C. § 102(e), to the asserted patent if the application

resulting in that patent or patent publication was filed in the United States before the alleged invention of the asserted patent.

26. I understand that documents and materials that qualify as prior art can be used to invalidate a patent claim as anticipated or as obvious.

B. Legal Standard for Anticipation

27. I understand that once the claims of a patent have been properly construed, the second step in determining anticipation of a patent claim requires a comparison of the properly construed claim language to the prior art on a limitation-by-limitation basis.

28. I understand that a single prior art reference "anticipates" an asserted claim, and thus renders the claim invalid, if each and every element of the claim is disclosed in that prior art reference, either explicitly or inherently (*i.e.*, necessarily present), and it enables a person having ordinary skill in the art (POSITA) to make and use the claimed invention without undue experimentation.

29. I understand that a patent is anticipated if before such person's invention thereof, the invention was made in this country by another inventor who had not abandoned, suppressed, or concealed it.

30. I have written this declaration with the understanding that in an *inter partes* review anticipation must be shown by a preponderance of the evidence.

C. Legal Standard for Obviousness

31. I have been instructed by counsel on the law regarding obviousness, and understand that even if a patent is not anticipated, it is still invalid if the differences between the claimed subject matter and the prior art are such that the subject matter as a whole would have been obvious to a POSITA at the time the invention was made.

32. I understand that a POSITA provides a reference point from which the prior art and claimed invention should be viewed. This reference point prevents a person from using one's own insight or hindsight in deciding whether a claim is obvious. I understand that hindsight is impermissible and that permissible reasoning takes into account only knowledge which was within the level of ordinary skill in the art at the time the claimed invention was made and does not include knowledge gleaned only from the '958 Patent's disclosure.

33. I also understand that an obviousness determination includes the consideration of various factors such as (1) the scope and content of the prior art, (2) the differences between the prior art and the challenged claims, (3) the level of ordinary skill in the pertinent art, and (4) the existence of objective evidence of non-obviousness (*i.e.*, "secondary considerations") such as commercial success, long-felt but unresolved needs, failure of others, etc.

34. I am informed that secondary considerations of non-obviousness may include (1) a long felt but unmet need in the prior art that was satisfied by the claimed

invention of the patent; (2) commercial success attributable to the claimed invention; (3) unexpected results achieved by the claimed invention; (4) praise of the claimed invention by others skilled in the art; (5) taking of licenses under the patent by others for reasons related to the alleged non-obviousness of the claimed invention; (6) deliberate copying of the claimed invention instead of using the prior art; (7) failure of others; (8) teaching away by others; and (9) skepticism by experts. I also understand that there must be a relationship, or nexus, between any such secondary indicia and the claimed invention. I further understand that contemporaneous and independent invention by others is a secondary consideration supporting an obviousness determination.

35. I understand that an obviousness evaluation can be based on a combination of multiple prior art references. I understand that rationales that may support a conclusion of obviousness include:

(A) Combining prior art elements according to known methods to yield predictable results;

(B) Substituting one known element for another to obtain predictable results;

(C) Using known techniques to improve similar devices (methods, or products) in the same way;

(D) Applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;

(E) Using an "obvious to try" solution, *i.e.*, choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success;

(F) Applying work known in one field of endeavor to yield variations of that work 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; and

(G) Applying some 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 prior art reference disclosures to arrive at the claimed invention.

36. Accordingly, I understand that the prior art references themselves may provide a suggestion, motivation, or reason to combine, but other times the link between two or more prior art references is simple common sense. I further understand that obviousness analysis recognizes that market demand, rather than scientific literature, often drives innovation, and that a motivation to combine references may be supplied by the direction of the marketplace.

37. I understand that if a technique has been used to improve one device, and a POSITA would have recognized 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.

38. I also understand that practical and common-sense considerations should guide a proper obviousness analysis, because familiar items may have obvious uses beyond their primary purposes. I further understand that a POSITA looking to overcome a problem will often be able to fit the teachings of multiple publications together like pieces of a puzzle, although the prior art need not be like two puzzle pieces that must fit perfectly together. I understand that obviousness analysis therefore considers the inferences and creative steps that a POSITA would employ under the circumstances.

39. I understand that a particular combination may be proven obvious by showing that it was obvious to try the combination. For example, when there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions with a reasonable expectation of success, a POSITA has good reason to pursue the known options within his or her technical grasp because the result is likely the product not of innovation but of ordinary skill and common sense.

40. I understand that the combination of familiar elements according to known methods may be proven obvious when it does no more than yield predictable results. When a work is available in one field of endeavor, design incentives and other market forces can prompt variations of it, either in the same field or a different

one. If a POSITA can implement a predictable variation, that variation likely bars its patentability.

41. It is further my understanding that a proper obviousness analysis focuses on what was known or obvious to a POSITA, not just the patentee. Accordingly, I understand that any need or problem known in the field of endeavor at the time of the claimed invention and addressed by the patent can provide a reason for combining the elements in the manner claimed.

42. I understand that a claim can be obvious in light of a single reference, without the need to combine references, if the elements of the claim that are not found explicitly or inherently in the reference can be supplied by the common sense or technical knowledge of the POSITA. For example, combining two embodiments disclosed adjacent to each other in a prior art reference probably does not require a leap of inventiveness.

43. I understand that a POSITA could have combined two pieces of prior art or substituted one prior art element for another if the substitution can be made with predictable results, even if the swapped-in element is different from the swapped-out element. In other words, the prior art need not be like two puzzle pieces that must fit together perfectly. The relevant question is whether prior art techniques are interoperable with respect to one another, such that a POSITA would view them

as a design choice, or whether a POSITA would apply prior art techniques into a new combined system.

44. In sum, my understanding is that prior art teachings are properly combined where a POSITA having the understanding and knowledge reflected in the prior art and motivated by the general problem facing the inventor, would have been led to make the combination of elements recited in the claims. Under this analysis, the prior art references themselves, or any need or problem known in the field of endeavor at the time of the invention, can provide a reason for combining the elements of multiple prior art references in the claimed manner.

45. I have been informed and understand that the obviousness analysis requires a comparison of the properly construed claim language to the prior art on a limitation-by-limitation basis, while observing whether the differences between the claimed subject matter and the prior art are such that the subject matter *as a whole* would have been obvious to a POSITA at the time the invention was made.

46. I have written this declaration with the understanding that in an *inter partes* review obviousness must be shown by a preponderance of the evidence.

D. Legal Standard for Claim Construction

47. I have been instructed by counsel on the law regarding claim construction and patent claims and understand that a patent may include two types of claims, independent claims and dependent claims. An independent claim stands

alone and includes only the limitations it recites. A dependent claim can depend from an independent claim or another dependent claim. I understand that a dependent claim includes all the limitations that it recites in addition to all of the limitations recited in the claim from which it depends.

48. It is my understanding that in proceedings before the USPTO, claims are construed similarly as in district court litigation, and that this standard is sometimes referred to as the *Phillips* standard. Under this standard, it is my understanding that claim terms are given their ordinary and customary meaning as would be understood by a POSITA at the time of the claimed invention, in view of the specification and prosecution history.

49. In comparing the claims of the ’958 Patent to the prior art, I have carefully considered the ’958 Patent and its prosecution history in light of the understanding of a POSITA at the time of the alleged invention.

50. I understand that to determine how a POSITA would understand a claim term, one should look to those sources available that show what a POSITA would have understood the claim term to mean. Such sources include the words of the claims themselves, the remainder of the patent’s specification, the prosecution history of the patent (all considered “intrinsic” evidence), and “extrinsic” evidence concerning relevant scientific principles, the meaning of technical terms, and the state of the art.

51. I understand that, in construing a claim term, one looks primarily to the intrinsic patent evidence, including the words of the claims themselves, the remainder of the patent specification, and the prosecution history.

52. I understand that extrinsic evidence, which is evidence external to the patent and the prosecution history, may also be useful in interpreting patent claims when the intrinsic evidence itself is insufficient.

53. I understand that words or terms should be given their ordinary and accepted meaning unless it appears that the inventors were using them to mean something else. In making this determination, the claims, the patent specification, and the prosecution history are of paramount importance. Additionally, the specification and prosecution history must be consulted to confirm whether the patentee has acted as its own lexicographer (*i.e.*, provided its own special meaning to any disputed terms), or intentionally disclaimed, disavowed, or surrendered any claim scope.

54. I understand that the claims of a patent define the scope of the rights conferred by the patent. The claims must particularly point out and distinctly claim the subject matter which the patentee regards as his invention. Because the patentee is required to define precisely what he claims his invention to be, it is improper to construe claims in a manner different from the plain import of the terms used consistent with the specification. Accordingly, a claim construction analysis must

begin and remain centered on the claim language itself. Additionally, the context in which a term is used in the asserted claim can be highly instructive. Likewise, other claims of the patent in question, both asserted and unasserted, can inform the meaning of a claim term. For example, because claim terms are normally used consistently throughout the patent, the usage of a term in one claim can often illuminate the meaning of the same term in other claims. Differences among claims can also be a useful guide in understanding the meaning of particular claim terms.

55. I understand that in general, a term or phrase found in the introductory words of the claim, the preamble of the claim, should be construed as a limitation if it recites essential structure or steps, or is necessary to give life, meaning, and vitality to the claim. Conversely, a preamble term or phrase is not limiting where a patentee defines a structurally complete invention in the claim body and uses the preamble only to state a purpose or intended use for the invention.

56. I understand that pre-AIA 35 U.S.C. § 112 ¶ 6 allows for means plus function limitations. In particular, I understand that a "means plus function" limitation should be interpreted to cover only the corresponding structure described in the specification, and equivalents thereof. I also understand that a limitation will be presumed to be a means plus function limitation if (a) the claim limitation uses the phrase "means for"; (b) the "means for" is modified by functional language; and

(c) the "means for" is not modified by sufficient structure for achieving the specified function.

57. I understand that a structure will be considered structurally equivalent to the corresponding structure identified in the specification only if the differences between them are insubstantial. For example, a structure may be deemed to be a structural equivalent to the corresponding structure where the structure performs the same function in substantially the same way to achieve substantially the same result.

V. TECHNICAL BACKGROUND

A. Lockstep Processors or FRC Processors Were Known

58. Prior to the filing date of the '958 Patent, lockstep processing was used to improve the fault tolerance of a computer system. For example, shown below are two processors 22 and 24 (or two processor units or two processor cores) that are paired together to execute the same instructions on identical data in parallel. The outputs of the two processors are compared by lockstep logic 26, which may include a comparator or an XOR gate. Safford, [0003]-[0004]; *see also* Bigbee, [0033]; Nguyen, [0006], [0038]; Kondo, [0004]; Ex. 1001, 2:13-19. A discrepancy between the outputs of the pair of processors detected by lockstep logic 26 is considered to be a loss of lockstep. Safford, [0004].

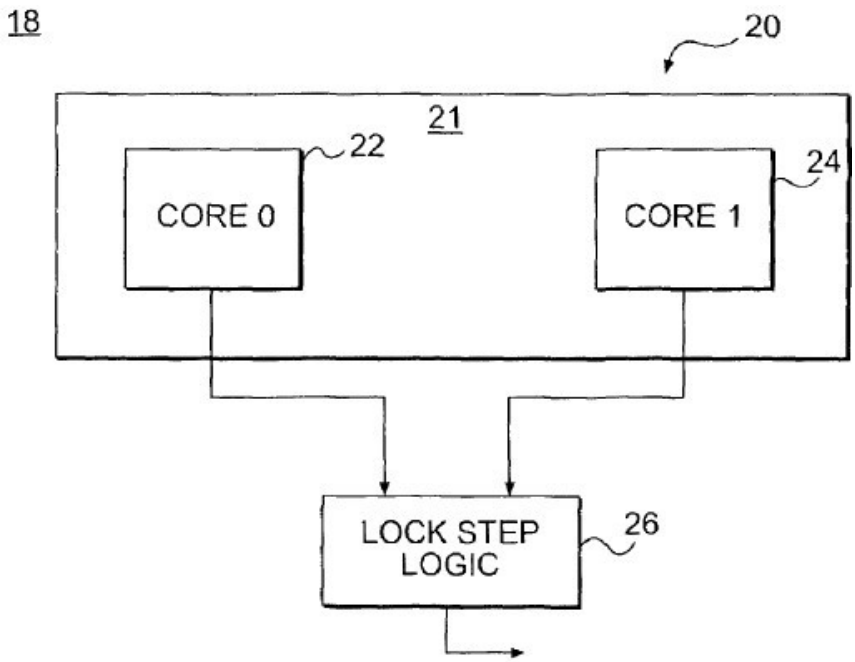


FIG. 1B
PRIOR ART

Safford, FIG. 1B.

59. Because lockstep processors execute redundant instructions in parallel, lockstep systems are said to perform a "functional redundancy check." Safford-181, 1:67-2:3. Accordingly, processors operating in lockstep are also referred to as processors operating in functional redundancy check (FRC) mode or FRC mode. A POSITA would have understood that the terms FRC processor or lockstep processor both refer to a processor including two or more processor cores operating in FRC mode or in lockstep.

60. When a comparison of the outputs of the pair of processors indicates that the pair of processors have produced the same outputs, it is assumed that there is no error in processing. In contrast, when a comparison of the outputs of the pair

of processors indicates a difference or mismatch, it is assumed that at least one processor has an error. Safford, [0016]. An example of such an error is a data cache error or a soft error.[1] *Id.*, [0003]; Nguyen, [0008]. A loss of lockstep, if not promptly corrected, may cause the computer system to crash. That is, a failure of one of the pair of processors may halt the entire computer system, even if the other processor does not encounter an error. Safford, [0003].

61. It was well known to a POSITA that the processors in a pair of lockstep processors are often differentiated by designating one as a master processor and the other as the slave processor, and that these roles may change over time. It also was well known to a POSITA that processors have status registers for recording such designations. Marshall, 1:11-16; Kondo, [0086]; Tu, [0031].

62. It was also known to a POSITA that an error detected in an individual processor of a pair of lockstep processors may signal an impending mismatch error.

---

[1] A POSITA would have understood that a "soft error" occurs when, for example, a high-energy particle switches the state of a memory element causing a parity or other error. Nguyen, [0004]; Kondo, [0006]. Soft errors are generally correctable by rewriting the affected memory location. Soft errors are distinct from "hard" errors which indicate the actual failure of a hardware element.

Safford, [0015]-[0017] (an error detected in a single processor core of a lockstep processor signals an *impending* loss of lockstep); Kondo, [0101] (detecting an error in the processor provides *early warning* of a divergence), [0103] ("When utilizing lockstepped processors, a recoverable error on one processor … *will* nevertheless result in a divergence, since it will take additional processing and some number of clock cycles on the part of the one processor to recover from the error."), [0106] (divergence between the processor cores of an FRC processor can be *predictively detected* by detecting errors in the processor cores); Nguyen, [0008] (soft errors in "FRC-enabled processors … are likely to be manifested as FRC errors, since they take the execution cores out of lock-step"), [0022] (data corruption in a processor core of an FRC processor can *trigger* an FRC error) (emphasis added).

B. Determining Whether Lockstep Is Recoverable Was Known

63. It was known to a POSITA that lockstep is recoverable from some errors in a lockstep processor, such as soft errors. It was also known that lockstep is not recoverable from some other errors, such as some lockstep mismatch errors.

64. For example, lockstep may be recoverable from an error that is detected in an individual processor before a mismatch error is detected and signaled. As shown below, for example, a pair of lockstep processors 111 and 113 can include error detection and signaling logic 112 and 114, respectively, to signal an impending loss of lockstep. Safford, [0017].

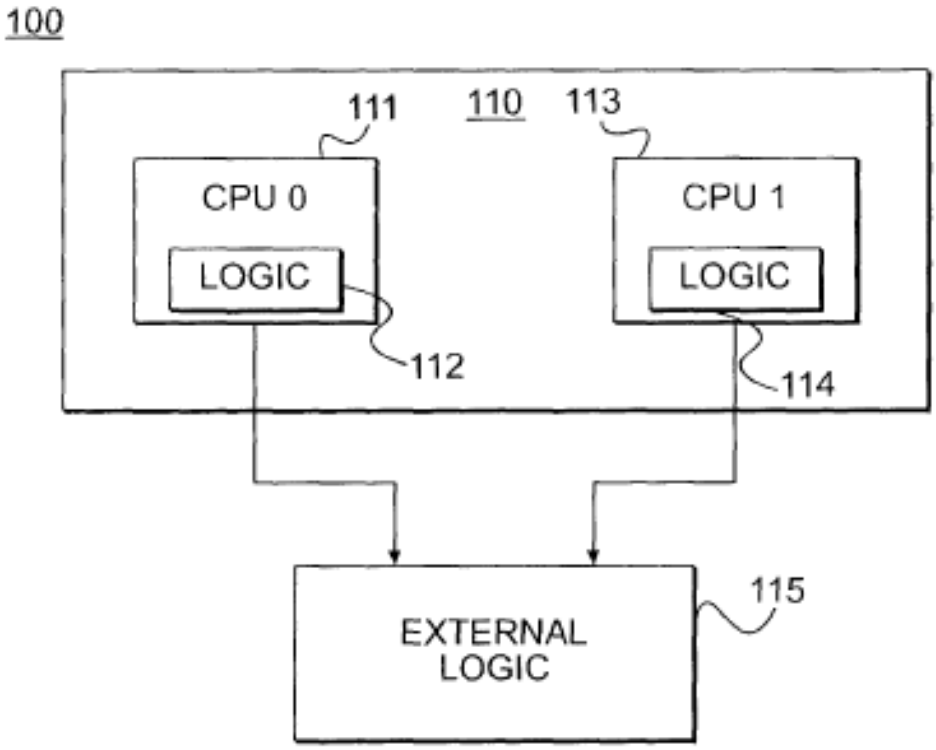


FIG. 2

Safford, FIG. 2.

65. It was known to a POSITA that detecting and handling such errors before signaling a mismatch error improves processor and/or system availability. Safford, [0017] (using the error detected in the processors of a lockstep processor pair to signal an impending loss of lockstep and improve availability of the processor assets); Nguyen, [0046] (detecting an error in the processor core before triggering an FRC reset allows recovery from the error); Kondo, [0100] (a processor error that will produce a divergence but will not compromise data integrity is a correctable memory error).

66. It was also known to a POSITA that lockstep may be non-recoverable from a mismatch error in various situations. For example, lockstep is not recoverable from a hard error that caused the mismatch (e.g., a partial or total hardware failure). Bigbee, [0033] (such inconsistencies can occur as a result of a partial or total hardware failure); Bossen, 1:35-40 (no recovery is possible for hard errors). Also, lockstep is not recoverable if it is not possible to determine which processor core was responsible for the mismatch error. Nguyen, [0023] ("Not all FRC-errors are traceable to underlying parity/ECC or other correctible soft errors.") [2]; Safford-181, 6:53-56 ("It can be difficult to recover from a lockstep error because typically it is not possible to determine which one of the cores … was responsible for the lockstep mismatch.").

67. As another example, lockstep is not recoverable if the mismatch error is triggered before the detection of the error in the individual processor. Nguyen, [0022] (triggering a non-recoverable FRC error before the underlying parity/ECC error in the processors is detected). Lockstep is also not recoverable if the mismatch

---

[2] A POSITA would have understood that the acronym "ECC" refers to error checking and correction, which includes techniques for coding data such that errors can not only be detected, but also corrected.

error is detected in the absence of detecting an error in the individual processor. Marshall, 2:17-34 (if an error is detected based on comparing outputs of the two processors, the system is stopped since comparison checks indicate only that there was a failure but do not indicate which processor failed). Such mismatch errors typically indicate a high risk of corrupt data or indicate incorrect results about to propagate through the system. Kondo, [0089] (divergence detection is indicative of potential error or data corruption about to propagate); Kondo, [0102] (divergence detection in the absence of detecting errors in the processors typically would indicate a more serious error).

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

68. It was known to a POSITA that once an error was detected in relation to FRC processors, the appropriate action responsive to the recoverability of the detected error should be taken to handle both non-recoverable and recoverable errors so as to reduce the impact of the errors on the system.

69. It was known for non-recoverable errors, for example, that there is risk of corrupt data or results propagating through the system. An appropriate action for responding to a non-recoverable error would have been immediately resetting or rebooting the system to return the system to a safe state or shut down the processor. Nguyen, [0007] (FRC errors are detectable but not recoverable; the FRC error handling routine typically resets the system, which is time consuming and reduces

system availability); Marshall, 2:3-6 (the only recovery action responsive to an output mismatch is to reset the system); Kondo, [0039] (conducting a hard-reset of the lockstep computer processing system in response to a non-recoverable error), [0190] (conventional non-recoverable error handling entails shutting down a processor, losing it as a resource, followed by repair/replacement); Schultz, [0030] ("When an error is not recoverable, the system may be rebooted to return it to a safe state"); Marisetty, 5:21-28 (for errors that cannot be corrected, the system will need a reboot and execution cannot continue).

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

71. For recoverable errors, it was known that various recovery routines have been used for handling such errors to reduce the impact to the overall system, such as those disclosed in Bigbee, Nguyen, Kondo, and Safford. Kondo, [0191] (structuring error recoveries to ensure data corruption will not leave the processor).

72. Treating an otherwise recoverable error as a non-recoverable error would reduce system availability. Recovering lockstep from a recoverable error can

be implemented with lower latency as compared to recovering lockstep in response to a non-recoverable error, which typically involves the reset or the repair/replacement of a processor. Nguyen, [0022]-[0023], [0040].

73. Similarly, treating a non-recoverable error as a recoverable error would reduce system reliability. For example, handling a non-recoverable error as a recoverable error would create risks of having corrupt data or results propagating through the system. Kondo, [0101] (the detection and signaling of an error may be taken as an indicator of imminent divergence; the longer time that passes from error or divergence detection without remedial action, the more serious the impact of the error or divergence will be); Marisetty, 2:38-50 (other processors of a multiprocessor system may continue execution without knowledge of the error, propagating the error and causing further damage in the system), 2:51-52 (multiprocessor systems may have to reboot or shut down for errors because of a lack of proper error handling).

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

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

75. It was also known that firmware can be used for handling processor errors. Nguyen, [0045] (using PAL or SAL layers of BIOS— firmware—for

implementing error recovery mechanisms in Itanium® family processors); Marisetty, FIGS. 1-2, 5:48-8:64 (teaching the use of firmware including PAL and SAL to execute error handling routines); Schultz, FIGS. 1-3, [0017]-[0018] (teaching the use of firmware including PAL and SAL for correcting an error that is not correctable directly by hardware); Bigbee, [0014]-[0025].

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

76. It was known to a POSITA to designate a particular processor as the boot processor in a multi-processor system. Natu, 1:5-9, 2:20-36; Goodrum, 2:15-31, 2:40-43, 6:1-3, 6:49-51, 8:60-61; Suffin, 12:3-18, 13:16-20. It was also known that the designated boot processor would be responsible for turning on and testing the other processors or otherwise initializing the overall system. Goodrum, 2:15-31; Natu, 2:20-36. If a designated boot processor does not function properly, then the overall system may become incapacitated as the remaining processors could not be turned on or the overall system could not be initialized, for example, because the operating system would not be booted. Goodrum, 2:15-31; Natu, 2:20-50.

77. It was therefore known to a POSITA to establish a backup processor to take over boot processor functions should the designated boot processor fail. Goodrum, 1:17-20, 3:7-31, 8:60-64, 9:39-50; Suffin, 2:18-3:11, 13:20-14:2, Fig. 4A; Natu, 1:5-9, 2:57-3:13. By having a backup processor fulfill the responsibilities of the initially designated boot processor, the reliability of the overall multi-processor

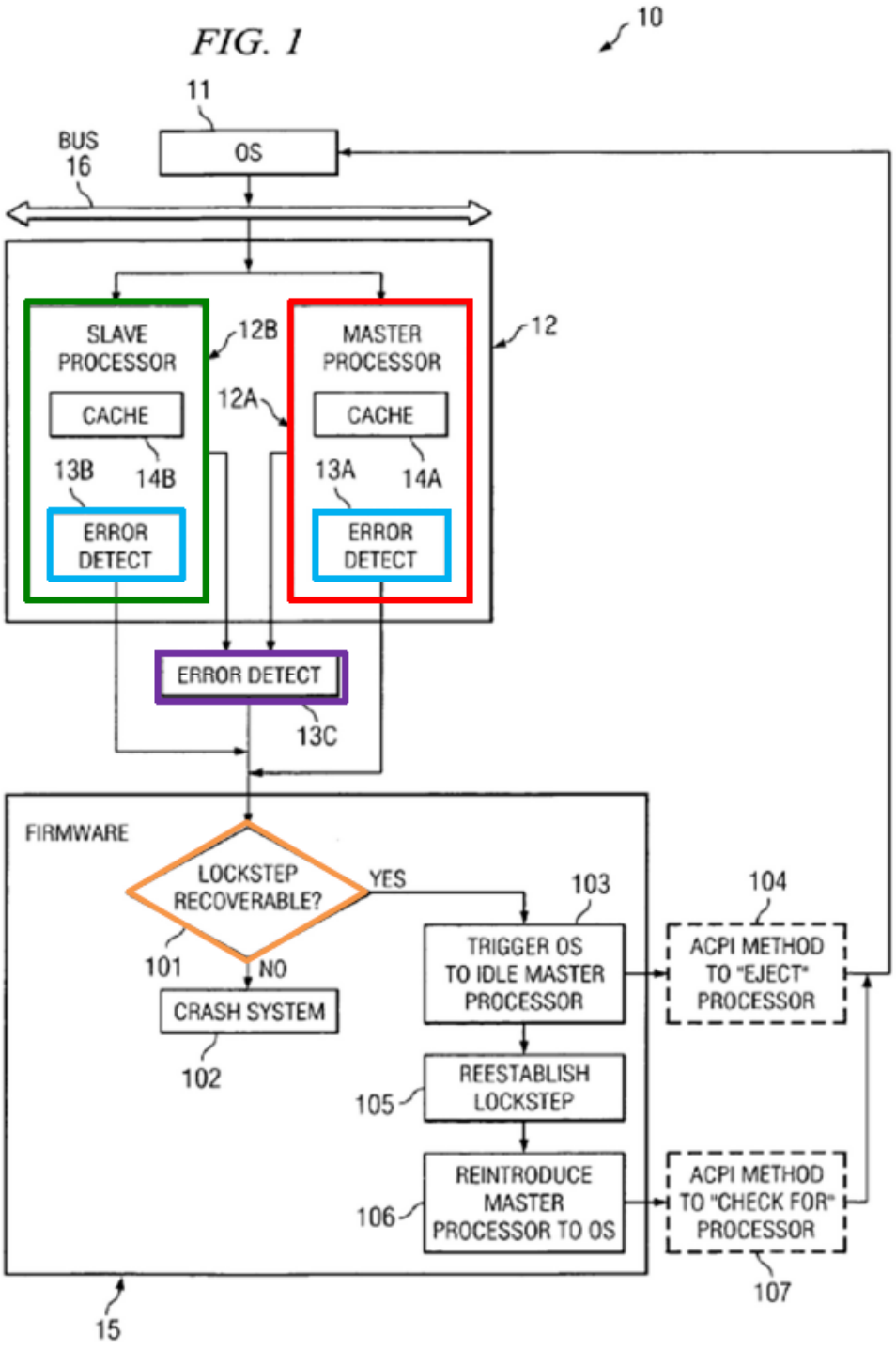
system is improved. Goodrum, 2:15-31; Natu, 2:20-50. Further, allowing another processor to take on the role of boot processor enables the system to diagnose errors with the failed boot processor. Absent a backup boot processor, a reset of the system is typically necessary. The reset clears the operational status of the system, including operational status information associated with the failed boot processor, thereby often preventing the system from being able to determine why a particular failure of the boot processor occurred, unless a backup boot processor is used. Suffin, 1:1-5, 1:12-3:3, 16:9-17:11.

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

VI. OVERVIEW OF THE '958 PATENT

79. The '958 Patent was filed on October 25, 2004. Ex. 1001, 1. The '958 Patent discloses a system and method for detecting loss of lockstep (LOL) between pairs of processors and LOL recovery. Ex. 1001, 3:51-59; FIGS. 1-6. The '958 Patent describes a system that includes a lockstep processor pair, including a master processor and a slave processor, each with error detect logic to detect errors present in the respective processor. *Id.*, 4:14-46, FIG. 1 (reproduced below). The system

also includes an error detect logic to detect a lockstep mismatch between the master and slave processors. *Id.*, 4:46-55. The specification describes detecting LOL as including at least one of (1) detecting mismatch of the output of the paired processors ("lockstep mismatch error"); and (2) detecting an error in either of the processors that may eventually propagate to a lockstep mismatch error. *Id.*, 2:26-59, 4:37-67. The specification also refers to detecting an error in either of the processors by the error detect logic before detecting a lockstep mismatch as a detection of a "precursor to lockstep mismatch." *Id.*, 4:60-67.



Ex. 1001, FIG. 1 (annotated).

80. Once the system detects LOL, the system uses firmware to determine whether lockstep is recoverable (or whether the detected LOL is a recoverable LOL);

if recoverable, the system's firmware cooperates with the operating system ("OS") via standard OS methods to recover the lockstep. *Id.*, 5:49-6:1, FIG. 1 (step 101). If the OS is Advanced Configuration and Power Interface (ACPI) compatible, the firmware uses ACPI methods to interact with the OS to idle the processors, recover lockstep of the processors, and reintroduce the processors to the OS. *Id.*, 6:9-7:21, FIG. 1 (steps 103-107). In the event the loss of lockstep is in the boot processor (the processor from which the system boots up), the system will transfer the role of boot processor to a "hot spare" processor. *Id.*, 9:15-29, 10:7-12, 10:26-31, FIG. 3. Loss of lockstep is then recovered in the processor that was the boot processor, and that processor becomes the new hot spare processor in the event the current boot processor loses lockstep. *Id.*

81. Independent claims 1 and/or 23 of the '958 Patent recite six primary components, all of which were disclosed in the prior art and all of which would have been obvious to a POSITA at the time:

1) Detecting loss of lockstep between a pair of lockstep processors;
2) With firmware, determining if lockstep is recoverable;
3) With firmware, determining if lockstep mismatch has occurred;
4) Triggering an operating system to idle the processors;
5) Attempting to recover lockstep; and

6) If recovered, triggering the operating system to recognize the processors to be available.

VII. THE PROSECUTION HISTORY OF THE '958 PATENT

82. As I note, I have reviewed the prosecution history of the '958 Patent (Ex. 1004).

VIII. OVERVIEW OF THE RELIED UPON PRIOR ART

A. Overview of Bigbee (Ex. 1005)

83. Bigbee discloses a data processing system including hardware 102, firmware 104, and an operating system 106. Bigbee, [0014]-[0025], FIG. 1 (reproduced below).

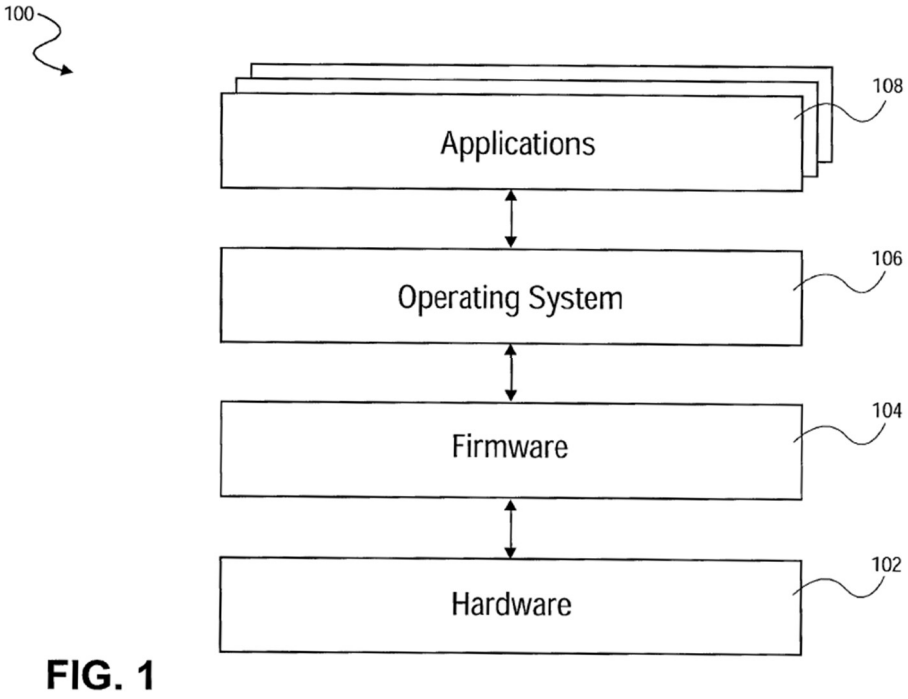


FIG. 1

Bigbee, FIG. 1.

84. Hardware 102 includes two or more processor packages, with each processor package including two processor cores (e.g., Itanium® family processors) operating in lockstep FRC mode as a single logical processor from the point of view of the operating system 106. *Id.*, [0019], [0032]-[0033], FIG. 3 (reproduced below); *see above* Section V.A. The two processor cores concurrently execute identical instructions on identical data and each processor generates results. *Id.*, [0002], [0033]. FRC logic 308 monitors and detects inconsistencies in the results of the two processor cores that indicate an FRC mismatch error, which may be the result of an alpha particle impacting on one or both processor cores. *Id.*

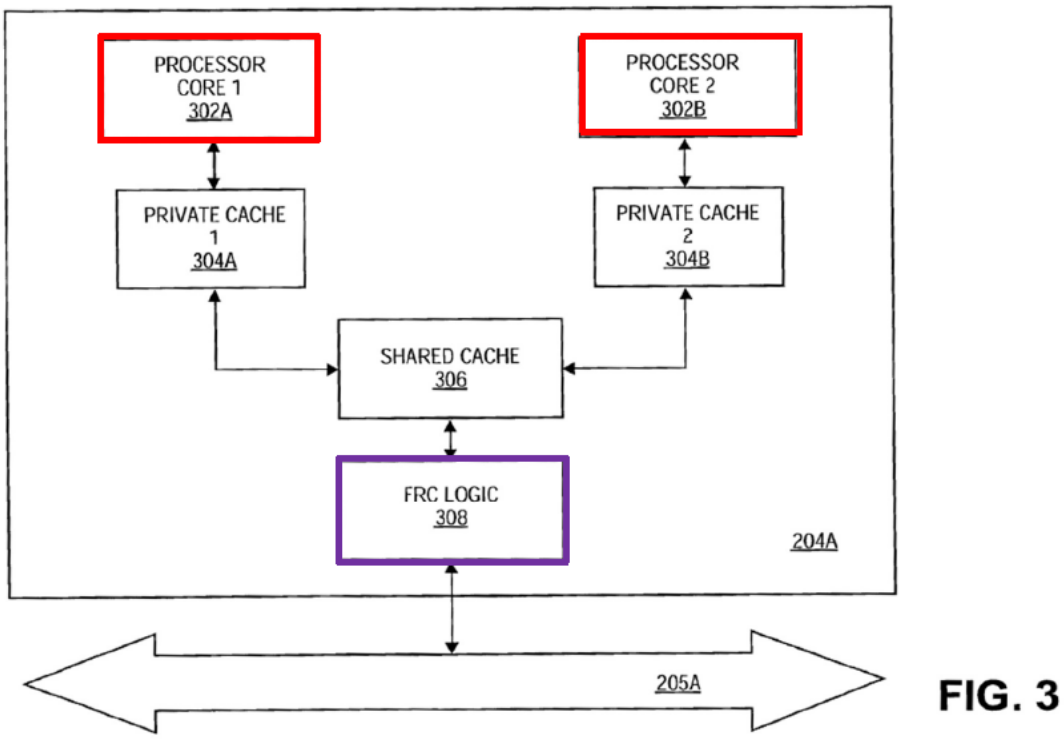


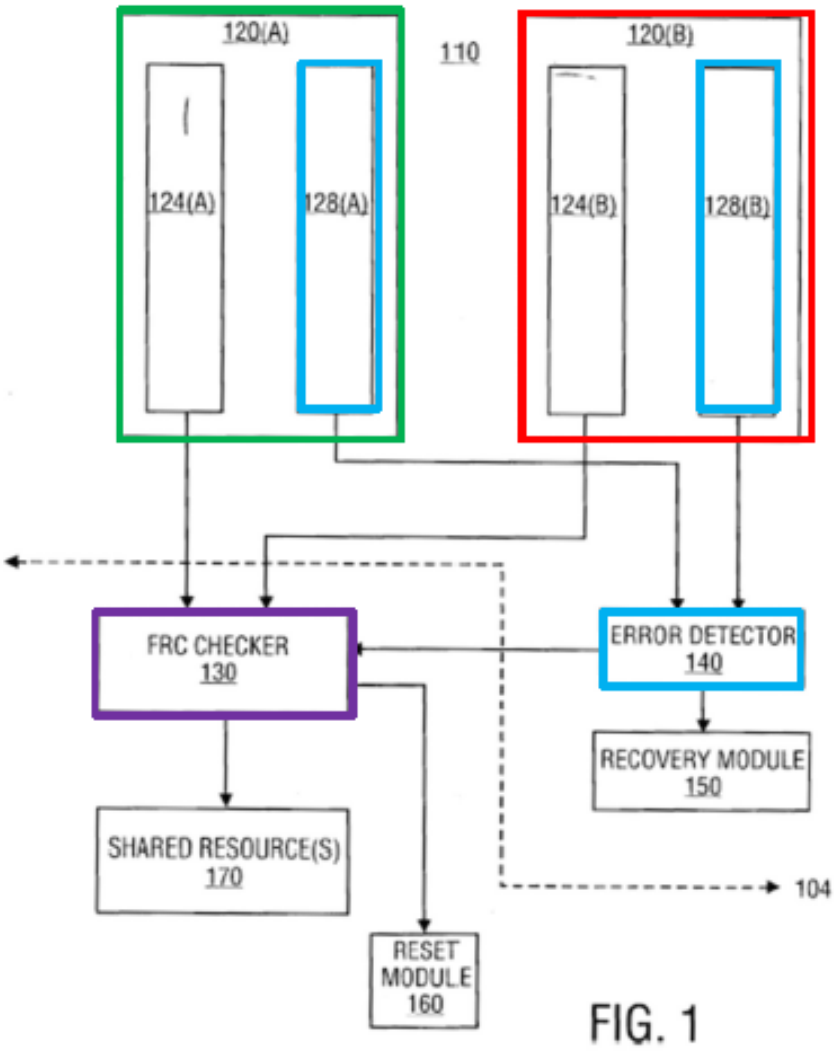
FIG. 3

Bigbee, FIG. 3 (annotated).

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

B. Overview of Nguyen (Ex. 1006)

86. Nguyen discloses a processor including two cores 120A and 120B that operate in an FRC mode ("lockstep"), an FRC checker 130 that compares results from the two cores, and an error detector 140 that detects errors in the respective cores. Nguyen, Abstract, [0006], [0020]-[0022], [0039], [0045], FIG. 1 (reproduced below). Each execution core 120 includes a data pipeline 124 and an error pipeline 128 that feed into both FRC checker 130 and error detector 140. *Id.*, [0021]. Error pipeline 128 represents logic that operates on various types of data to detect errors in the data and to provide appropriate signals, such as parity or ECC error flags, to detector 140. *Id.* Nguyen further discloses that one of the two cores "may be designated as the master core and the other as the slave core." *Id.*, [0047].



Nguyen, FIG. 1 (annotated).

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

88. ***First***, Nguyen discloses lockstep is not recoverable from an FRC error detected by FRC checker 130. *Id.*, [0007], [0022], [0040]. If FRC checker 130 signals an FRC error indicating a mismatch before error detector 140 detects an underlying soft error, the FRC error is not recoverable and a reset module 160 resets the system. *Id.*, [0007], [0022]-[0023]. This error is a mismatch error where the processor causing the underlying soft error cannot be determined, thereby requiring the whole system to be reset.

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

90. ***Third***, Nguyen discloses lockstep is recoverable from a mismatch error detected before an FRC error is triggered. *Id.*, [0056]-[0058]. As shown below, Nguyen's FRC checker includes a compare unit 734 and a timer unit 738. *Id.* Compare unit 734 compares the data from both cores and sets a status flag to indicate if the comparison yields a match. *Id.* If the data does not match, compare unit 734 sets the status flag to indicate the mismatch and triggers timer unit 738 to begin a

countdown interval. *Id.* If error detector 140 receives an error flag before the timeout interval expires, it disables FRC checker 730 and triggers recovery module 150. *Id.* But if the timeout interval does expire, the FRC checker signals a non-recoverable FRC error and triggers reset module 160 to reset the system. *Id.*, [0022], [0056]-[0058]. Nguyen's process for determining if lockstep is recoverable from an error in a lockstep processor pair was known to a POSITA and disclosed in the prior art. *See e.g.*, Tu (Ex. 1017), [0041]-[0043], FIG. 5.

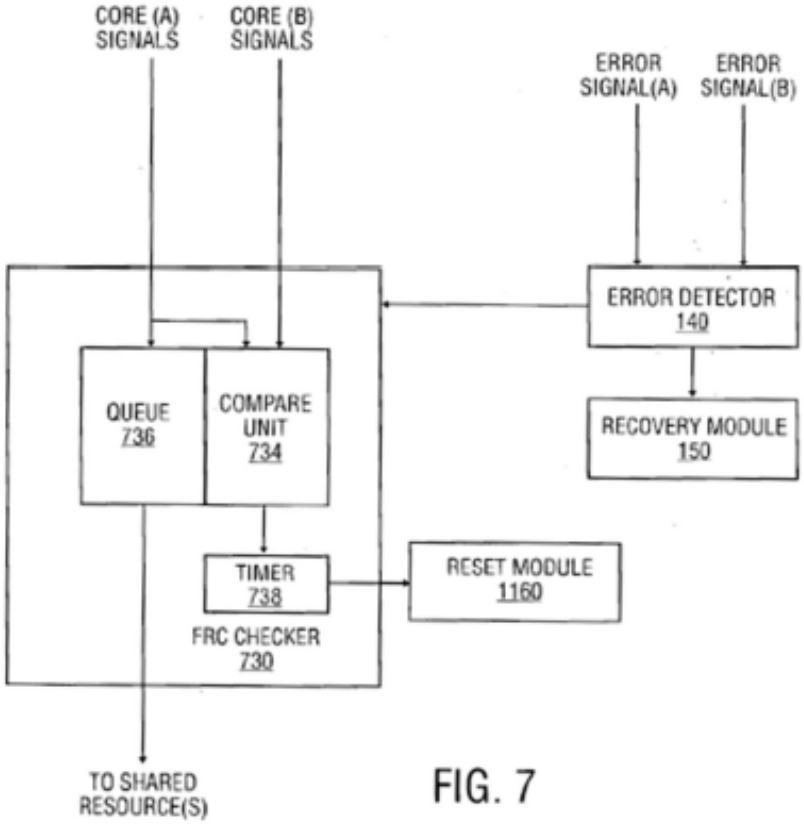


FIG. 7

Nguyen, FIG. 7.

C. Bigbee and Nguyen are Analogous Art to the '958 Patent

91. Like the '958 Patent, both Bigbee and Nguyen relate to fault tolerant systems that operate despite loss of lockstep in the system. A POSITA would have understood that, like the '958 Patent, Bigbee's processor package's two cores or Nguyen's processor's two cores, both operating in FRC mode or lockstep, are a lockstep pair of processors.[3] Safford-181, 1:67-2:3 ("Because lockstepped processors execute redundant instruction streams, lockstepped systems are said to perform a 'functional redundancy check.'").

92. A POSITA would also have understood that Bigbee's detecting errors within the individual processor cores by its FRC logic and Nguyen's detecting FRC errors, soft errors, or mismatch errors by its error detector and FRC checker each would constitute detecting LOL as claimed. *See* Section X.A, *infra*.

93. A POSITA would have understood that, like the '958 Patent, both Bigbee and Nguyen disclose that FRC mode or lockstep of the processor cores is

---

[3] A POSITA would have understood that an FRC processor or a lockstep processor refers to a processor including two or moreprocessor cores operating in FRC mode or in lockstep.

recoverable from a detected recoverable error by implementing a recovery routine. Bigbee, [0037] (teaching re-enabling FRC operation of the processor cores); Nguyen, [0046] (teaching that FRC mode is restored). And both Bigbee and Nguyen disclose using firmware, such as a processor abstraction level (PAL), a system abstraction level (SAL), or a basic input/output system (BIOS), to implement a recovery routine to recover lockstep of the processor cores. Bigbee, [0015], [0018]-[0023], [0035]-[0037]; Nguyen, [0045]-[0046]. Accordingly, Bigbee and Nguyen are analogous to the '958 Patent.

D. Overview of Suffin (Ex. 1020)

94. Suffin, like the Bigbee and Nguyen references, discloses a computer system having multiple redundant CPUs and associated controllers, operating in lockstep. Suffin, 6:1-7:14, FIG. 2. When a failure occurs in a processor, such as can occur from a loss of lockstep (*Id.*, 7:12-16), Suffin describes using an ordered list of processors (a "boot list") to determine which processor to use as a boot processor. *Id.*, 2:5-4:5, 11:3-9, 11:23-12:18, 13:1-7, 13:16-14:10. If booting up using a first processor is unsuccessful, then the system selects the next processor on the list to be the boot processor. *Id.* When recovering from a system failure, if a processor is suspended or not operating, then that processor is skipped on the boot list, so that the next processor on the list is used for rebooting. *Id.*, 16:20-17:11, 13:1-7. Further, the state of the processor that failed may be copied for analysis. *Id.*

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

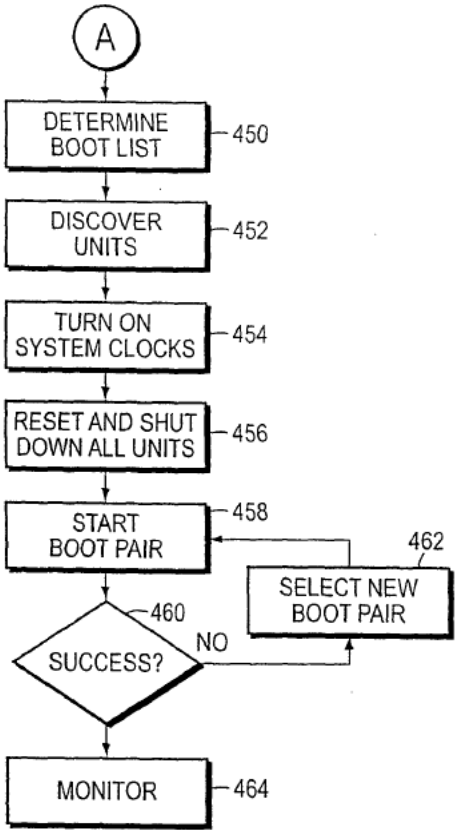


FIG. 4A

Suffin, FIG. 4A.

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

4A. Suffin further teaches that the system can iterate the boot list, to avoid repetitive failures that could be caused by trying to boot from a failed processors, which allows another (e.g., backup processor) to remain the boot processor and the original boot processor to become a backup processor. *Id*. 16:20-17:11.

E. Overview of Goodrum (Ex. 1021)

97. Goodrum, like Suffin, discloses a multiple processor system and methods for switching the role of boot processor to another processor in the event of a failure of the boot processor. For example, "a hot spare boot circuit … automatically reassigns the power up responsibilities to an operational second processor should the primary processor fail." Goodrum, 3:10-13, 8:60-64, 9:39-50, 18:34-49, claims 1, 4, 7, 10, FIG. 2. Goodrum generally uses a timer to determine that the boot processor has failed and thus needs to reassign the role of boot processor to another of the processors. *Id.*

F. Overview of Safford (Ex. 1007)

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

[0007], [0021]. This process is shown, for instance, in Figure 4, step 220 (Safford, [0027]):

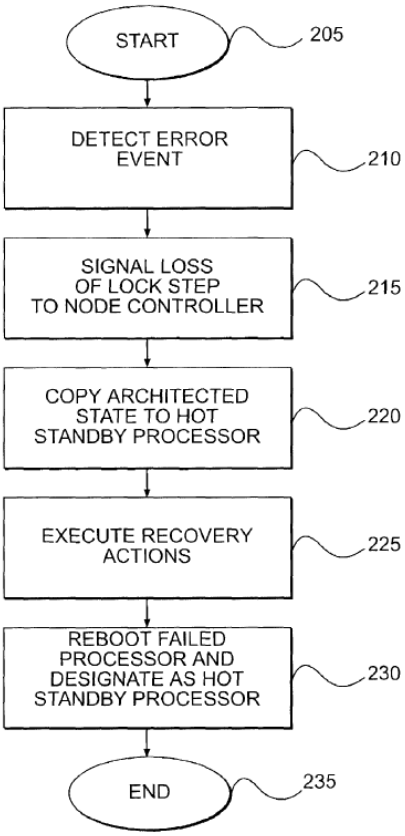


FIG. 4

Safford, FIG. 4.

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

G. Suffin, Goodrum, and Safford are Analogous Art to the '958 Patent

100. Like the '958 Patent, Suffin, Goodrum, and Safford relate to fault tolerant multi-processor systems that operate despite the occurrence of an error in a processor. Suffin explains that it "is directed to methods and apparatus for robust operation of a fault-tolerant computer system with redundant components." Suffin, 2:5-6. Like the '958 Patent, Suffin provides a spare or backup processor for a system boot processor when it fails. *Id.*, 2:6-9 ("It provides methods and apparatus for booting a computer system with redundant hardware and/or software components in a deterministic fashion."), 2:18-21. Goodrum similarly describes providing a spare or backup processor to an initially designated system boot processor that experiences an error. Goodrum, 1:17-20 ("The invention relates to multiprocessor computer systems, and more particularly, to a circuit for reassigning the power-on processor in a dual processor system when a processor fails."), 3:7-10 ("It is therefore an object of the present invention to identify an operational microprocessor in a multiprocessor system so that the system can be properly powered up when the primary microprocessor is nonoperational."). Safford, like the '958 Patent, describes providing a hot standby processor to speed recovery from a loss of lockstep experienced by a processor pair. Safford, [0001], [0006]-[0007], [0018], [0020]. Accordingly, Suffin, Goodrum, and Safford are analogous to the '958 Patent.

IX. THE RELEVANT ART AND LEVEL OF ORDINARY SKILL IN THE RELEVANT ART

101. I understand that my assessment of the claims of the ’958 Patent must be undertaken from the perspective of what would have been known or understood by a person having ordinary skill in the art, reading the ’958 Patent on its relevant priority date and in light of the specification and file history of that patent. In this declaration I refer to such a hypothetical person as a “POSITA”. In determining the characteristics of a POSITA at the time of the claimed invention (which I understand to be October 25, 2004), I considered several things, including (a) the types of problems encountered by those working in the field and prior art solutions thereto; (b) the sophistication of the technology in question, and the rapidity with which innovations occur in the field; (c) the educational level of active workers in the field; and (d) the educational level of the inventor.

102. In my opinion, the relevant field of art for the ’958 Patent is the field of fault-tolerant computing systems as of October 25, 2004, the filing date of the ’958 Patent. Ex. 1001, 2:13-15, 3:51-59. With over 35 years of experience in electrical and computer engineering, I am well acquainted with the level of ordinary skill required to implement the subject matter of the ’958 Patent.

103. In my opinion, at the time of the alleged invention of the ’958 Patent, a POSITA would have at least a bachelor’s degree in Computer Science, Computer Engineering, or equivalent, and at least two years of prior work experience with

computer architecture design, including fault tolerant computer architecture design. Additional education could substitute for professional experience and vice versa.

104. My analysis and opinions regarding the claims of the '958 Patent have been based on the perspective of a POSITA as defined above as of October 25, 2004 (the filing date of the '958 Patent). In this Declaration, wherever I refer to a POSITA, I am referring to a POSITA as of October 25, 2004, as I have defined such a hypothetical person in this section.

X. CLAIM CONSTRUCTION

105. As discussed in more detail above, I understand that, in an *inter partes* review proceeding, the claim terms under consideration are to be given their ordinary and customary meaning, as would be understood by a POSITA at the time of the invention, consistent with the specification and the prosecution history.

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

106. It is my opinion that the terms "detection of loss of lockstep" and "detecting loss of lockstep" should be construed to include both mismatch errors and errors that may eventually result in a mismatch error constituting a "precursor to lockstep mismatch." Ex. 1001, 4:56-67. Therefore, as used in the claims, the terms "detection of loss of lockstep" and "detecting loss of lockstep" should be construed as including at least one of: (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error.

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

108. These proposed constructions are also consistent with the claims. For example, dependent claims 4-8 reflect that detection of LOL should cover these two examples. Ex. 1001, 15:7-19, 15:30-49 (claims 1, 4-8).

109. Claim 4, for example, further defines “detecting loss of lockstep” as comprising “detecting corrupt data … ***that would lead to a lockstep mismatch***.” Ex. 1001, 15:30-34. Similarly, claim 5 defines “detecting loss of lockstep” as comprising “***detecting aprecursor to lockstep mismatch***.” Ex. 1001, 15:35-37. Both claims 4 and 5 expressly require that a precursor to lockstep mismatch (errors that may

eventually cause a lockstep mismatch error—definition 2) is included in the definition of "detecting loss of lockstep."

110. Claim 6 further defines "detecting loss of lockstep" as "detecting lockstep mismatch between the lockstep pair of processors," expressly requiring that lockstep mismatch (mismatch errors—definition 1) are included. Ex. 1001, 15:38-41.

111. Accordingly, it is my opinion that "detection of loss of lockstep" and "detecting loss of lockstep" should be construed as including at least one of (1) detecting a lockstep mismatch error; and (2) detecting an error that may eventually cause a lockstep mismatch error.

B. "Lockstep Is Recoverable" and "Loss of Lockstep Is Recoverable"

112. It is my opinion that the terms "lockstep is recoverable" and "loss of lockstep is recoverable" should be construed as "lockstep is recoverable from the detected loss of lockstep" because the specification only describes lockstep recovery from the detected LOL—"whether the detected LOL is a recoverable LOL." Ex. 1001, 5:49-62.

C. All Other Claim Terms

113. For all other claim terms, I have considered and applied their plain and ordinary meaning as they would have been understood by one skilled in the art at the time of the alleged invention and consistent with the specification.

XI. SUMMARY OF ANALYSIS

114. Based on my study of the references listed in the table below, as well as on my education and years of experience in the relevant art, it is my opinion that those references in various combinations render obvious all of the elements of the claims shown therein. In the table, I have associated each of the '958 Patent claims with a specific combination of references, following the legal principles that I explained in Section IV above, and using the *Phillips* standard discussed above in Section IV.D. Furthermore, in accordance with the legal principles that I explained in Section IV above, there is an appropriate basis to combine the references, as I explain below. I understand that the Petitioners have assigned labels (e.g., [1A], [1B], etc.) to different limitations of the claims to facilitate analysis of the claims. Accordingly, in my analysis below, I have referred to the claim limitations using those same labels, which are shown in the Claim Listing Appendix attached to this declaration as Appendix B.

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

XII. GROUND A: CLAIMS 9-12 ARE RENDERED OBVIOUS BY BIGBEE-NGUYEN IN VIEW OF GOODRUM

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

115. Claims 9-12 all depend from claim 1 (either directly or indirectly through other claims). As I explain below, it is my opinion that the elements of claim 1 are taught by the combination of Bigbee and Nguyen. It is my further opinion that the additional elements of claims 9-12 are taught by Goodrum and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Goodrum (among other combinations I discuss further below). Below, I provide an overview of the Bigbee-Nguyen combination that teaches the elements of claim 1.

116. It is my opinion that Bigbee discloses most of the limitations of claim 1. Bigbee discloses a pair of lockstep processors, using firmware and an operating system to detect loss of lockstep, and recovering from loss of lockstep using

firmware. Bigbee, therefore, discloses all elements of claim 1 except for those elements that require determining if the lockstep of a lockstep processor pair is recoverable.

117. Nguyen discloses a high-reliability processor that includes a lockstep processor pair along with a method of determining if the lockstep of the processor pair is recoverable or non-recoverable, before using firmware to recover lockstep if the error is recoverable. It is my opinion that Bigbee's firmware performing Nguyen's determining if the lockstep is recoverable or non-recoverable renders claim 1 obvious. In addition, a POSITA would have known that Nguyen's determination can be performed with firmware. *See above* Section V.D.

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

Core Claim Requirement	Prior Art Reference
1) Detecting loss of lockstep between a pair of lockstep processors	Bigbee
2) With firmware, determining if lockstep is recoverable	Bigbee (with firmware) Nguyen (determining if lockstep is recoverable)
3) With firmware, determining if lockstep mismatch has occurred	Bigbee

4) Triggering an operating system to idle the processors	Bigbee
5) Attempting to recover lockstep	Bigbee
6) If recovered, trigger the operating system to recognize the processors to be available	Bigbee

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

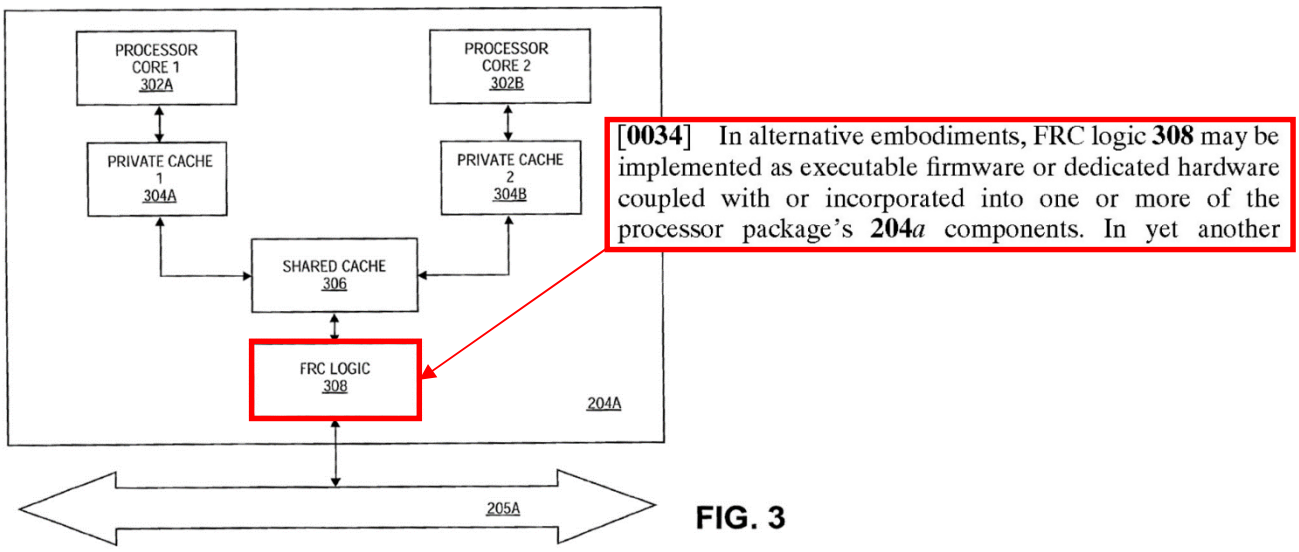
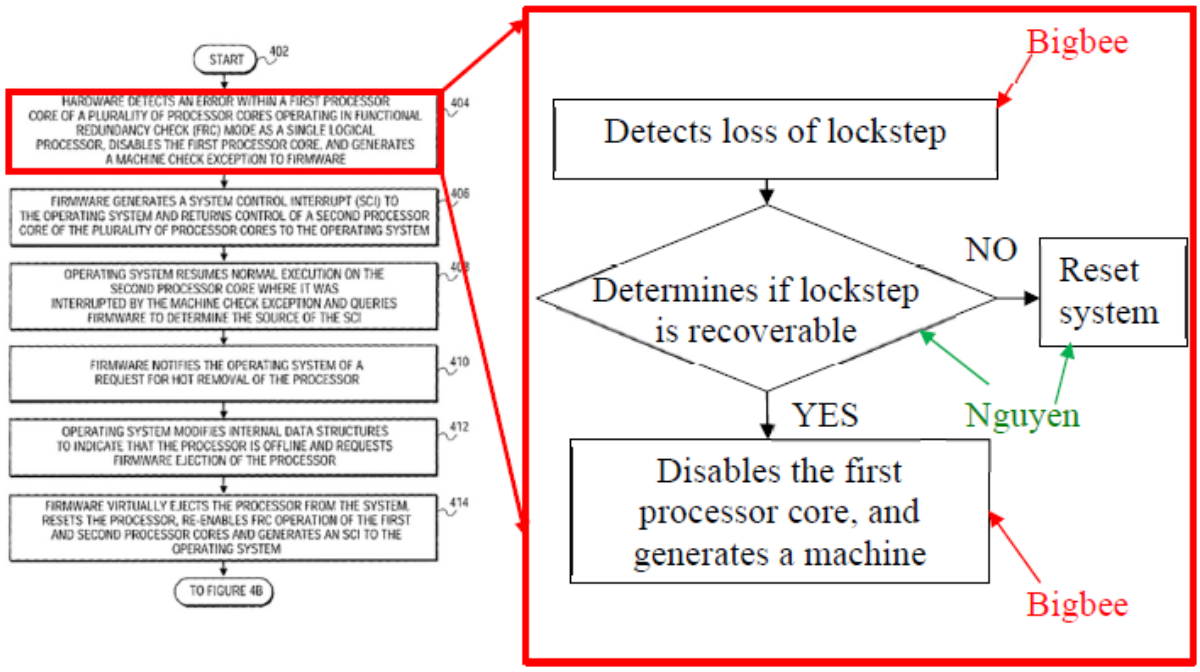


FIG. 3

Bigbee, FIG. 3 (annotated).



Bigbee, FIG. 4A (annotated, modified).

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

120. By the time of the '958 Patent, a known problem in the art was addressing errors causing loss of lockstep in a lockstep pair of processers that were both recoverable and non-recoverable. It is my opinion that a POSITA would have understood that a complete and robust system would be able to handle both recoverable (e.g., soft errors) and non-recoverable errors (e.g., mismatch errors). Bigbee only addresses handling recoverable errors. But diagnosing and managing non-recoverable errors was a known solution to the problem at the time. Nguyen discloses one known solution to addressing both recoverable and non-recoverable errors, explaining the benefits of being able to determine whether an error is

recoverable or non-recoverable, and addressing the error accordingly. Bigbee's recoverable error specific system would have been understood by a POSITA to be ready for the improvement of Nguyen's determination and non-recoverable error responsive system to provide the known benefits of reducing impact of the detected errors, and increasing the system's fault tolerance and availability by reducing system downtime.

121. Bigbee discloses detecting errors within a lockstep pair of processors using FRC logic including detecting both recoverable (e.g., soft errors) and non-recoverable errors (e.g., mismatch errors when the processor causing the error cannot be determined). Bigbee, [0033]. But Bigbee only discloses a responsive routine for recovering from recoverable errors. *See above* Section VIII.A. Bigbee does not disclose handling non-recoverable errors. Because Bigbee only discloses recovering from recoverable lockstep errors, Bigbee does not disclose performing a threshold determination of whether detected errors are recoverable or non-recoverable. Bigbee, [0019], [0035], [0037] (error recovery routine re-enables FRC operation of the processor cores).

122. A POSITA would have recognized that Bigbee's disclosure omits the portion of the process after a non-recoverable error is detected (e.g., a mismatch error when the processor causing the error cannot be determined) and would have

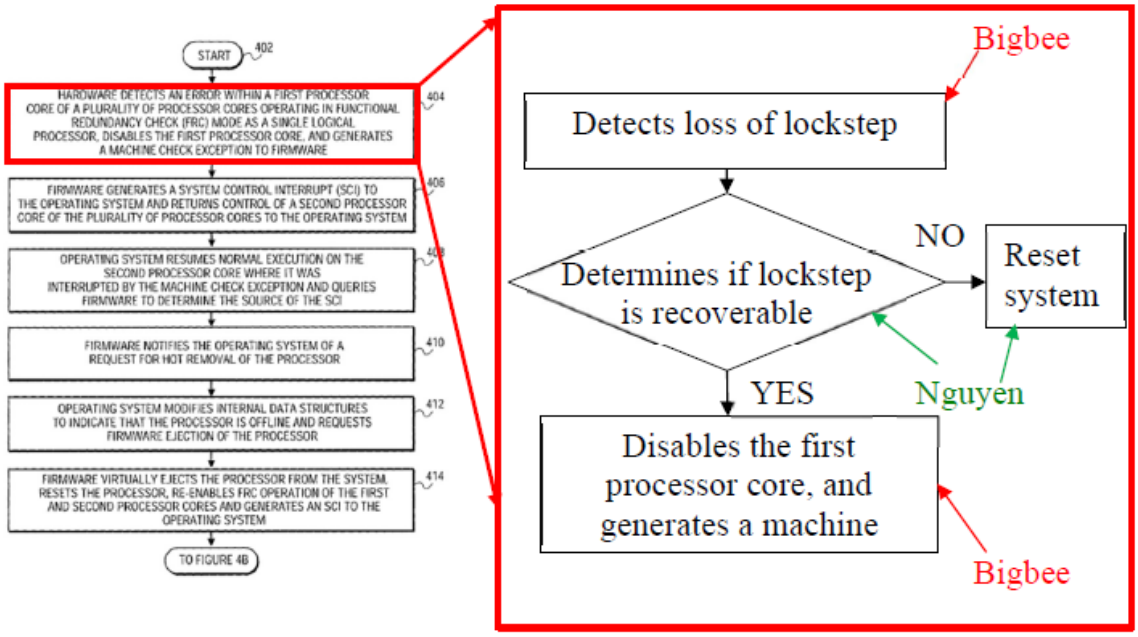
been motivated to add a process for handling non- recoverable errors. Nguyen is one such reference disclosing such a process.

123. Nguyen discloses detecting errors within a lockstep pair of processors and determining if lockstep is recoverable from the detected errors using an error detector and FRC checker before taking an appropriate error handling routine. Nguyen, [0056]-[0057].

124. A POSITA would have found it obvious to incorporate Nguyen's determination of whether the errors detected by Bigbee's FRC logic were recoverable or non-recoverable to properly handle the detected errors, reduce the impact of the detected errors, and increase system fault tolerance and availability by reducing system downtime. This was expressly recognized by the prior art. Bigbee, [0002] (it is desirable to reduce the amount of "downtime" resulting from component failures, faults, or internal errors of system with FRC processors); Nguyen, [0008]-[0009] (handling otherwise recoverable errors as non-recoverable errors, such by resetting the system, reduces system availability and efficiency); Kondo, [0002] (recovering FRC processors from recoverable errors reduces the impact of the recoverable error).

125. Combining Bigbee and Nguyen would result in a Bigbee-Nguyen system that would have been capable of handling both recoverable and non-recoverable errors efficiently. Taking step 404 of Bigbee's Figure 2, and adding

Nguyen's step of determining whether lockstep is recoverable after Bigbee detects loss of lockstep, would address both recoverable and non-recoverable errors, as the following figure shows.



Bigbee, FIG. 4A (annotated, modified).

126. As I noted, it is my opinion that a POSITA would have been motivated to build a system that would handle both recoverable and non-recoverable errors. Because Bigbee does not address handling non-recoverable errors, a POSITA would have been motivated to modify Bigbee's firmware FRC logic to perform Nguyen's disclosed determination of whether lockstep is recoverable from the detected errors to create a system that can handle both types of lockstep errors.

1. Motivation - A POSITA would have been motivated to add Nguyen's determining whether lockstep is recoverable from the detected error.

127. It is my opinion that a POSITA would have been motivated to implement Nguyen's determination of whether the detected error causing the loss of lockstep is recoverable by Bigbee's firmware FRC logic in order to appropriately handle the detected error. Because different responsive actions should be taken to handle each of non-recoverable and recoverable errors, determining whether the error is recoverable or non-recoverable is important to reduce the impact of the errors on the system. *See above* Section V.C; Marisetty, 4:43-5:63 (the processor of a multiprocessor system divides errors into four categories ranging from least severe to most severe), 7:24-27 (after detection of an error, the processor determines the severity of the error and takes action depending on the severity); Kondo, [0100] (determining whether an error is a) fatal to the processor or b) a correctable error that will produce a divergence but will not compromise data integrity); Schultz, [0027] (processor errors are classified into five different categories of increasing severity, including a firmware corrected category), [0154] (firmware SAL reports the appropriate error severity as being fatal or recoverable), [0021] (errors detected are classified according to the error type and severity).

128. As I discussed in Section V.C, non-recoverable errors cause a risk of data corruption or propagating an incorrect result if the system is not immediately rebooted or reset to return to a safe state. As I also explained in Section V.C, recoverable errors can be recovered using known recovery routines that reduce the

impact on the system. Therefore, it is my opinion that a POSITA would have understood that it is critical to handle recoverable errors differently from non-recoverable errors.

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

130. Likewise, handling non-recoverable errors differently from recoverable errors prevents reduction in system reliability. For example, handling non-recoverable errors as recoverable creates risks of having corrupt data or incorrect results propagating through the system. Kondo, [0101] (the detection and signaling of an error may be taken as an indicator of imminent divergence; the longer time that passes from error or divergence detection without remedial action, the more serious the impact of the error or divergence will be), [0103] (a nonrecoverable error involves the risk of data corruption); Marisetty, 2:38-50 (other processors of a multiprocessor system may continue execution without knowledge of the error, propagating the error and causing further damage in the system), 2:51-52

(Multiprocessor systems may have to reboot or shut down for errors because of a lack of proper error handling); Safford, [0003] (a data cache error in one processor, if not promptly corrected, may cause the computer system to crash).

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

132. Accordingly, it is my opinion that a POSITA would have understood that it would be desirable for fault tolerant systems to distinguish between recoverable and non-recoverable errors by first determining whether an error is recoverable and then responding accordingly based on the determination.

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

2. Reasonable Expectation of Success - A POSITA would have understood how to use Bigbee's disclosed firmware to perform Nguyen's determining whether a detected error is recoverable or non-recoverable.

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

135. It is my opinion that a POSITA would further have expected success in implementing Nguyen's determining step in Bigbee's firmware because Bigbee discloses that its system is compliant with well-known industry standards. For example, Bigbee discloses that its system is ACPI-compliant and its firmware includes ACPI BIOS code and a processor abstraction level (PAL) that provides an abstraction of implementation-specific processor features. Bigbee, [0016]-[0018].

136. A POSITA would have known how to use ACPI Source Language (ASL) to program Bigbee's firmware code to implement Nguyen's determining step using ordinary computer programming skills. Indeed, using such firmware, such as the PAL or SAL layer of ACPI BIOS, for implementing error diagnosis and recovery routines was well known and only involves ordinary skill and common knowledge. Schultz, [0027] (processor errors are classified into five different categories of

increasing severity, including a firmware corrected category), [0154] (firmware SAL reports the appropriate error severity as being fatal or recoverable); Nguyen, [0045] (using PAL or SAL layers of BIOS for implementing error recovery mechanisms in processors); Marisetty, FIGS. 1-2, 5:48-8:64 (teaching using firmware including PAL and SAL to execute error handling routines); Schultz, FIGS. 1-3, [0017]-[0018] (teaching using firmware including PAL and SAL for correcting an error that is not correctable directly by hardware).

137. A POSITA would have understood that firmware is one of three possible solutions—firmware, hardware, or combination—all of which would have been obvious to try, and any of which would be a suitable alternative for the others. A POSITA would have been familiar with logic design and would have understood the concepts of logic delays, timing, and the possibility of race conditions[4] as well the ability to handle such conditions using each of the alternatives.

---

[4] A POSITA would understand that a race condition may occur in a system, e.g., when a logic gate combines signals that travel different paths from the same source, thereby causing the inputs to the logic gate to change at different times in response to a change in the same source signal.

138. A POSITA would have understood the benefits of implementing Nguyen's determination in firmware. As a POSITA would have understood, firmware is flexible. Firmware can be modified after production, updated, and supported relatively quickly and inexpensively. Hardware, on the other hand, is inflexible. Once manufactured, hardware cannot be easily modified, and any modification or revision is often expensive because it requires remanufacture of new components. Therefore, a POSITA would have understood that a firmware implementation of Nguyen's determining step would have been desirable.

C. Independent Claim 1

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

139. It is my opinion that Bigbee teaches element [1A].

140. Bigbee discloses that the FRC logic detects loss of lockstep between the lockstep pair of processors and further discloses implementing FRC logic by executable firmware. Bigbee, [0019], [0034].

141. As shown below, Bigbee discloses a processor package including two processor cores, each with a private cache, which are coupled to a shared cache. Bigbee, FIG. 3, [0032]. The two processor cores function as a single logical processor and operate in an FRC mode ("lockstep pair of processors"), executing identical instructions on identical data to generate results for storage in associated private cache and/or the shared cache. Bigbee, [0033]. Bigbee discloses "detecting

loss of lockstep for a lockstep pair of processors" because the FRC logic detects both inconsistencies in the generated results that indicate an FRC mismatch error and errors within the processor cores of a processor package that cause a break in the lockstep between the cores, disabling FRC mode operation. Bigbee, [0019], [0033]; *see above* Section VIII.A. Bigbee further discloses implementing the FRC logic as executable firmware. Bigbee, [0034]; *see also* Safford-181, 15:60-64.

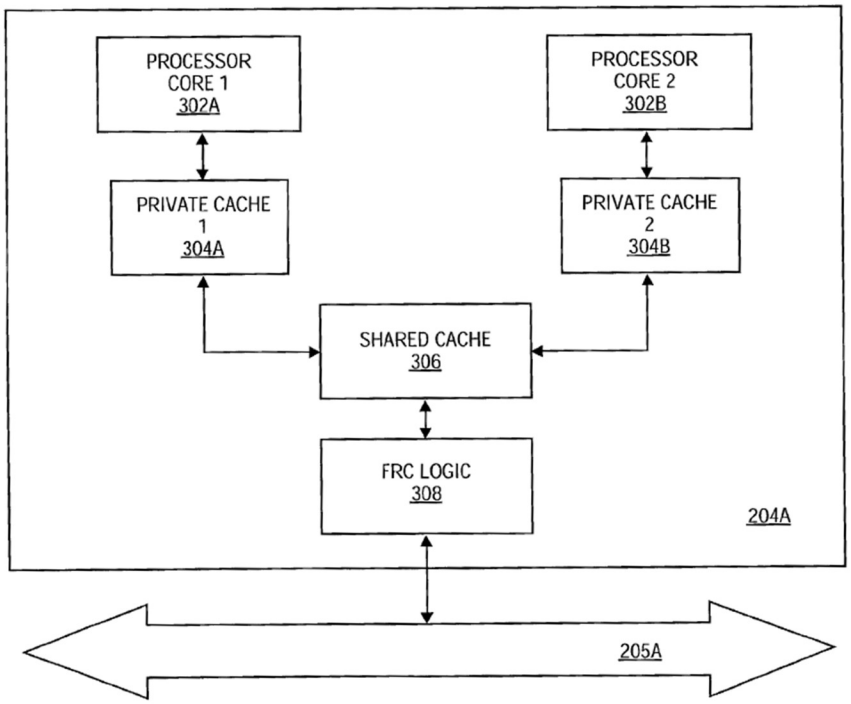


FIG. 3

Bigbee, FIG. 3.

2. Limitation [1B]: "using firmware to trigger an operating system to idle the lockstep pair of processors, recover lockstep for the lockstep pair of processors, and trigger the operating system to recognize the lockstep pair of processors having recovered lockstep"

142. Bigbee in view of Nguyen teaches this claim element.

143. As I explained above, a POSITA would have incorporated Nguyen's determining if the lockstep is recoverable into Bigbee to take an appropriate action responsive to the recoverability of the error detected in the lockstep processor pair. *See above* Section XII.B.1. Bigbee in view of Nguyen further teaches, responsive to determining if the lockstep is recoverable from the detected error, that the firmware triggers the operating system to idle the processors and to recover the lockstep between the pair of processors.

144. First, Bigbee discloses "using firmware to trigger an operating system to idle the lockstep pair of processors." As shown in Figure 4A below, Bigbee discloses that firmware (e.g., SAL program code) generates a system control interrupt (SCI) to the operating system. Bigbee, [0035], FIG. 4A (block 406). In response to the SCI, the operating system queries firmware (ACPI BIOS) to determine the source of the SCI; the firmware responds to the query by notifying the operating system of a request for processor hot removal (e.g., by an ACPI Notify

command).[5] Bigbee, [0036], FIG. 4A (blocks 408-410) ("trigger an operating system"). The operating system then modifies its internal data structures to indicate that the processor is offline and requests firmware ejection of the processor, e.g., by executing an _EJX control method ("to idle the lockstep pair of processors"). Bigbee, [0036]-[0037], FIG. 4A (block 412); Ex. 1022, 162-163.

[5] The term "processor" used in Bigbee's error recovering routine refers to the single logical processor that includes the pair of processor cores operating in lockstep. *See* Bigbee, [0032], [0035]-[0038].

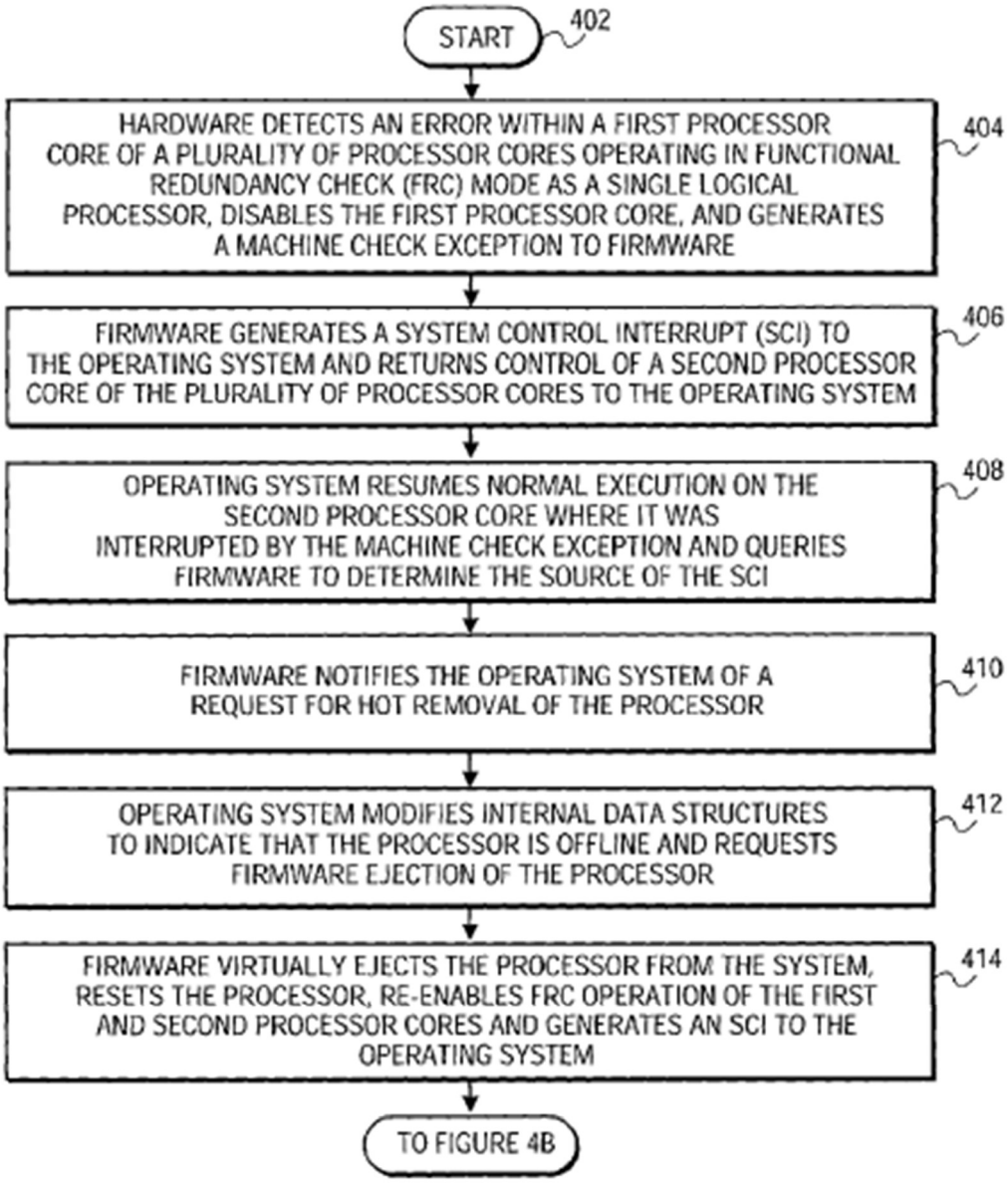


FIG. 4A

Bigbee, FIG. 4A.

145. Second, Bigbee discloses "using firmware to … recover lockstep for the lockstep pair of processors." In response to the operating system's ejection request, Bigbee's firmware virtually ejects the processor from the system, resets the

processor, re-enables FRC operation of the processor cores ("recover lockstep for the lockstep pair of processor"), and generates an SCI to the operating system. Bigbee, [0037], FIG. 4A (block 414).

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

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

lockstep pair of processors having recovered lockstep"). Bigbee, [0037], FIG. 4B (e.g., block 428).

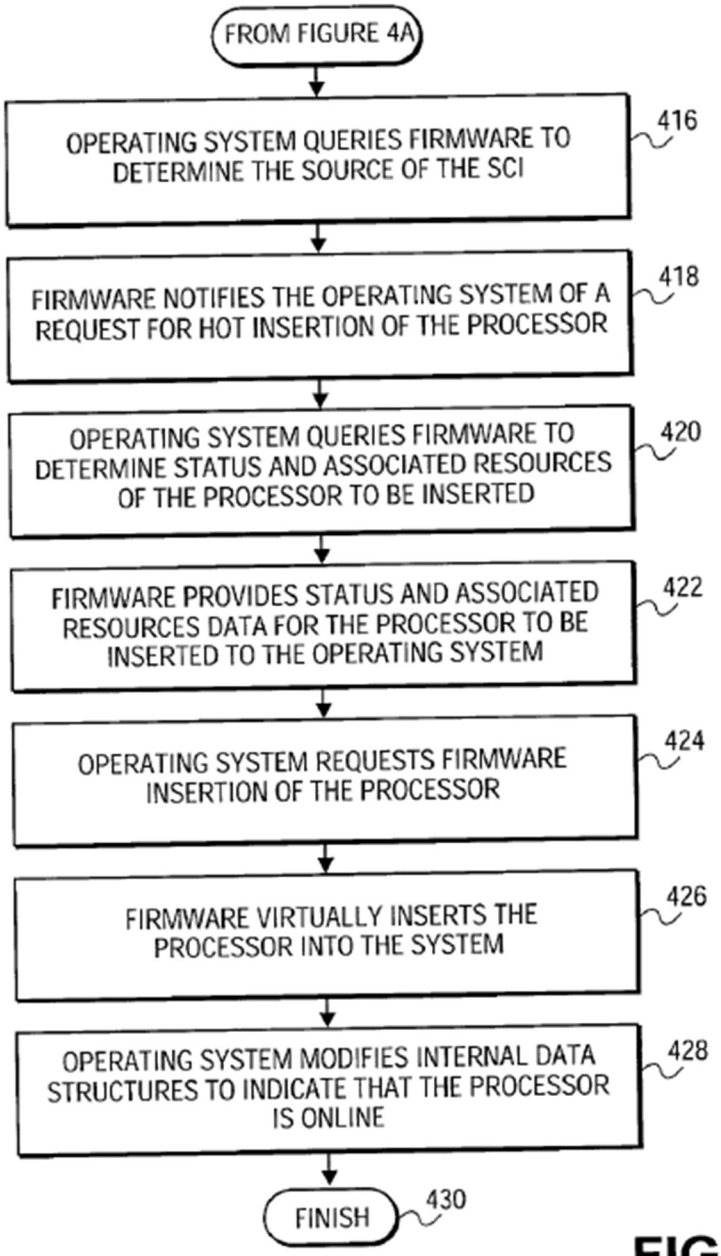


FIG. 4B

Bigbee, FIG. 4B.

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

3. Limitation [1C]: “responsive to said detecting loss of lockstep, using said firmware to determine if lockstep is recoverable for the lockstep pair of processors, wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.”

149. Bigbee-Nguyen renders obvious element [1C].

150. As I explained above for element [1A], Bigbee teaches detecting a loss of lockstep using firmware. Section XII.C.1.

151. Nguyen discloses determining if lockstep is recoverable in response to detection of lockstep errors. *See above* Section VIII.B. Nguyen also discloses that responsive to the detection of an error in the processor cores (“responsive to said detecting loss of lockstep”), the FRC checker and error detector operate together to determine whether lockstep is recoverable from the detected error. Nguyen, [0022]-[0025], [0039]-[0041], [0056]-[0057].

152. Nguyen’s FRC checker and error detector determine whether lockstep is recoverable from the detected error in two instances and non-recoverable from the detected error in one instance. First, Nguyen’s FRC checker and error detector determine that lockstep is recoverable from a soft error detected by the error detector

before detecting an FRC error. Nguyen, [0022], [0046]. Second, Nguyen's FRC checker and error detector determine lockstep is recoverable from a mismatch error if an error in the processors is detected within a timeout interval from the detection of the mismatch error. Nguyen, [0023], [0030]. Third, Nguyen's FRC checker and error detector determine an FRC error is not recoverable. Nguyen, [0007], [0023], [0037]; *see above* Section VIII.B.

153. As explained in Section XII.B.1, a POSITA would have been motivated to incorporate Nguyen's determination of whether lockstep is recoverable into Bigbee's FRC logic to reduce the impact of errors and increase system fault tolerance and availability. A POSITA would have had an expectation of success in implementing Nguyen's determining step in Bigbee's firmware FRC logic for the reasons explained in Section XII.B.2.

154. Accordingly, Bigbee in view of Nguyen discloses "using said firmware to determine if lockstep is recoverable." Bigbee-Nguyen further discloses "using said firmware . . . [for] determining if a lockstep mismatch has occurred between the pair of lockstep processors."

155. Nguyen discloses "determining if a lockstep mismatch has occurred between the lockstep pair of processors" because Nguyen determines if the lockstep is recoverable or non-recoverable after detecting a mismatch error between the pair of lockstep processors. For example, with Nguyen's determining if lockstep is

recoverable or non-recoverable: (1) Nguyen's FRC checker's compare unit compares the data from both cores, and it sets a status flag to indicate whether the data matches or mismatches ("determining if a lockstep mismatch has occurred"); and (2) Nguyen's error detector detects an error in the processors before (lockstep recoverable) or after (lockstep non-recoverable) the timeout interval of timer unit expires ("determine if lockstep is recoverable"). Nguyen, [0056]- [0057]; *see above* Section VIII.B.

156. As explained in Section XII.B.1, a POSITA would have been motivated to incorporate Nguyen's determination of whether lockstep is recoverable, and therefore Nguyen's additional determination if a lockstep mismatch has occurred, in Bigbee's firmware FRC logic to reduce impact of errors and increase system fault tolerance and availability. A POSITA would have had an expectation of success in implementing Nguyen's determining step in Bigbee's firmware FRC logic for the reasons explained in Section XII.B.2.

D. Overview of The Bigbee-Nguyen-Goodrum Combination

157. As I noted above (Section XII.A), claims 9-12 all depend from claim 1 (either directly or indirectly through other claims). As I explained above, the elements of claim 1 are taught by Bigbee-Nguyen. Section XII.C. As I explain below, it is my opinion that the additional elements of claims 9-12 are taught by

Goodrum and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Goodrum. Sections XII.E-H, *infra.*

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

159. To provide a fault-tolerant system, Goodrum describes a hot spare boot circuit that detects a failure of a designated boot processor and reassigns the role of boot processor to a backup processor. Goodrum, Abstract, 1:17-20, 2:15-64, 3:7-31, 5:66-6:3, 6:28-57, 8:11-16, 8:60-64, 9:39-63, 17:7-16, 18:34-49, FIG. 2. Goodrum distinguishes the boot processor from the backup processor using, for example, first and second identifier values, respectively. Goodrum, 8:14-16, 17:7-19. In response to detecting a failure with the initial boot processor, the hot spare circuit disables the boot processor, modifies the identifier value of the backup processor to designate the backup processor as the new boot processor (by changing

the identifier value of the backup processor to the first identifier value that is used to identify or designate the boot processor), and then causes the new boot processor to complete system power on operations. Goodrum, 3:7-31, 9:39-63, 18:34-49.

160. A POSITA would have understood the benefits to the Bigbee-Nguyen system of applying Goodrum's techniques for providing a backup processor for a failed boot processor. In particular, based on the teachings of Goodrum and Bigbee-Nguyen, it is my opinion that a POSITA would have understood the desirability of detecting an error associated with a boot processor and the ability to identify a replacement backup processor both at startup and after startup (e.g., at any time during the operation of the system). While Goodrum focuses on detecting boot processor errors during startup, Bigbee-Nguyen describes continuously monitoring its processors, and idling and restarting a processor for which a loss of lockstep is detected. After startup, if a failed boot processor is not switched to a backup processor, then on a subsequent system reset, the system would attempt to power up using the failed boot processor which Goodrum explains would leave the system incapacitated. Further, a POSITA would have understood that it would be beneficial for Bigbee-Nguyen's processor, for which loss of lockstep was detected and then recovered, to be used as a backup or spare processor for the newly designated boot processor, based on Goodrum's teaching regarding the benefits of maintaining a backup boot processor. The addition of Goodrum to the Bigbee-Nguyen system

does not remove the functionality of Bigbee-Nguyen that reads on claim 1, as described above.

161. Accordingly, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen), to use the original boot processor as a backup processor for the newly identified boot processor (based on the teachings of Goodrum to provide a spare processor). Further, the Bigbee-Nguyen-Goodrum system would replace the faulty original boot processor with the spare processor before attempting to recover from any error (e.g., loss of lockstep) experienced by the faulty original boot processor. A POSITA would have understood that doing so furthers the goal of reducing the amount of time the system is without an identified, operational boot processor as taught by Goodrum (Goodrum, 2:15-31) and because recovering from errors may require a significant amount of time and recovery is not certain. Overall, by doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a reliable boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or

reset the processor for which a loss of lockstep was detected. The addition of Goodrum does not remove the functionality of Bigbee-Nguyen that reads on claim 1, as described above in Section XII.A-C.

162. It is my opinion that a POSITA would have understood that Bigbee-Nguyen's firmware-based processor error handling routines, which already detect a loss of lockstep in the processors, would be modified to determine if a processor error (e.g., loss of lockstep) occurred in the processor identified as the boot processor and, if so, to cause a backup processor take over the role of boot processor and to use the initial boot processor as a backup to the new boot processor once the error experienced by the original boot processor was addressed, based on Goodrum's teachings. A POSITA would have further understood that combining the teachings of Bigbee-Nguyen and Goodrum in this regard is nothing more than applying a known technique (Goodrum's technique for providing a backup processor for a boot processor and for identifying a boot processor and a backup processor, and designating the backup processor as the new boot processor when the originally-designated boot processor has an error detected) to a known device (Bigbee-Nguyen's system having multiple processors in lockstep that can suffer a loss of lockstep fault involving a boot processor) ready for improvement (the ability to designate a spare, non-faulty processor as the boot processor when there is a fault with the boot processor and to use the recovered original boot processor as backup

processor to the new boot processor) to yield a predictable result (Bigbee-Nguyen-Goodrum which, when a loss of lockstep is detected that involves a boot processor, identifies another processor to act as the boot processor and uses the recovered original boot processor as a spare for the new boot processor, yielding better system reliability).

163. A POSITA would have expected success in combining Goodrum with Bigbee-Nguyen because this combination merely involves routine programming skills that a POSITA would have had. As I explained above (Section XII.B), a POSITA would know how to use ASL to program Bigbee-Nguyen's firmware. As acknowledged by Bigbee, control methods for handling processor errors may be written in ASL and ACPI provides the ability to store information regarding the hardware devices (e.g., processors) present in a system. Bigbee, [0016] ("firmware 104 comprises an ACPI BIOS including ACPI machine language (AML) program code describing what hardware devices are present within data processing system 100, as well as their configuration and interfaces, via ACPI control methods, objects, registers, and tables."), [0017] ("Control methods may be written in an ACPI source language (ASL) and compiled into AML for inclusion with the system's BIOS. The operating system 106 executes a control method by retrieving its associated AML code from BIOS and then interpreting the retrieved code utilizing its ACPI device driver/AML interpreter."), [0018]-[0019], [0020]-[0021] (describing a disclosed

processor error handler as comprising a firmware routine), [0022]. A POSITA would therefore use ASL and/or other available ACPI features to program Bigbee-Nguyen's firmware to, for example, (1) associate different identifiers to a boot processor and a backup processor, (2) determine if a processor with an error detected is designated as a boot processor based on the identifier of the processor, and (3) if so, to cause a backup processor to take over the role of boot processor and to use the original boot processor as a spare processor for the new boot processor once the error to the original boot processor is addressed. Thus, a POSITA would have expected success in combining Goodrum with Bigbee-Nguyen as set forth above.

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

164. As I explained above in section XII.C, Bigbee-Nguyen teaches the method of claim 1. Claim 9 further requires the step of determining if the lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor. As I explained above (Section XII.D), determining whether an error occurs in a boot processor is taught by Goodrum. Goodrum teaches a system whereby the primary processor is designated as such, e.g., by having a particular identifier. Goodrum, 3:1-31, 8:14-16, 17:7-19; Sections VIII.E, XII.D. Goodrum also notes that an identifier of CPU0 is typically used to designate the boot processor. Goodrum, 2:39-50. As explained above, Goodrum identifies the boot processor so

that it can switch to a spare processor in the event an error or failure is detected in the boot processor. Goodrum, Abstract, 1:17-20, 2:15-64, 3:7-31, 5:66-6:3, 6:28-57, 8:11-16, 8:60-64, 9:39-63, 17:7-16, 18:34-49. I discussed the motivations for incorporating this teaching into Bigbee-Nguyen above. Section XII.D. For example, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen), to use the original boot processor as a backup processor for the newly identified boot processor (based on the teachings of Goodrum to provide a spare processor). By doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a reliable boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or reset the processor for which a loss of lockstep was detected.

165. Accordingly, in the Bigbee-Nguyen-Goodrum system, when a loss of lockstep is detected for a pair of processors, as taught in Bigbee-Nguyen (Section XII.C), the Bigbee-Nguyen-Goodrum system would also identify whether the pair

includes a processor designated as the boot processor and if so, would then identify a different, standby processor as the new boot processor.

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

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

167. As I explained above (Section XII.E.), claim 9 is taught by Bigbee-Nguyen-Goodrum. Goodrum also teaches the additional limitation of claim 10, and this claim is rendered obvious by Bigbee-Nguyen-Goodrum.

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

169. As I explained above (Section XII.D), Goodrum teaches that this beneficially allows for a more fault-tolerant system that can continue to operate, even when the boot processor suffers a failure, in part because it will not attempt to reboot, if that becomes necessary, from a malfunctioning processor. Goodrum, 2:15-3:13. I also explained the motivations to make this combination above. Section XII.D. For example, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the

error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen), to use the original boot processor as a backup processor for the newly identified boot processor (based on the teachings of Goodrum to provide a spare processor). By doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a reliable boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or reset the processor for which a loss of lockstep was detected.

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

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

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

172. As I explained above (Section XII.F), claim 10 is taught by Bigbee-Nguyen-Goodrum. Bigbee teaches the additional limitation of claim 11, and so this claim is rendered obvious by Bigbee-Nguyen-Goodrum.

173. In particular, as I explained in Section XII.C.2 with respect to element [1B], Bigbee teaches to recover the loss of lockstep. In Bigbee-Nguyen-Goodrum, the system will recover loss of lockstep after having (as taught in Goodrum) swapped the role of boot processor to the spare processor. I explained the motivations for adding Goodrum's teachings to swap the processors above. Section XII.D. For example, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen), to use the original boot processor as a backup processor for the newly identified boot processor (based on the teachings of Goodrum to provide a spare processor). Further, the Bigbee-Nguyen-Goodrum system would replace the faulty original boot processor with the spare processor before attempting to recover from any error (e.g., loss of lockstep)

experienced by the faulty original boot processor as Goddrum (Goodrum, 2:15-31) teaches the benefits of reducing the amount of time the system is without an identified, operational boot processor. By doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a reliable boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or reset the processor for which a loss of lockstep was detected.

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

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

175. As I explained above (Section XII.G), claim 11 is taught by Bigbee-Nguyen-Goodrum. Bigbee-Nguyen-Goodrum also teaches the additional limitation of claim 12, and this claim is rendered obvious by Bigbee-Nguyen-Goodrum.

176. As described above for Bigbee-Nguyen, and which also occurs in Bigbee-Nguyen-Goodrum, when the loss of lockstep is recovered, the processor pair for which loss of lockstep has been recovered is inserted and the operating system modifies its internal data structures to indicate that the processor is "online and available" (claimed "holding the recovered pair of processors in idle"). Section XII.C.2; Bigbee, [0037]. As taught by Bigbee, in the Bigbee-Nguyen-Goodrum

system, insertion of the processor "puts [the] device into its D0 or 'active' state." Bigbee, [0037]. A POSITA would have understood that a processor in the D0 or "active" state is fully on and operational and would have also understood that Bigbee's description of a processor as "available" teaches that the processor is capable and ready to execute instructions. Ex. 1022, p. 35; Bigbee, [0037], FIG. 4B (block 430). The '958 Patent explains that a processor in idle (e.g., a hot spare processor) is similarly available to the system. Ex. 1001, 9:23-27, 10:26-34. Bigbee therefore describes holding a recovered processor in the same manner (i.e., in idle). And, applying the further teachings of Goodrum, the processor pair would be available for use as a spare boot processor because it would be operational, yet would not be identified as the boot processor (claimed "spare for the processor to which the role of system boot processor was swapped"), and so this claim is also rendered obvious by Bigbee-Nguyen-Goodrum.

177. I described the motivations for adding Goodrum's teachings to recognize that the idled processor pair for which lockstep was recovered is available as a spare boot processor above. Section XII.D. For example, a POSITA would have been motivated to modify Bigbee-Nguyen's response to a detected loss of lockstep error to include determining if the error was experienced by a boot processor (based on Goodrum's teachings of using identifiers to distinguish a boot processor from other processors), to reassign the role of boot processor to a backup

processor (as further taught by Goodrum), and once lockstep is recovered (as is still taught by Bigbee and Nguyen), to use the original boot processor as a backup processor for the newly identified boot processor (based on the teachings of Goodrum to provide a spare processor). By doing so, a POSITA would have understood that the reliability and availability of the Bigbee-Nguyen system would be improved, by ensuring that a reliable boot processor is always available to power up the system in response to a system reset and/or to help recover lockstep or reset the processor for which a loss of lockstep was detected. It is my opinion that a POSITA would have recognized that, once recovered, the processor pair which was no longer identified as the boot processor would become the spare processor, because of the reliability benefits of having a spare processor as described above.

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

XIII. GROUND B: CLAIMS 9-12 ARE RENDERED OBVIOUS BY BIGBEE-NGUYEN IN VIEW OF SUFFIN

A. Overview of Bigbee-Nguyen-Suffin Combination

179. As I noted above, claims 9-12 all depend from claim 1 (either directly or indirectly through other claims). Sections XII.A, XII.D. As I explained above, the elements of claim 1 are taught by the combination of Bigbee and Nguyen. Section XII.C. As I explain below, the additional elements of claims 9-12 are taught

by Suffin and/or Bigbee, rendering these claims obvious in light of Bigbee-Nguyen-Suffin. Sections XIII.B-E, *infra*.

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

181. Suffin explains that using this fault-tolerant process beneficially allows for "robust operation of a fault-tolerant computer, including initialization and recovery from operational failures." Suffin, 1:1-6, 2:5-17. Suffin also touts the "superior reliability" caused by having such built-in redundancy, and that the use of a second processor for reboot beneficially allows analysis of the faulty processor,

because its state characteristics can be copied for later analysis, and the state characteristics of the non-faulty processor can be copied to the faulty processor. *Id.,* 1:12-18, 3:16-4:5, 16:20-17:11.

182. A POSITA would have applied Suffin to Bigbee-Nguyen to yield a Bigbee-Nguyen-Suffin system that monitors the processors for loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the Bigbee-Nguyen-Suffin system identifies a spare processor from Suffin's boot list to act as the boot processor. Further, the Bigbee-Nguyen-Suffin system would replace the faulty original boot processor with the spare processor before attempting to recover from any error (e.g., loss of lockstep) experienced by the faulty original boot processor as taught by Suffin (Suffin, Abstract (describing selecting a spare processor before restarting the failed processor), 3:22-4:5, 12:3-18, 13:16-14:6). A POSITA would have understood that doing so furthers the goal of reducing the amount of time the system is without an identified, operational boot processor because recovering from errors may require a significant amount of time and recovery is not certain. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but it is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a

POSITA would have understood the benefits of using the original boot processor that experienced an error as a backup processor to the new boot processor, once the error experienced by the original boot processor was addressed. The addition of Suffin to the Bigbee-Nguyen system does not remove the functionality of Bigbee-Nguyen that reads on claim 1, as described above.

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

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

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

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

186. As I explained above in Section XII.C, Bigbee-Nguyen teaches the method of claim 1. Claim 9 further requires the step of determining if the lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor. As I explain below, determining whether an error, such as a loss of lockstep, occurs in a boot processor is taught by Suffin.

187. In Suffin, the boot list contains values for each processor indicating whether each processor is functioning properly, and the boot list also contains processor identification values to distinguish processors. Suffin, 12:3-18. Thus, Suffin monitors whether a particular processor, such as the first processor on the boot list which was used to initially boot the system (i.e., the boot processor), remains functional, including whether it has suffered a loss of lockstep with another processor. *Id*., 6:9-11, 7:4-16, 8:1-4, 12:3-18, 13:1-7, 14:15-18, 15:7-18. As explained above, Suffin teaches that this beneficially allows for a more robust, reliable system that can better diagnose problems with processors and that will not attempt to reboot the system, if that becomes necessary, from a malfunctioning processor. I explained the motivations to make the Bigbee-Nguyen-Suffin above. Section XIII.A. For example, a POSITA would have applied Suffin to Bigbee-Nguyen to yield a Bigbee-Nguyen-Suffin system that monitors the processors for

loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the Bigbee-Nguyen-Suffin system identifies a spare processor from Suffin's boot list to act as the boot processor. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but it is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a POSITA would have understood the benefits of using the original boot processor that experienced an error as a backup processor to the new boot processor, once the error experienced by the original boot processor was addressed. Further, a POSITA would have been motivated to apply Suffin's teaching to achieve the benefits described in Suffin: including (a) superior reliability from having spare processors identified to be used for rebooting when the boot processor fails and (b) the ability to better diagnose and correct faults with the boot processor. Suffin, 1:1-6, 1:12-18, 2:5-17.

188. Accordingly, in Bigbee-Nguyen-Suffin, when a loss of lockstep is detected, as taught in Bigbee-Nguyen (Section XII.C), the system's boot list would indicate which processors were involved with the loss of lockstep detection, so that a different processor would be used for further rebooting, if needed. In the event the detected loss of lockstep involved the system boot processor, then the Bigbee-

Nguyen-Suffin system would have identified that the boot processor had a loss of lockstep detected, idled it as described above with respect to claim 1, and the processor would be treated as unavailable for use as a boot processor until it was recovered. The role of boot processor would be assigned to a different, functioning processor on the boot list.

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

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

190. As I explained above, claim 9 is taught by Bigbee-Nguyen-Suffin. Section XIII.B. Suffin also teaches the additional limitation of claim 10, and this claim is rendered obvious by Bigbee-Nguyen-Suffin.

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

that will not attempt to reboot the system, if that becomes necessary, from a malfunctioning processor. The motivations to make this combination were described above. Section XIII.A. For example, a POSITA would have applied Suffin to Bigbee-Nguyen to yield a Bigbee-Nguyen-Suffin system that monitors the processors for loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the Bigbee-Nguyen-Suffin system identifies a spare processor from Suffin's boot list to act as the boot processor. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but it is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a POSITA would have understood the benefits of using the original boot processor that experienced an error as a backup processor to the new boot processor, once the error experienced by the original boot processor was addressed. Further, a POSITA would have been motivated to apply Suffin's teaching to achieve the benefits described in Suffin: including (a) superior reliability from having spare processors identified to be used for rebooting when the boot processor fails and (b) the ability to better diagnose and correct faults with the boot processor. Suffin, 1:1-6, 1:12-18, 2:5-17.

192. Accordingly, it is my opinion that in Bigbee-Nguyen-Suffin, when a loss of lockstep is detected and the processors are idled, as taught in Bigbee-Nguyen (Section XII.C) and as still occurs in Bigbee-Nguyen-Suffin, the system's boot list would indicate which processors were involved with the loss of lockstep detection and thus unavailable, so that a different processor on the boot list would be swapped in as the current boot processor and used for rebooting.

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

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

194. As I explained above (Section XIII.C), claim 10 is taught by Bigbee-Nguyen-Suffin. Bigbee also teaches the additional limitation of claim 11, which is simply the recovery of lockstep that is part of element [1B] (Section XII.C.2), and which is taught in Bigbee and still occurs in Bigbee-Nguyen-Suffin, rendering this claim obvious.

195. In particular, as I explained in Section XII.C.2 with respect to element [1B], Bigbee teaches to recover the loss of lockstep after it is detected. In Bigbee-Nguyen-Suffin, the system will recover that loss of lockstep after having (as taught in Suffin) detected the error and swapped the role of boot processor to the spare processor when the previous boot processor was idled and thus determined to be unavailable on the boot list. I described the motivations for adding Suffin's

teachings to remove the faulty, idled processor from the available processors on the boot list above. Section XIII.A. For example, a POSITA would have applied Suffin to Bigbee-Nguyen to yield a Bigbee-Nguyen-Suffin system that monitors the processors for loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the Bigbee-Nguyen-Suffin system identifies a spare processor from Suffin's boot list to act as the boot processor. Further, the Bigbee-Nguyen-Suffin system would replace the faulty original boot processor with the spare processor before attempting to recover from any error (e.g., loss of lockstep) experienced by the faulty original boot processor as taught by Suffin (Suffin, Abstract (describing selecting a spare processor before restarting the failed processor), 3:22-4:5, 12:3-18, 13:16-14:6), which furthers the goal of reducing the amount of time the system is without an identified, operational boot processor. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but it is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a POSITA would have understood the benefits of using the original boot processor that experienced an error as a backup processor to the new boot processor, once the error experienced by the original boot processor was addressed. Further, a POSITA would have been

motivated to apply Suffin's teaching to achieve the benefits described in Suffin: including (a) superior reliability from having spare processors identified to be used for rebooting when the boot processor fails and (b) the ability to better diagnose and correct faults with the boot processor. Suffin, 1:1-6, 1:12-18, 2:5-17

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

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

197. As explained above, Claim 11 is taught by Bigbee-Nguyen-Suffin. Suffin teaches the additional limitation of claim 12, and this claim is rendered obvious by Bigbee-Nguyen-Suffin. As described above for Bigbee-Nguyen, and which also occurs in Bigbee-Nguyen-Suffin, when the loss of lockstep is recovered, the processor pair for which loss of lockstep has been recovered is inserted and the operating system modifies its internal data structures to indicate that the processor pair is online and available (claimed "holding the recovered pair of processors in idle"). Section XII.C.2. As I noted in Section XII.H, Bigbee describes a recovered processor as being available to execute instructions in the same manner that the '958 Patent describes holding a processor (e.g., a hot standby processor) as available to a system. Ex. 1001, 9:23-27, 10:26-34. Applying the teachings of Suffin (Suffin, 16:20-17:11, 13:1-7), the processor pair would then be available for use as a spare

boot processor on the boot list (claimed "spare for the processor to which the role of system boot processor was swapped"), and so this claim is also rendered obvious by Bigbee-Nguyen-Suffin.

198. I described the motivations for adding Suffin's teachings to recognize that the idled processor pair for which lockstep was recovered is available as a spare boot processor above. Section XIII.A. For example, a POSITA would have applied Suffin to Bigbee-Nguyen to yield a Bigbee-Nguyen-Suffin system that monitors the processors for loss of lockstep (as taught by Bigbee-Nguyen) and determines whether the faulty processor having a loss of lockstep is the boot processor (as taught by Suffin). If it is, then the Bigbee-Nguyen-Suffin system identifies a spare processor from Suffin's boot list to act as the boot processor. If the loss of lockstep is recovered, as is still taught by Bigbee-Nguyen, then the recovered processor is once again available for being used as a boot processor on Suffin's boot list, but it is now a backup processor. As Suffin teaches the benefits of always having a backup processor to the boot processor, a POSITA would have understood the benefits of using the original boot processor that experienced an error as a backup processor to the new boot processor, once the error experienced by the original boot processor was addressed. Further, a POSITA would have been motivated to apply Suffin's teaching to achieve the benefits described in Suffin: including (a) superior reliability from having spare processors identified to be used for rebooting when the boot

processor fails and (b) the ability to better diagnose and correct faults with the boot processor. Suffin, 1:1-6, 1:12-18, 2:5-17. It is my opinion that a POSITA would have recognized that, once recovered, the processor pair which was no longer identified as the boot processor would become the spare processor, because of the reliability benefits of having a spare processor described above.

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

XIV. GROUND C: CLAIMS 23-25 AND 12 ARE RENDERED OBVIOUS BY BIGBEE-NGUYEN-SUFFIN-SAFFORD

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

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

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

202. A POSITA would have applied Safford to Bigbee-Nguyen-Suffin to yield a Bigbee-Nguyen-Suffin-Safford system that, pursuant to the teachings of

Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies the processor's state to the hot spare processor. The addition of Safford to the system does not remove the functionality of Bigbee-Nguyen-Suffin that reads on claims 9-12, as described above (Section XIII).

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

204. It also is an example of applying a known technique (Safford's techniques described above) to improve similar devices (Bigbee-Nguyen-Suffin's system which, like the system of Safford, has multiple processors operating in

lockstep that are subject to faults, and spare processors that become active in the event a processor has a loss of lockstep detected) in the same way (when a loss of lockstep is detected that involves a boot processor, identifying another processor to act as the boot processor and copying the state of the first processor to the second, yielding enhanced recovery from the loss of lockstep).

205. A POSITA would have expected success in combining Safford with Bigbee-Nguyen-Suffin because this combination merely involves routine programming skills that a POSITA would have had. As I noted above (Sections XII.B.2, XII.D, XIII.A), a POSITA would know how to use ASL to program Bigbee-Nguyen-Suffin's firmware to modify its processor error handling responses. Bigbee explains that these processor error responses include saving processor state information and ensuring that the state of a processor for which an error is detected is reinitialized to enable reintroduction. Bigbee, [0021] (describing a firmware routine that "saves minimal processor state" and "saves additional processor and platform error and state information"), [0023] (using firmware "to ensure that [the processor's package] is reset or 're-initialized' to the state needed for it to be subsequently awakened or 'activated' by operating system 106."), [0024]. It is my opinion that a POSITA would therefore use ASL or other ACPI features to program Bigbee-Nguyen-Suffin's firmware to copy to a hot spare processor the saved state of a processor for which a loss of lockstep has been detected, based on the

capabilities of firmware to do so as taught by Bigbee. Thus, a POSITA would have expected success in combining Safford with Bigbee-Nguyen-Suffin.

B. Independent Claim 23

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

206. Limitation [23A] is rendered obvious by Bigbee-Nguyen-Suffin. Bigbee-Nguyen-Suffin possesses the functionality of Bigbee-Nguyen, described above in Sections XII.A-C, and also adds the teachings of Suffin, as described in Section XIII. In the Bigbee-Nguyen-Suffin system there are multiple processors, and, per the teachings of Suffin (Suffin, 2:5-4:5, 11:3-8, 11:23-12:18, 13:1-7, 13:16-14:10, FIG. 4A), the Bigbee-Nguyen-Suffin system has a boot list that contains a listing of the current boot processor and other available (functioning) processors, any of which is a claimed "hot spare" processor for the system boot processor, because each such processor can be designated as the boot processor if the boot processor becomes unavailable (i.e., malfunctions).

207. Safford also teaches limitation [23A]], and this claim is rendered obvious by Bigbee-Nguyen-Suffin-Safford. Specifically, Safford teaches, when a loss of lockstep is detected in a multi-processor system, "moving an architected state of the processor unit experiencing the loss of lock step to a spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer

system." Safford, [0007], [0018]-[0021] (discussion of using "hot standby" processors), FIGs. 3-4 (copying architected state of processor to "hot standby" processor, designating new processor as hot standby processor), Abstract. I described the benefits and motivations for having such a spare processor above and apply equally to Safford's teaching of this feature. Sections XIII.A, XIII.E, XIV.A.

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

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

209. As I described with respect to limitation [1A], this step is taught by Bigbee and is performed in the Bigbee-Nguyen-Suffin-Safford system. Section XII.C.1.

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

210. As I described with respect to claim 9, this step is taught by Suffin and is performed in the Bigbee-Nguyen-Suffin-Safford system. Section XIII.B.

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

211. Bigbee-Nguyen-Suffin-Safford teaches this step.

212. As I described above (Sections VIII.F, XIV.A), Safford, like Bigbee-Nguyen-Suffin, relates to a multi-processor system with processors operating in lockstep. Safford, Abstract, [0001], [0006]-[0007], [0018]-[0021], Figs. 2-4. In

order to "enhance recovery from loss of lock step," Safford copies the state of one or more of the processors in a processor pair for which a loss of lockstep has been detected to a hot spare processing unit, which becomes an active processor unit in the computer system. *Id*. This process is shown, for instance, in Figure 4, step 220:

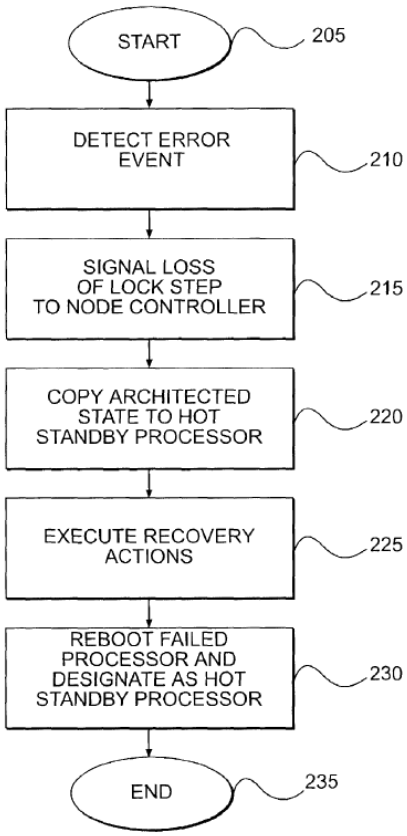


FIG. 4

Safford, FIG. 4, [0021].

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

213. Accordingly, in Bigbee-Nguyen-Suffin-Safford, when the processor for which a loss of lockstep has been detected is the boot processor, the system will copy a state of the system boot processor to the hot spare processor. I discussed the motivations for adding this teaching of Safford in the introduction to this ground. Section XIV.A. For example, Safford explicitly provides a motivation to combine, as it explains that a benefit of copying the state of the processor for which a loss of lockstep has been detected is to "enhance recovery from loss of lock step." Safford, [0006], [0021], Abstract. A POSITA would have applied Safford to Bigbee-Nguyen-Suffin to yield a Bigbee-Nguyen-Suffin-Safford system that, pursuant to the teachings of Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies the processor's state to the hot spare processor. A POSITA would have been motivated to apply Safford's teaching to achieve the benefits described in Safford: enhanced, quicker recovery from a loss of lockstep. Safford, Abstract, [0005]-[0007], [0021]. Bigbee-Nguyen-Suffin-Safford therefore teaches this limitation.

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

214. Bigbee-Nguyen-Suffin-Safford teaches this step.

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

216. Accordingly, Bigbee-Nguyen-Suffin-Safford teaches claim 23.

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

217. Bigbee-Nguyen-Suffin-Safford teaches this step.

218. As I explained above, claim 23 is taught by Bigbee-Nguyen-Suffin-Safford. Section XIV.B. As I explained with respect to claim 11, Bigbee-Nguyen-Suffin teach that loss of lockstep for the processor pair is recovered after swapping the role of boot processor to the spare processor Section XIII.D. This functionality remains in the Bigbee-Nguyen-Suffin-Safford combination. Further, Safford explicitly teaches in Figure 4 that the copying occurs before attempting to recover the loss of lockstep, wherein step 225 (attempting to recover lockstep) occurs after step 220 (the copying of the processor to the hot spare):

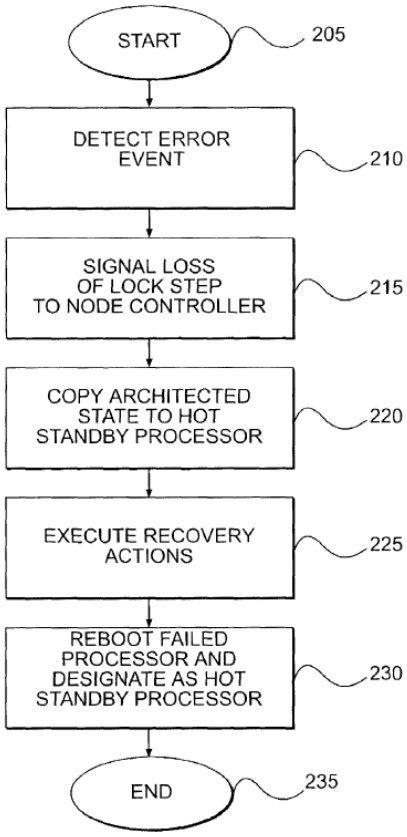


FIG. 4

Safford, FIG. 4, [0006]-[0007], [0021]; Section XIV.A. Therefore, in the combined Bigbee-Nguyen-Suffin-Safford system, the attempted recovery of the loss of lockstep occurs after copying the state of the system boot processor to the hot spare processor. I explained the motivations for applying Safford to Bigbee-Nguyen-Suffin above in Section XIV.A. For example, Safford explicitly provides a motivation to combine, as it explains that a benefit of copying the state of the processor for which a loss of lockstep has been detected is to "enhance recovery from loss of lock step." Safford, [0006], [0021], Abstract. A POSITA would have applied Safford to Bigbee-Nguyen-Suffin to yield a Bigbee-Nguyen-Suffin-Safford

system that, pursuant to the teachings of Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies the processor's state to the hot spare processor. A POSITA would have been motivated to apply Safford's teaching to achieve the benefits described in Safford: enhanced, quicker recovery from a loss of lockstep. Safford, Abstract, [0005]-[0007], [0021].

219. Accordingly, Bigbee-Nguyen-Suffin-Safford teaches claim 24.

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

220. Bigbee-Nguyen-Suffin-Safford teaches this step.

221. As I explained above (Section XIV.C), claim 24 is taught by Bigbee-Nguyen-Suffin-Safford. As I explained with respect to claim 12, Bigbee-Nguyen-Suffin teaches that, when the loss of lockstep is recovered in the boot processor pair, the boot processor pair becomes available as a spare processor. Section XIII.E. This functionality remains in the Bigbee-Nguyen-Suffin-Safford combination, which renders claim 25 obvious.

222. Further, Safford also teaches that, once loss of lockstep is recovered in the processor pair that was idled and copied, that processor pair is designated as the hot standby processor. Safford, FIG, 4, [0021] (discussing that the recovered processors "become the new 'hot standby' processor pair"). As I noted above

(Sections XIII.A, XIV.A), a POSITA would have recognized that, once recovered, the processor which was no longer identified as the boot processor would become a spare processor, because of the reliability benefits (also taught in Safford) of having a spare processor. I described the motivations for applying Safford to Bigbee-Nguyen-Suffin above (Section XIV.A) and the motivations for having a hot spare processor described with respect to the teachings of Suffin and Goodrum (to provide a more reliable and fault-tolerant system) also apply to those teachings in Safford. Moreover, Safford explicitly provides a motivation to combine, as it explains that its system beneficially can "enhance recovery from loss of lock step." Safford, [0006], [0021], Abstract. A POSITA would have applied Safford to Bigbee-Nguyen-Suffin to yield a Bigbee-Nguyen-Suffin-Safford system that, pursuant to the teachings of Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies the processor's state to the hot spare processor and uses the recovered processor as a new spare processor (as taught by Suffin). A POSITA would have been motivated to apply Safford's teaching to achieve the benefits described in Safford: enhanced, quicker recovery from a loss of lockstep. Safford, Abstract, [0005]-[0007], [0021]. And, as I note above, having the recovered processor available as a hot spare to the new boot processor increases the robustness of the system. Accordingly, for this reason also, Bigbee-Nguyen-Suffin-Safford renders claim 25 obvious.

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

223. As I explained above, claim 11 is taught by Bigbee-Nguyen-Suffin. Section XIII.D. As also explained in Section XIV.D, immediately above, Safford also teaches that, after lockstep is recovered for a pair of processors, the recovered pair of processors become the new hot standby processors, which are held in idle until needed. Safford, [0021], [0006]-[0007], Abstract, FIG. 4; Ex. 1001, 6:11-15, 9:23-27, 10:26-34. I described the motivations for applying Safford to Bigbee-Nguyen-Suffin above. Section XIV.A. For example, Safford explicitly provides a motivation to combine, as it explains that a benefit of copying the state of the processor for which a loss of lockstep has been detected is to "enhance recovery from loss of lock step." Safford, [0006], [0021], Abstract. A POSITA would have applied Safford to Bigbee-Nguyen-Suffin to yield a Bigbee-Nguyen-Suffin-Safford system that, pursuant to the teachings of Safford, upon detecting a loss of lockstep for a processor (including when the processor is the boot processor), copies the processor's state to the hot spare processor. A POSITA would have been motivated to apply Safford's teaching to achieve the benefits described in Safford: enhanced, quicker recovery from a loss of lockstep. Safford, Abstract, [0005]-[0007], [0021]. Thus, Bigbee-Nguyen-Suffin-Safford renders claim 12 obvious.

XV. GROUND D: CLAIMS 23-25 AND 12 ARE RENDERED OBVIOUS BY BIGBEE-NGUYEN-GOODRUM-SAFFORD

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

A. Independent Claim 23

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

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

226. As described in Section XIV.B.1, Safford also teaches limitation [23A], and it is my opinion that a POSITA would be motivated to add this teaching to

Bigbee-Nguyen-Goodrum for the same reasons as described for combining it with Bigbee-Nguyen-Suffin. Sections XII.D, XIV.A, XIV.B.1.

227. Accordingly, Bigbee-Nguyen-Goodrum-Safford teaches the method of limitation [23A].

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

228. As described with respect to limitation [1A], this step is taught by Bigbee and is performed in the Bigbee-Nguyen-Goodrum-Safford system. Section XII.C.1.

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

229. As described with respect to claim 9, this step is taught by Goodrum and is performed in the Bigbee-Nguyen-Goodrum-Safford system. Section XII.E.

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

230. Bigbee-Nguyen-Goodrum-Safford teaches this step.

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

232. Accordingly, in Bigbee-Nguyen-Goodrum-Safford, when the processor for which a loss of lockstep has been detected is the boot processor, the system will copy a state of the system boot processor to the hot spare processor. I discussed the motivations for adding this teaching of Safford in the introduction to this ground (Section XV) and in Section XIV.A. Bigbee-Nguyen-Goodrum-Safford thus teaches this limitation.

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

233. Bigbee-Nguyen-Goodrum-Safford teaches this step.

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

235. Accordingly, Bigbee-Nguyen-Goodrum-Safford teaches claim 23.

B. Dependent Claim 24: "The method of claim 23 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising: attempting to recover lockstep between the lockstep pair of

processors after copying the state of the system boot processor to the hot spare processor."

236. Bigbee-Nguyen-Suffin-Safford teaches this step.

237. As I explained above (Section XV.A), claim 23 is taught by Bigbee-Nguyen-Goodrum-Safford. As I explained with respect to claim 11, Bigbee-Nguyen-Goodrum teaches that loss of lockstep for the processor pair is recovered after swapping the role of boot processor to the spare processor Section XII.G. This functionality remains in the Bigbee-Nguyen-Goodrum-Safford combination. Further, Safford explicitly teaches in Figure 4 that the copying occurs before attempting to recover the loss of lockstep, wherein step 225 (attempting to recover lockstep) occurs after step 220 (the copying of the processor to the hot spare):

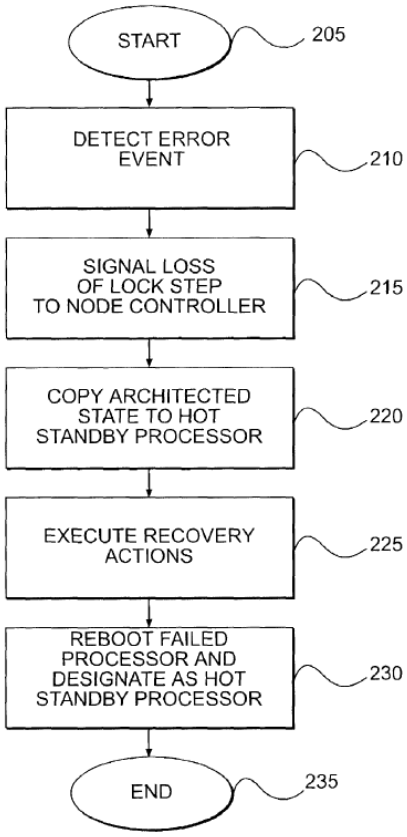


FIG. 4

Safford, FIG. 4, [0006]-[0007], [0021]; Section XV.A. Therefore, in the combined Bigbee-Nguyen-Goodrum-Safford system, the attempted recovery of the loss of lockstep occurs after copying the state of the system boot processor to the hot spare processor. I described the motivations for applying Safford to Bigbee-Nguyen-Goodrum in the introduction to this ground (Section XV) and in Section XIV.A.

238. Accordingly, Bigbee-Nguyen-Goodrum-Safford renders claim 24 obvious.

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

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

239. Bigbee-Nguyen-Goodrum-Safford teaches this step.

240. As I explained above (Section XV.B), claim 24 is taught by Bigbee-Nguyen-Goodrum-Safford. As explained with respect to claim 12, Bigbee-Nguyen-Goodrum teaches that, when the loss of lockstep is recovered in the boot processor pair, the boot processor pair becomes available as a spare processor. Section XII.H. This functionality remains in the Bigbee-Nguyen-Goodrum-Safford combination, and therefore renders claim 25 obvious.

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

D. Dependent Claim 12: "The method of claim 11 further comprising: after recovering lockstep for the lockstep pair of processors, holding the recovered pair of processors in idle as a

spare for the processor to which the role of system boot processor was swapped."

242. As explained above (Section XII.G), claim 11 is taught by Bigbee-Nguyen-Goodrum. As explained in Section XV.C, immediately above, Safford also teaches that, after lockstep is recovered for a pair of processors, the recovered pair of processors become the new hot standby processors, which are held in idle until needed. Safford, [0021], [0006]-[0007], Abstract, FIG. 4; Ex. 1001, 6:11-15, 9:23-27, 10:26-34. The motivations for applying Safford to Bigbee-Nguyen-Goodrum were described by me above in Section XV, introduction. Thus, Bigbee-Nguyen-Goodrum-Safford renders claim 12 obvious.

XVI. CONCLUDING STATEMENT

243. In conclusion, as set forth above, claims 9-12 and 23-25 of the '958 Patent are unpatentable as rendered obvious by the aforementioned prior art references.

244. Prior to working on this declaration, I had never heard of the inventors on the '958 Patent, nor had I heard of the '958 Patent. I have never heard anyone offer praise for the '958 Patent, nor am I aware of any commercial success attributable to the '958 Patent.

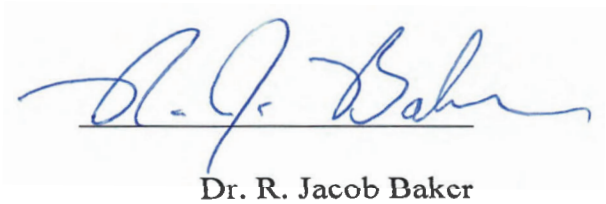
245. I am unaware of any copying of the alleged invention of the '958 Patent. Thus, I am unaware of any secondary considerations supporting the nonobviousness of the '958 Patent.

246. Under penalty of perjury, all statements made herein of my own knowledge are true, and I believe that all statements made herein on information and belief are true, it being understood that my final draft of this declaration is being forwarded on my behalf via Counsel. I have been warned and I am aware that willful false statements are punishable by fine or imprisonment or both under Section 1001 of Title 18 of the United States Code.

247. In signing this Declaration, I understand that the Declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in the case and that such cross-examination will take place in the United States. If cross examination is required of me, I undertake to appear within the United States during the time allotted for cross-examination, and within the limits of my ability so to do.

I declare under penalty of perjury that the foregoing is true and correct.

Executed on June 24th, 2024 in Las Vegas, Nevada.



Dr. R. Jacob Baker

APPENDIX A

List of Materials Considered

Ex. 1001:	U.S. Patent No. 7,502,958 to Michaelis et al. ("the '958 Patent")
Ex. 1004:	Prosecution History of the '958 Patent
Ex. 1005:	U.S. Patent Application Publication No. US 2003/0126498 A1 to Bigbee et al., filed January 2, 2002, published July 3, 2003 ("Bigbee")
Ex. 1006:	U.S. Patent Application Publication No. US 2004/0123201 Al to Nguyen et al., filed December 19, 2002, published June 24, 2004 ("Nguyen")
Ex. 1007:	U.S. Patent Application Publication No. US 2004/0006722 Al to Safford, filed July 3, 2002, published January 8, 2004 ("Safford")
Ex. 1008:	U.S. Patent No. 7,085,959 B2 to Safford, filed July 3, 2002, issued August 1, 2006 ("Safford-959")
Ex. 1009:	U.S. Patent No. 7,296,181 B2 to Safford, filed April 6, 2004, issued November 13, 2007 ("Safford-181")
Ex. 1010:	U.S. Patent Application Publication No. US 2002/0144177 Al to Kondo et al., filed January 31, 2002, published October 3, 2002 ("Kondo")
Ex. 1011:	U.S. Patent No. 6,516,429 B1 to Bossen, filed November 4, 1999, issued February 4, 2003 ("Bossen")
Ex. 1012:	U.S. Patent No. 6,158,015 to Klein, filed March 30, 1998, issued December 5, 2000 ("Klein")
Ex. 1013:	U.S. Patent Application Publication No. US 2005/0172164 A1 to Fox et al., filed January 21, 2004, published August 4, 2005 ("Fox")
Ex. 1014:	U.S. Patent No. 7,404,105 B2 to Arai, filed August 16, 2004, issued July 22, 2008 ("Arai")

Ex. 1015:	U.S. Patent No. 6,675,324 B2 to Marisetty et al., filed September 27, 1999, issued January 6, 2004 ("Marisetty")
Ex. 1016:	U.S. Patent Application Publication No. US 2003/0074601 A1 to Schultz et al., filed September 28, 2001, published April 17, 2003 ("Schultz")
Ex. 1017:	U.S. Patent Application Publication No. 2003/0126142 A1 to Tu et al., filed December 31, 2001, published July 3, 2003 ("Tu")
Ex. 1018:	U.S. Patent No. 5,915,082 to Marshall et al., filed June 7, 1996, issued June 22, 1999 ("Marshall")
Ex. 1020:	PCT Patent Application No. WO 01/80007 A2 to Suffin, filed April 12, 2001, published October 25, 2001 ("Suffin")
Ex. 1021:	U.S. Patent No. 5,627,962 to Goodrum et al., filed December 30, 1994, issued May 6, 1997 ("Goodrum")
Ex. 1022:	Advanced Configuration and Power Interface Specification, Revision 2.0, July 27, 2000 ("ACPI 2.0")
Ex. 1030:	U.S. Patent No. 5,790,850 to Natu, filed September 30, 1996, issued August 4, 1998 ("Natu")

APPENDIX B

Claim Listing Appendix

Designation	Claim Language
Claim 1	
[1A]	1. A method comprising: detecting loss of lockstep for a lockstep pair of processors;
[1B]	using firmware to trigger an operating system to idle the lockstep pair of processors, recover lockstep for the lockstep pair of processors, and trigger the operating system to recognize the lockstep pair of processors having recovered lockstep; and
[1C]	responsive to said detecting loss of lockstep, using said firmware to determine if lockstep is recoverable for the lockstep pair of processors, wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.
Claim 9	
[9A]	9. The method of claim 1 further comprising:
[9B]	determining if said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor.
Claim 10	
[10A]	10. The method of claim 9 wherein if determined that said lockstep pair of processors for which said loss of lockstep is detected comprises a system boot processor, further comprising:
[10B]	swapping the role of system boot processor to another processor in the system.
Claim 11	
[11A]	11. The method of claim 10 further comprising:
[11B]	after said swapping, recovering the lockstep for the lockstep pair of processors.
Claim 12	
[12A]	12. The method of claim 11 further comprising:
[12B]	after recovering lockstep for the lockstep pair of processors, holding the recovered pair of processors in idle as a spare for the processor to which the role of system boot processor was swapped.

Designation	Claim Language
Claim 23	
[23A]	23. A method comprising: establishing, in a multi-processor system, a hot spare processor for a system boot processor;
[23B]	detecting loss of lockstep for a lockstep pair of processors in said multi-processor system;
[23C]	determining if said lockstep pair of processors for which the loss of lockstep is detected is the system boot processor;
[23D]	if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor, copying a state of the system boot processor to the hot spare processor; and
[23E]	if determined that said lockstep pair of processors for which the loss of lockstep is not the system boot processor, then triggering an operating system to a) idle the lockstep pair of processors, b) attempt to recover lockstep between the lockstep pair of processors, and c) if lockstep is successfully recovered between the lockstep pair of processors, trigger said operating system to recognize the processors as being available for receiving instructions.
Claim 24	
[24A]	24. The method of claim 23 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising:
[24B]	attempting to recover lockstep between the lockstep pair of processors after copying the state of the system boot processor to the hot spare processor.
Claim 25	
[25A]	25. The method of claim 24 wherein if determined that said lockstep pair of processors for which the loss of lockstep is the system boot processor further comprising:
[25B]	if lockstep is successfully recovered between the lockstep pair of processors, establishing the lockstep pair of processors for which lockstep was recovered as a hot spare processor.