UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

BMW of North America, LLC and Robert Bosch LLC,

Petitioners,

v.

Foras Technologies Ltd.,

Patent Owner.

IPR2023-01373
U.S. Patent 7,502,958

DECLARATION OF DR. MICHAEL C. BROGIOLI

TABLE OF CONTENTS

I. Introduction....1
II. Qualifications....2
III. Materials Considered....7
IV. Relevant Legal Standards....8
A. Validity....10
B. Level of Ordinary Skill....12
C. Claim Construction....13
1. "Detection of Loss of Lockstep" and "Detecting Loss of Lockstep"....14
2. "Lockstep Is Recoverable" and "Loss of Lockstep Is Recoverable"....17
3. All Other Claim Terms....17
V. The State of Art....17
A. Lockstep Processors or FRC Processors Were Known....17
B. Determining Whether Lockstep Is Recoverable Was Known....20
C. Handling Recoverable or Non-Recoverable Errors Differently Was Known....23
D. Using Firmware to Detect and Handle Processor Errors Was Known....26
VI. Overview of the '958 Patent....27
VII. Overview of the Prior Art....30
A. Overview of Bigbee (Ex. 1005)....30
B. Overview of Nguyen (Ex. 1006)....32

C. Bigbee and Nguyen are Analogous Art to the '958 Patent 36

VIII. Ground of Unpatentability 37

IX. Ground 1: Claims 1-8 and 13-22 Are Obvious over Bigbee in view of Nguyen 38

A. Bigbee in view of Nguyen 38

B. A Person of Ordinary Skill in the Art Would Have Found it Obvious to Combine Bigbee and Nguyen 41

1. Motivation - A person of skill in the art would have been motivated to add Nguyen's determining whether lockstep is recoverable from the detected error to fill a gap in Bigbee's disclosure. 45

2. Reasonable Expectation of Success - A person of skill in the art would have understood how to use Bigbee's disclosed firmware to perform Nguyen's determining whether a detected error is recoverable or non-recoverable. 48

C. Independent Claim 19 51

1. [19preamble]: "A system comprising a pair of lockstep processors and computer-executable firmware code stored to a computer-readable medium, the computer-executable firmware code comprising:" 51

2. [19a]: "firmware code, responsive to detection of loss of lockstep between the pair of lockstep processors, for determining if the lockstep is recoverable for the pair of lockstep processors;" 53

3. [19b]: "wherein said firmware code for determining if the lockstep is recoverable further comprises firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors;" 56

4. [19c]: "firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating

system to idle the processors, attempting to recover lockstep between the pair of processors, and” 57

5. [19d]: “if lockstep is successfully recovered between the pair of processors, [firmware] triggering said operating system to recognize the processors as being available for receiving instructions.” 60

D. Independent Claim 1 63

E. Independent Claim 13 64

1. [13preamble]: “A system comprising:” 64

2. [13a]: “an Advanced Configuration and Power Interface (ACPI)-compatible operating system;” 65

3. [13b]: “master processor operable to process instructions scheduled by said ACPI-compatible operating system;” 66

[13c]: “slave processor lockstepped with the master processor operable to perform redundant processing of said instructions scheduled for the master processor; and” 66

4. [13d]: “firmware operable, responsive to detection of loss of lockstep between the master and slave processors, to use an ACPI method to trigger the operating system to idle the master processor, attempt to recover lockstep between the master and slave processors, and” 71

5. [13e]: “[firmware operable to] if recovery of lockstep is successful then use an ACPI method to reintroduce the master processor to the operating system.” 74

F. Claim 2: “wherein said using firmware to trigger an operating system to idle the lockstep pair of processors further comprises: using the firmware to trigger the operating system to idle a master processor of the lockstep pair of processors.” 76

G. Claim 3: “wherein said using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep further comprises: using the firmware to

trigger the operating system to recognize the master of the lockstep pair of processors." 76

H. Claim 4: "wherein said detecting loss of lockstep comprises: detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon."..... 77

I. Claim 5: "wherein said detecting loss of lockstep comprises: detecting a precursor to lockstep mismatch." 79

J. Claim 6: "wherein said detecting loss of lockstep comprises: detecting lockstep mismatch between the lockstep pair of processors." 80

K. Claim 7: "if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable." 80

L. Claim 8: "if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable."..... 81

M. Claim 14: "error detection logic for detecting loss of lockstep between the master and slave processors." 82

N. Claim 15: "error detection logic for detecting a precursor to loss of lockstep."..... 83

O. Claim 16: "wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep." 83

P. Claim 17: "wherein said error detection logic comprises a parity-based mechanism." 84

Q. Claim 18: "further comprising error detection logic for detecting lockstep mismatch between the master and slave processors." 84

R. Claim 20: "wherein the code for attempting to recover lockstep further comprises: code for resetting the pair of processors. 85

S. Claim 21 86

1. [21a]: “wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system,”.....87

2. [22b]: “said code for triggering the operating system to idle the processors comprises: code for using an ACPI method to idle the processors.”.....87

T. Claim 22.....87

1. [22a]: “wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system,”.....87

2. [22b]: “said code for triggering said operating system to recognize the processors as being available for receiving instructions comprises: code for using an ACPI method to cause the operating system to check for the processors.”.....88

X. Conclusion.....89

I. Introduction

1. I, Michael C. Brogioli, submit this declaration to state my opinions on the matters described below.

2. I have been retained on behalf of Petitioners BMW of North America, LLC ("BMW") and Robert Bosch LLC ("Bosch") as an independent expert for the above-identified *inter partes* review proceeding involving U.S. Patent No. 7,502,958 ("the '958 patent") (Ex. 1001). Although I am being compensated for my time in connection with this IPR at my standard hourly consulting rate and reimbursed for reasonable out-of-pocket expenses, no part of my compensation depends on the outcome of this proceeding, and I have no other interest in this proceeding.

3. I have been asked to provide my technical review, analysis, insights, and opinions regarding the '958 Patent, the prior art references that form the basis for the invalidity ground set forth in the Petition for *Inter Partes* Review of the '958 Patent, and any other prior art publications cited in this declaration.

II. Qualifications

4. I believe that I am well qualified to serve as a technical expert in this matter based upon my educational and work experience summarized below. My *curriculum vitae* (Ex. 1003) records my education, experience, and publications in greater detail.

5. I am currently an Adjunct Professor of Electrical and Computer engineering at Rice University in Houston, Texas, and Managing Director of Polymathic Consulting in Austin, Texas. I received my Bachelor of Electrical Engineering from Rensselaer Polytechnic Institute in 1999, my Master of Science in Electrical and Computer Engineering from Rice University in 2003, and my Doctorate of Electrical and Computer Engineering from Rice University in 2007.

6. While at Rice University, I developed various computer architecture designs for embedded and high-performance systems, including various microprocessor technologies. For example, from 1999 to 2003, I worked in the area of low-power computing, specifically focusing on dynamic power management and performance of configurable memory systems and related microprocessor technology. From 2002 to 2004, I developed Spinach, a computer architecture design and modeling toolset, which models system components common to all programmable computing environments, including memory

systems, microprocessors, hardware accelerators, interconnects, etc. From 2004 to 2009, I developed Spinach DSP-FPGA, a modular and composable simulator design infrastructure for programmable and reconfigurable embedded SOC architectures specifically targeting mobile, low-power, and embedded and portable computing devices. From 2005 to 2009, I developed and published a retargetable compiler infrastructure and hardware design exploration toolkit for systems related to embedded computing technologies, which is also used in the design space exploration of memory systems and related technologies. Many of these tools have been used at United States universities in the area of electrical and computer engineering research.

7. In 1997-1998, I was a Hardware Test Engineer at Stratus Computer in Massachusetts, debugging locked step CPU operation and memory management issues in Stratus' fault tolerant UNIX systems and Hewlett Packard PA–8000 series CPU modules.

8. In 1999, I was a hardware, firmware, and software developer at Vicarious Visions in New York, developing third-party titles for Nintendo's handheld consoles, in addition to various hardware and firmware interfaces and software optimizations related to the limited memory I/O and size.

9. In 2000, I was a Compiler Engineer in the Microprocessor Research Labs of Intel Corporation in California, implementing speculative multi–threading support in Intel's IA–64 compiler specifically for Itanium® family processors, and developing new program analysis and back end code generation phases to support speculatively launching threads at runtime.

10. In 2000-2003, I was an Instructor at the Department of Electrical and Computer Engineering Rice University in Texas, teaching graduate level courses relating to Computer Architecture and Circuit Design, including topics related to fault tolerant computer architecture design.

11. In 2004, I co-authored "Spinach: A Liberty–Based Simulator For Programmable Network Interface Architectures," published in the ACM SIGPLAN/SIGBED Conference on Languages Compilers and Tools for Embedded Systems, Washington, D.C., 2004. Spinach was a tool for multiprocessor system architecture design and was made available by open source in 2004.

12. From June 2006 to August 2007, I worked as the Technical Co-Founder of Method Seven LLC in Boston, MA, working with high-performance software and hardware systems architecture.

13. From 2008 to 2009, I was Senior Engineer working in high-performance compiler designs and next-generation microprocessors and architectures at Freescale Semiconductor in Austin, TX.

14. From November 2009 to October 2011, I was Chief Architect, Senior Member Technical Staff, at Freescale Semiconductor in Austin, TX (formerly Motorola), responsible for management of technology, engineering roadmaps, design lead on software infrastructure, and next-generation microprocessor architectures for embedded computing. During my tenure at Freescale Semiconductor, I was in charge of system developer tools for processor design, both in the hardware, firmware, and software spaces.

15. I have held the position of Adjunct Professor at Rice University since 2009 and the position of Managing Director at Polymathic Consulting since 2011. At Rice University, I instruct graduate-level curriculum in the areas of computer architecture, hardware and software systems, with a focus on computer architecture. I also advise on university research and various design initiatives. At Polymathic Consulting, I work with a range of technologists from early-stage start-ups to Fortune 500 companies on similar technologies.

16. I am recognized as an expert in the field of computer architecture, computer hardware, computer firmware, and computer software systems as they

relate to the subject matter at hand. I am a member of the Institute of Electrical and Electronics Engineers (IEEE) and have been a Program Committee member for the IEEE and ACM Design Automation Conference since 2011, and have held the role of Program Chair of Design Automation Conference in the area of Embedded Computing.

17. Over the past 15 plus years, I have authored numerous peer-reviewed publications, as well as engineering books in the area of computer hardware and software design. Many of these incorporate technologies specific to the subject matter at hand, including microprocessor technology and related aspects of computing. These publications are disclosed in my *curriculum vitae* (Ex. 1003). I have previously served as an engineering consultant and testifying witness on matters related to, and including, microprocessors, chip technology, and firmware design and optimization. My *curriculum vitae* contains more information on my background and experience, as well as the cases in which I have served as an expert witness over at least the past four years.

III. Materials Considered

18. In forming my opinions, I have reviewed the following documents and any other documents cited in this declaration:

Exhibit	Description
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 2003/0126498A1 to Bigbee et al., filed January 2, 2002, published July 3, 2003 ("Bigbee")
Ex. 1006	U.S. Patent Application Publication No. US 2004/0123201 Al to Nguyen et al., filed December 19, 2002, published June 24, 2004 ("Nguyen")
Ex. 1008	U.S. Patent No. 7,085,959 B2 to Safford, filed July 3, 2002, published August 1, 2006 ("Safford")
Ex. 1009	U.S. Patent No. US 7,296,181 B2 to Safford, filed April 6, 2004, published November 13, 2007 ("Safford-181")
Ex. 1010	U.S. Patent Application Publication No. US 2002/0144177 Al to Kondo et al., filed January 31, 2002, published October 3, 2002 ("Kondo")
Ex. 1011	U.S. Patent No. 6,516,429 B1 to Bossen, filed November 4, 1999, published February 4, 2003 ("Bossen")
Ex. 1012	U.S. Patent No. 6,158,015 to Klein, filed March 30, 1998, published December 5, 2000 ("Klein")
Ex. 1013	U.S. Patent Application Publication No. US 2005/017216 A1 to Fox et al., filed January 21, 2004, published August 4, 2005 ("Fox")

Exhibit	Description
Ex. 1014	U.S. Patent No. 7,404,105 B2 to Arai, filed August 16, 2004, published July 22, 2008 ("Arai")
Ex. 1015	U.S. Patent No. 6,675,324 B2 to Marisetty et al., filed September 27, 1999, published January 6, 2004 ("Marisetty")
Ex. 1016	U.S. Patent Application Publication No. 2003/0074601 A1 to Schultz et al., filed September 28, 2001, published April 17, 2003 ("Schultz")
Ex. 1017	U.S. Patent Application Publication No. 2003/0126142 A1 to Tu et al., filed December 31, 2001, published July 3, 2003 ("Tu")
Ex. 1018	U.S. Patent No. 5,915,082 to Marshall et al., filed June 7, 1996, published June 22, 1999 ("Marshall")
Ex. 1019	U.S. Patent No. 5,904,733 to Jayakumar, filed July 31, 1997, published May 18, 1999 ("Jayakumar")
Ex. 1022	Advanced Configuration and Power Interface Specification, Revision 2.0, July 27, 2000 ("ACPI 2.0")
Ex. 1023	Intel® Itanium® Architecture Software Developer's Manual, Volume 2: System Architecture, Revision 2.1, October 2002
Ex. 1025	Prosecution History of U.S. Patent No. 7,356,733

IV. Relevant Legal Standards

19. In preparing this declaration and forming my opinions, I am relying on certain legal principles that counsel explained to me. My understanding of these concepts is summarized below.

20. I have been asked to provide my opinions regarding whether claims 1-8 and 13-22 ("the Challenged Claims") of the '958 Patent are unpatentable, because they would have been obvious to a person having ordinary skill in the art at the time of the alleged invention, in light of the prior art. I have been advised and understand that prior art includes all analogous art, and that two separate tests define the scope of analogous prior art: (1) whether the art is from the same field of endeavor, regardless of the problem addressed and, (2) if the reference is not within the field of the inventor's endeavor, whether the reference still is reasonably pertinent to the particular problem with which the inventor is involved. It is my opinion that the Challenged Claims would have been obvious to a person having ordinary skill in the art at the time of the alleged invention.

21. In forming my opinions expressed below, in addition to my knowledge and experience based upon my work in the fields of computer architecture design, including fault tolerant computer architecture design, I have considered the documents listed in the Table in Section III above.

22. I have been advised and understand that a dependent claim is a patent claim that refers back to another patent claim. I have been informed and understand that a dependent claim includes all of the limitations of the claim to which it refers.

A. Validity

23. I have been advised and understand that there are two ways in which prior art may render a patent claim unpatentable. First, the prior art can "anticipate" the claim. Second, the prior art can make the claim "obvious" to a person of ordinary skill in the art. I understand that for an invention claimed in a patent to be patentable, it must not be anticipated and must not be obvious based on what was known before the invention was made.

24. I have been advised and understand that a patent claim is unpatentable as anticipated under 35 U.S.C. § 102 if each element of that claim is present either explicitly or inherently in a single prior art reference. I have also been advised and understand that, to be an inherent disclosure, the prior art reference must necessarily disclose the limitation. The fact that the reference might practice or contain a claimed limitation is insufficient to establish that the reference inherently teaches the limitation.

25. I have been advised and understand that a claimed invention is unpatentable under 35 U.S.C. § 103 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 at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. I have also been advised and understand

that the obviousness analysis takes into account factual inquiries, including the level of ordinary skill in the art, the scope and content of the prior art, and the differences between the prior art and the claimed subject matter.

26. I have further been advised and understand that the Supreme Court has recognized several rationales for combining references or modifying a reference to show obviousness of claimed subject matter. Some of these rationales include the following: (a) combining prior art elements according to known methods to yield predictable results; (b) simple substitution of one known element for another to obtain predictable results; (c) use of a known technique to improve a similar device (method, or product) in the same way; (d) applying a known technique to a known device (method, or product) ready for improvement to yield predictable results; (e) choosing from a finite number of identified, predictable solutions, with a reasonable expectation of success; and (f) 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 teachings to arrive at the claimed invention.

27. When considering the issue of obviousness, I have been advised and understand that I should consider (1) the scope and content of the prior art; (2) the differences between the prior art and the claims at issue; (3) the level of ordinary

skill in the art; and (4) evidence of secondary indicia of nonobviousness. I have been informed that secondary indicia of nonobviousness include (i) "long-felt need" for the claimed invention, (ii) commercial success attributable to the claimed invention, (iii) unexpected results of the claimed invention, and (iv) "copying" of the claimed invention by others.

B. Level of Ordinary Skill

28. In rendering the opinions set forth in this declaration, I have been asked to consider the '958 Patent's claims and the prior art through the eyes of a person of ordinary skill in the art. I understand that a person of ordinary skill in the art is determined by considering (1) the type of problems encountered in the art, (2) prior art solutions to those problems, (3) the rapidity with which innovations are made, (4) the sophistication of the technology, and (5) the educational level and years of experience level of those working in the pertinent field.

29. I understand that I must evaluate the '958 Patent from the perspective of a person of ordinary skill in the art. That is, the '958 Patent must be evaluated through the eyes of a person with ordinary skill in the relevant art at the time of the alleged invention of this patent as of its earliest priority date. It is my opinion that such a person of ordinary skill in the art as related to the '958 Patent 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 experience relating to fault tolerant computer architecture design, as of the earliest priority date of the '958 Patent—October 25, 2004. If a person did not have a bachelor's degree, additional relevant and practical experience with fault tolerant computer systems could provide an acceptable substitute. Likewise, further education beyond a bachelor's degree could compensate for a deficiency in practical experience.

30. Based on my educational background and experience, such as my experience with computer architecture design, including fault tolerant computer architecture design (*see* paragraphs 4-17), I am qualified as at least a person of ordinary skill in the art with respect to the '958 Patent. Thus, I am familiar with the knowledge of the person of ordinary skill in the art at the time of the alleged invention. I am able to opine on how a person of ordinary skill in the art would have understood the disclosure and claims of the '958 Patent, the disclosures of the prior art, the motivation to combine the prior art, and what combinations would have been obvious and not have been obvious to one of ordinary skill in the art.

C. Claim Construction

31. I have been informed that in an *inter partes* review proceeding, claim terms should be construed under the same standard applied in federal district court

cases. Under this standard, I have been informed that claim terms are generally given their ordinary and customary meaning as understood by one of ordinary skill in the art in light of the specification and the prosecution history pertaining to the patent. I understand, however, that claim terms are generally not limited by the embodiments described in the specification.

32. I understand that in addition to the claims, specification, and prosecution history, other evidence may be considered to ascertain the meaning of claim terms, including textbooks, encyclopedias, articles, and dictionaries. I understand that the specification is highly relevant to the claim construction analysis and can be the single best guide to the meaning of a disputed term. I have been informed that the specification acts as a dictionary when it expressly defines terms used in the claims or when it defines terms by implication. I have been informed that this additional evidence is often less significant and less reliable than the claims, specification, and prosecution history.

33. I have also been informed that claims should only be construed to the extent necessary to resolve any controversy.

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

34. Based on my review of these materials and my personal knowledge and experience, 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. Thus, 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.

35. The specification provides detecting these two types of errors as examples of detection of LOL. *See e.g.*, Ex. 1001, 2:26-59, 4:37-67. It states that "[l]ockstep mismatch is one way of detecting an LOL"—(mismatch errors, definition 1). Ex. 1001, 4:56-57. It also states that "[b]ecause the detection of LOL … may occur before an actual lockstep mismatch occurs, the detection of LOL … may be referred to as a detection of a 'precursor to lockstep mismatch'"—(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.

36. The 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, 30-49 (claims 1, 4-8).

37. Claim 4, for example, further defines "detecting loss of lockstep" as comprising "detecting corrupt data … ***that would lead to a lockstep mismatch***." Similarly, claim 5 defines "detecting loss of lockstep" as comprising "***detecting a precursor to lockstep mismatch***." Both claims 4 and 5 expressly require that a precursor to lockstep mismatch (errors that may eventually cause a lockstep mismatch error—definition 2) are included in the definition of "detecting loss of lockstep."

38. 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.

39. Accordingly, "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.

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

40. The terms "lockstep is recoverable" and "loss of lockstep is recoverable" should be construed as "lockstep is recoverable from the detected loss of lockstep" because the specification only describes lockstep recovery from the detected LOL—"whether the detected LOL is a recoverable LOL." *See e.g.*, Ex. 1001, 5:49-62.

3. All Other Claim Terms

41. 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.

V. The State of Art

A. Lockstep Processors or FRC Processors Were Known

42. Before the time of the '958 Patent, a typical approach to improve fault tolerance of a computer system is to use lockstep processing. For example, as shown below, two processors 22 and 24 (or two processor units or two processor cores) are paired together to execute the same instructions on identical data in parallel; and the outputs of the two processors are compared by a comparator or a lockstep logic 26, such as an XOR gate. Safford, 1:32-43; *see also* Bigbee, [0033]; Nguyen, [0006]; Kondo, [0004]; Ex. 1001, 2:13-19. A discrepancy between the

outputs of the pair of processors detected by logic 26 is a loss of lockstep. Safford, 1:51-55.

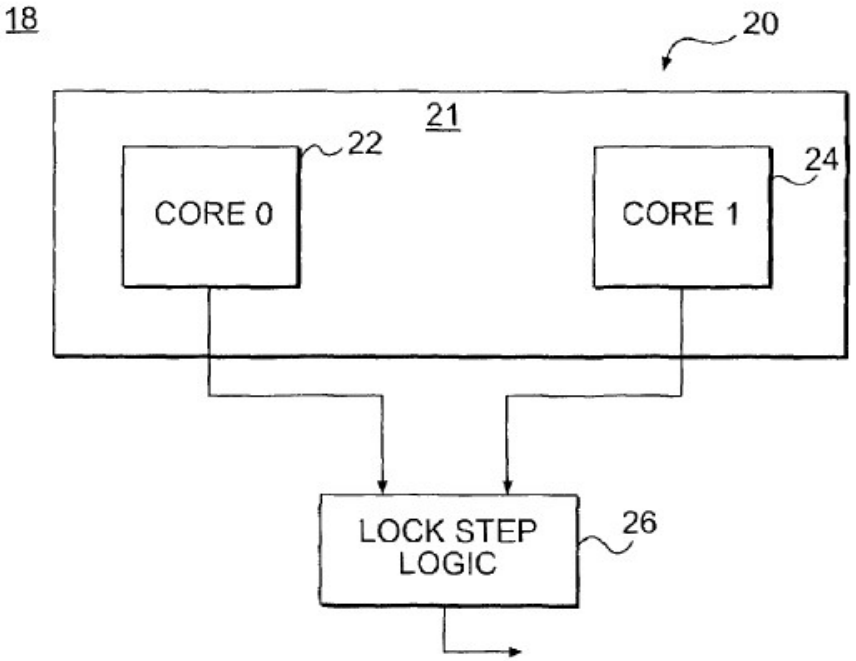


FIG. 1B
PRIOR ART

Safford, FIG. 1B.

43. Because lockstep processors execute redundant instructions in parallel, lockstep systems are said to perform a "functional redundancy check." Safford-181, 1:67-2:3. Thus, processors operating in lockstep were also referred to as processors operating in functional redundancy check mode or FRC mode. A person of ordinary skill in the art would have understood that both the terms FRC

processor or a lockstep processor refer to a processor including two or more processor cores operating in FRC mode or in lockstep.

44. If comparing the outputs of the pair of processors indicates that they have produced the same outputs, it is assumed that there is no error in processing. If the comparison indicates a difference or mismatch, it is assumed that at least one processor has an error. Safford, 3:10-20. An example of such an error is a data cache error or a soft error.[1] *Id.*, 1:46-50; 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. *Id.*, 1:46-50.

45. It was conventional and well known to a person of ordinary skill in the art that 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

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

change over time, and that processors have status registers for recording such designations. Marshall, 1:11-16; Kondo, [0086]; Tu, [0031].

46. It was also known that an error detected in an individual processor of a pair of lockstep processors signals an impending mismatch error. Safford, 3:5-34 (error detected in an single processor core of a lockstep processor signals an *impending* loss of lockstep); Kondo, [0101] (detecting an error in the processor provides *early warning* of a divergence), [0103] ("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 lockstep), [0022] (data corruption in a processor core of an FRC processor can *trigger* an FRC error).

B. Determining Whether Lockstep Is Recoverable Was Known

47. It was known that lockstep is recoverable from some errors in a lockstep processor, such as soft errors, but not recoverable from some other errors, such as some lockstep mismatch errors.

48. For example, lockstep is recoverable from an error detected in an individual processor before a mismatch error is detected and signaled. As shown below, a pair of lockstep processors 111 and 113 can include error detection and signaling logic 112 and 114, respectively, to signal an impending loss of lockstep. Safford, 3:21-51.

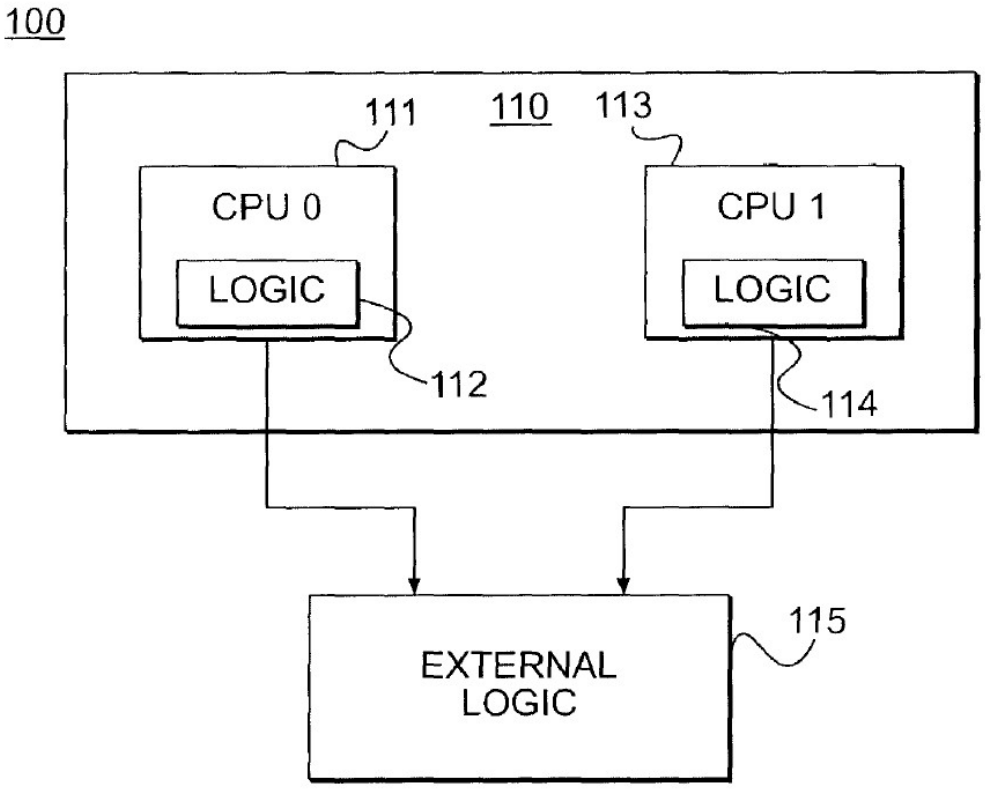


FIG. 2

Safford, FIG. 2.

49. It was known that such detecting and handling such error before signaling a mismatch error improves processor or system availability. Safford, 3:21-51 (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 indicating that a divergence will occur but will not compromise data integrity is a correctable memory error).

50. It was also known that lockstep is 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., partial or total hardware failure). Bigbee, [0033] (such inconsistencies can occur as a result of partial or total hardware failure; Bossen, 1:35-40 (no recovery is possible for hard errors). Also, lockstep is not recoverable if the processor or FRC logic does not know which processor core was responsible for the mismatch error. Nguyen, [0023] ("Not all FRC-errors are traceable to underlying parity/ECC or other correctible soft errors.");[2] Safford-181,

[2] A person of ordinary skill in the art would understand that the acronym "ECC" refers to error checking and correction, a widely-used technique for coding data such that errors can not only be detected, but also corrected.

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."). 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). And lockstep is not recoverable if the mismatch error is detected in the absence of detecting an error in the individual processor. Marshall, 2:17-34 (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 not who failed). Such mismatch error typically would indicate a high risk of corrupt data or 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

51. It was known that once an error was detected in FRC processors, the appropriate action responsive to the recoverability of the detected error should be

taken to handle non-recoverable and recoverable errors to reduce impact of the errors on the system.

52. With respect to non-recoverable errors, for example, it was known that when a non-recoverable error occurs, there is risk of corrupt data or results propagating through the system. An appropriate action for handling 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 for handling a non-recoverable error), [0190] (shut down processor under conventional non-recoverable error handling, 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.).

53. Also, a mismatch error can be non-recoverable because it indicates a non-recoverable partial or hardware failure of the processor (Bigbee, [0033]), for

which an appropriate action would have been replacing it with another processor. Klein, 1:19-66 (hot swapping a failed CPU while the system is running); Fox, Abstract, [0005]-[0014] (replacing a failing processor with a hot spare processor); Arai, 1:39-45, 2:23-41 (replacing a failed processor with a reserved processor).

54. With respect to recoverable errors, various recovery routines have been used for handling such errors to reduce the impact of such errors on the 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).

55. Treating an otherwise recoverable error as a non-recoverable error would reduce system availability. Recovery of lockstep from a recoverable error can be implemented with a lower latency as compared to the reset or repair/replacement processes for handling a non-recoverable error, which can result in system availability loss over a longer latency. Nguyen, [0022]-[0023], [0040].

56. Treating a non-recoverable error as a recoverable error would reduce system reliability. For example, it would create risks of having corrupt data or results propagating through the system. Kondo, [0101] (the 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).

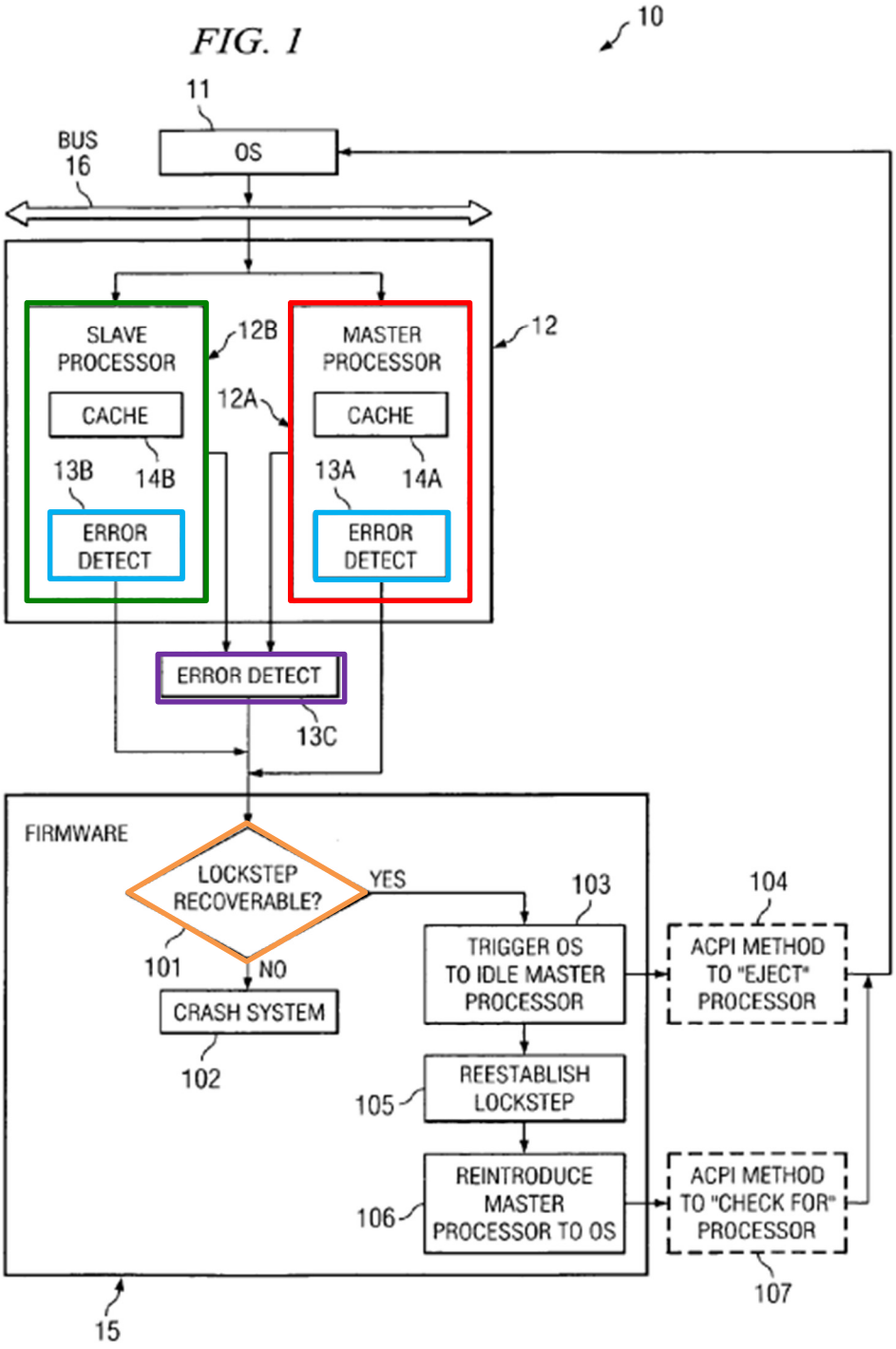
D. Using Firmware to Detect and Handle Processor Errors Was Known

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

58. It was also common and known that firmware can be used for handling processors 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 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).

VI. Overview of the '958 Patent

59. The '958 Patent claims priority to October 25, 2004. Ex. 1001, 1. The 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 system includes a lockstep processor pair, including a master processor and a slave processor, each with error detect logic to detect errors present in the respective processor. *Id.*, 4:14-46, FIG. 1 (reproduced below). The system also includes 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).

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

61. The '958 Patent's independent claims recite six primary components, all of which were disclosed in the prior art and all of which would have been obvious to a person of ordinary skill in the art 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. Overview of the Prior Art

A. Overview of Bigbee (Ex. 1005)

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

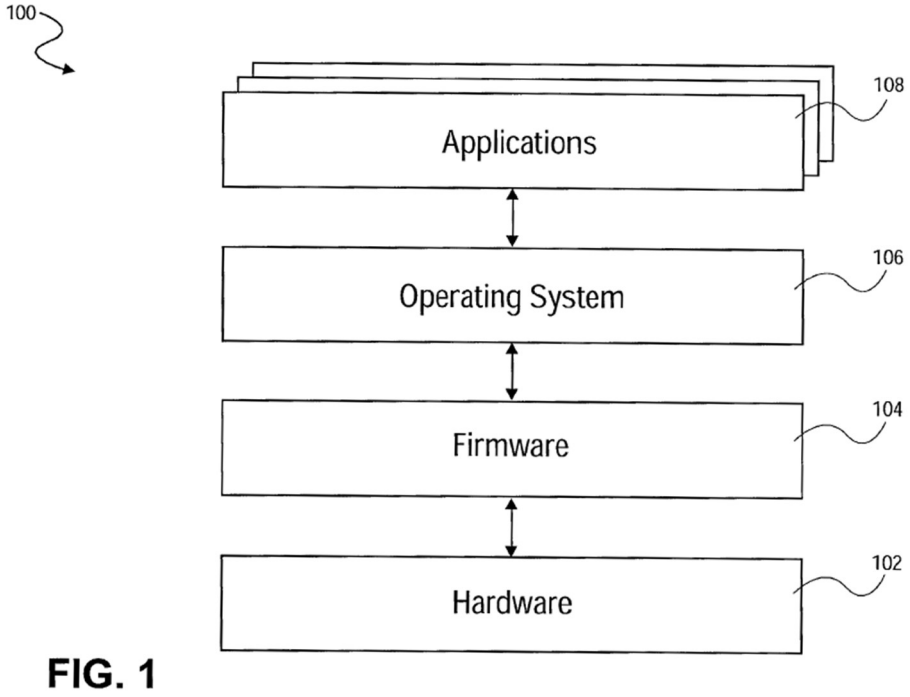


FIG. 1

Bigbee, FIG. 1.

63. Hardware 102 includes two or more processor packages, 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 operating system 106. *Id.*, [0019], [0032]-[0033], FIG. 3 (reproduced below); *see above* Sections V.A, VII.A. The two processor cores concurrently

execute identical instructions on identical data and each processor generates results. *Id.*, [0002], [0033]. An FRC logic 308 monitors and detects inconsistencies in the results of the two processor cores that indicate an FRC mismatch error, which may be the result of an alpha particle impacting on one or both processor cores. *Id.*

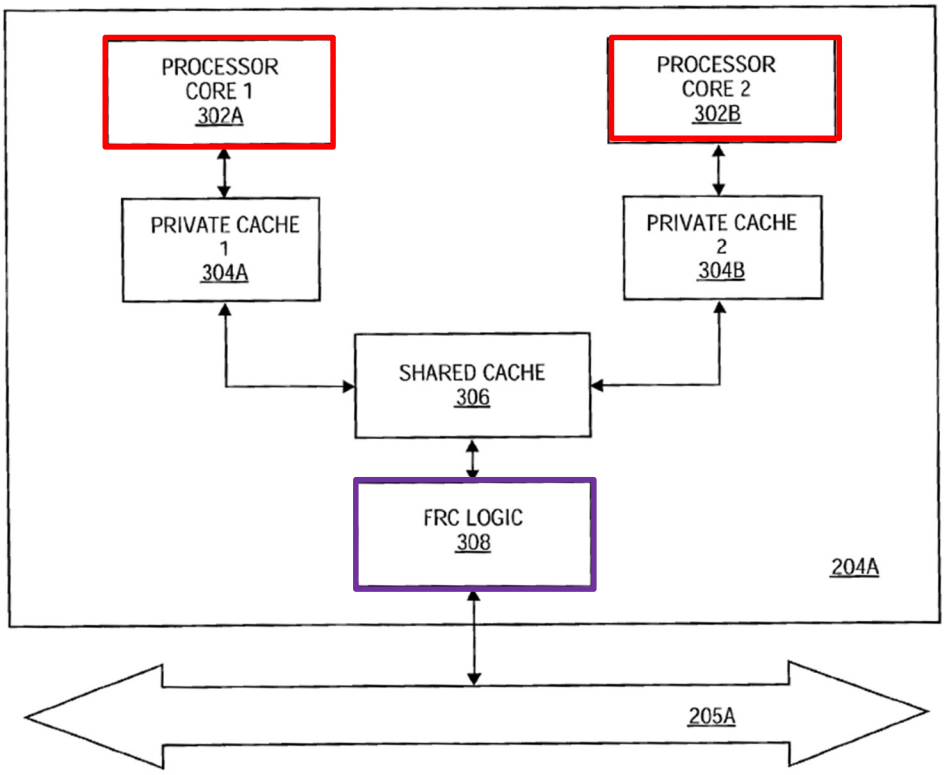


FIG. 3

Bigbee, FIG. 3 (annotated).

64. The FRC logic also detects an error within each of the processor core and generates a system machine check to cause the firmware and the operating

system to perform an error recovery routine that recovers the FRC operation ("lockstep") of the cores. *Id.*, [0035]-[0037], FIGS. 4A-4B.

B. Overview of Nguyen (Ex. 1006)

65. Nguyen discloses a processor including two cores 120A and 120B (e.g., Itanium® family processors) 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 FRC checker 130 and error detector 140. *Id.*, [0021]. Error pipeline 128 represents the logic that operates on various types of data to detect errors in the data and 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].

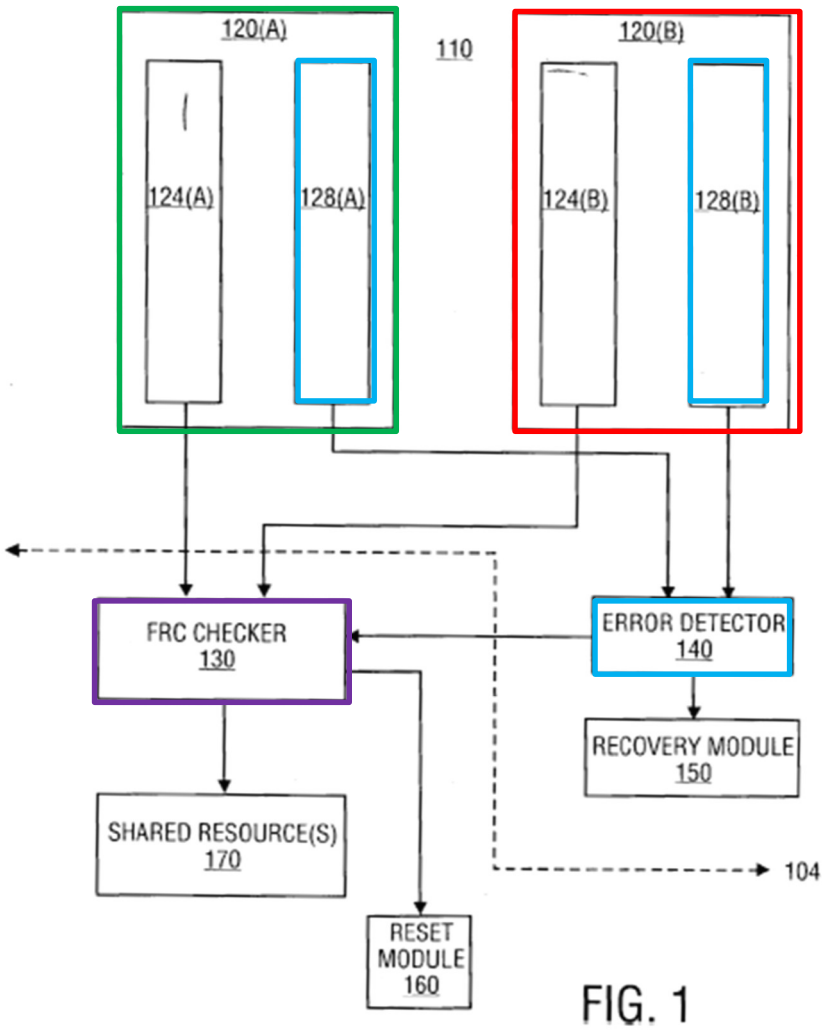


FIG. 1

Nguyen, FIG. 1 (annotated).

66. 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].

67. ***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 the 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, so the whole system needs to be reset.

68. ***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., result from 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.*

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

unit 734 sets the status flag to indicate the mismatch and triggers timer unit 738 to begin a countdown interval. *Id.* If error detector 140 receives an error flag before the timeout interval expires, it disables FRC checker 730 and triggers recovery module 150. *Id.* But if the timeout interval does expire, the FRC checker signals a non-recoverable FRC error and triggers reset module 160 to reset the system. *Id.*, [0022], [0056]-[0058].

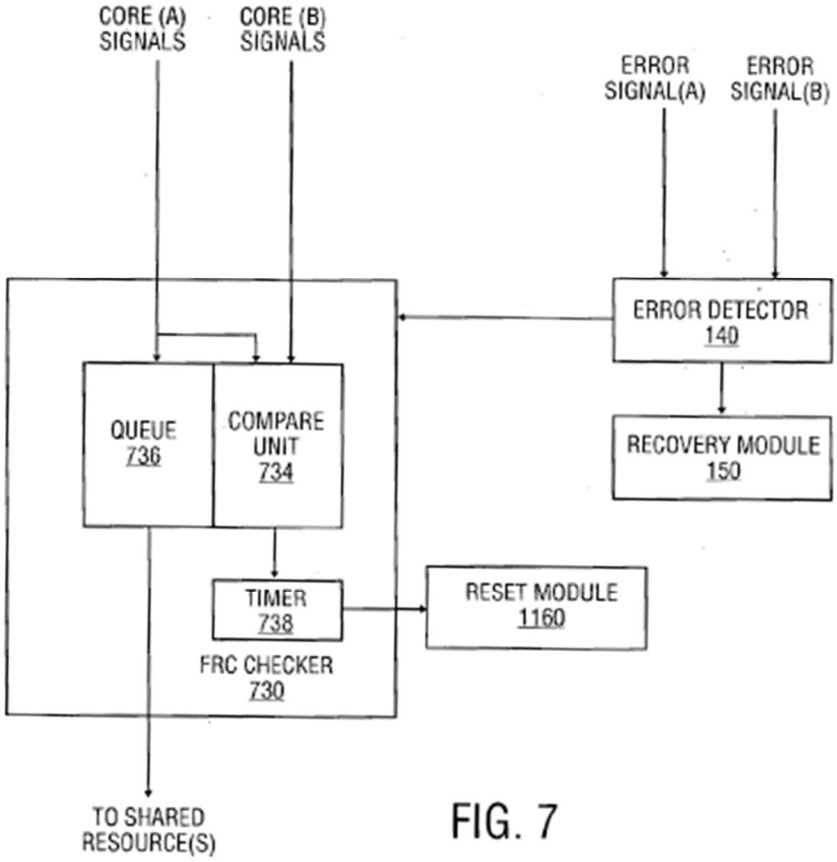


FIG. 7

Nguyen, FIG. 7. Nguyen's process for determining if lockstep is recoverable from an error in a lockstep processor pair was known in the prior art. *See e.g.*, Tu (Ex. 1017), [0041]-[0043], FIG. 5.

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

70. Like the '958 Patent, both Bigbee and Nguyen relate to fault tolerant systems that operate despite loss of lockstep in the system. A person of ordinary skill in the art 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.'").

71. A person of ordinary skill in the art 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. Section IV.C.

[3] A person of ordinary skill in the art would have understood that an FRC processor or a lockstep processor refers to a processor including two or more processor cores operating in FRC mode or in lockstep.

72. A person of ordinary skill in the art would have understood that, like the ’958 Patent, both Bigbee and Nguyen disclose that FRC mode or lockstep of the processor cores is 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 PAL, SAL, or BIOS, to implement a recovery routine to recover lockstep of the processor cores. Bigbee, [0018]-[0023], [0035]-[0037]; Nguyen, [0045]-[0046]. Accordingly, Bigbee and Nguyen are analogous to the ’958 Patent.

VIII. Ground of Unpatentability

73. Based on my review of the materials set forth above, including my application of the knowledge of a person of ordinary skill in the art, it is my opinion that the Challenged Claims of the ’958 Patent would have been rendered obvious by certain prior art to one of ordinary skill in the art.

74. In particular, it is my opinion that the Challenged Claims of the ’958 Patent would have been rendered obvious to one of ordinary skill in the art, based on the ground shown below:

Ground	Claims	Reference(s)	Basis
1	1-8, 13-22	Bigbee in view of Nguyen	§ 103

IX. Ground 1: Claims 1-8 and 13-22 Are Obvious over Bigbee in view of Nguyen

75. In my opinion, Bigbee in view of Nguyen renders obvious claims 1-8 and 13-22.

A. Bigbee in view of Nguyen

76. Bigbee discloses most of the challenged claim limitations. Bigbee discloses a pair of lockstep processors and using firmware and an operating system to detect and recover loss of lockstep using firmware. Bigbee, therefore, discloses all elements of claims 1-8 and 13-22 except for those elements that require determining if the lockstep of a lockstep processor pair is recoverable.

77. 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. Bigbee's firmware performing Nguyen's determining if the lockstep is recoverable or non-recoverable renders claims 1-8 and 13-22 obvious. In addition, a personal of ordinary skill in the art would have known that Nguyen's determination can be performed with firmware. *See above* Section V.D.

78. Applying the combination of Bigbee and Nguyen to the six core components of the independent claims, 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

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

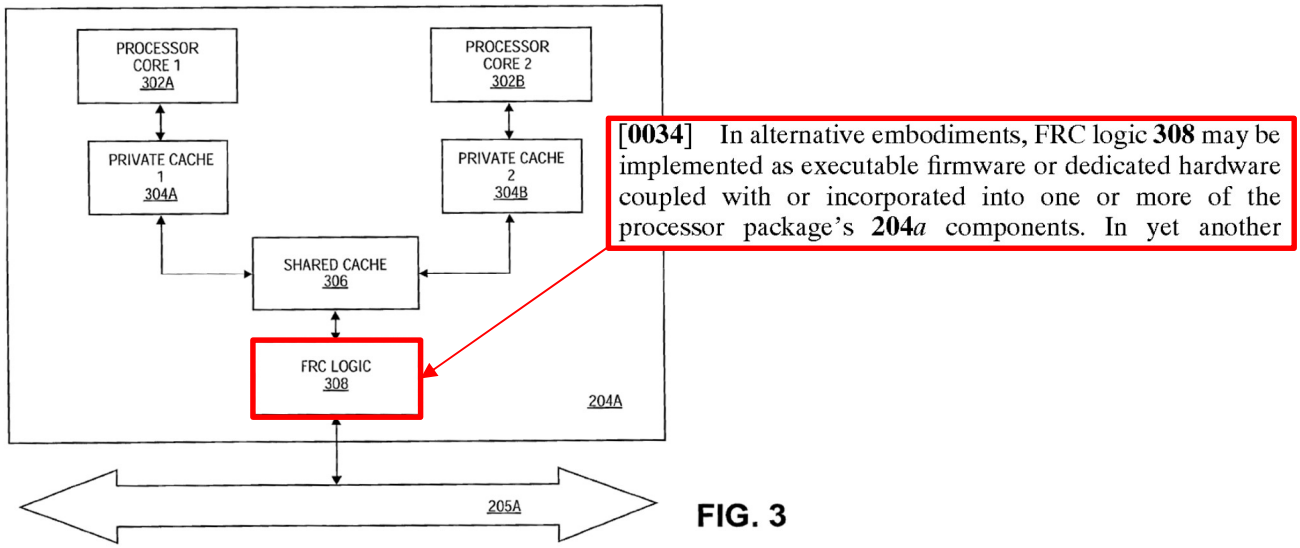
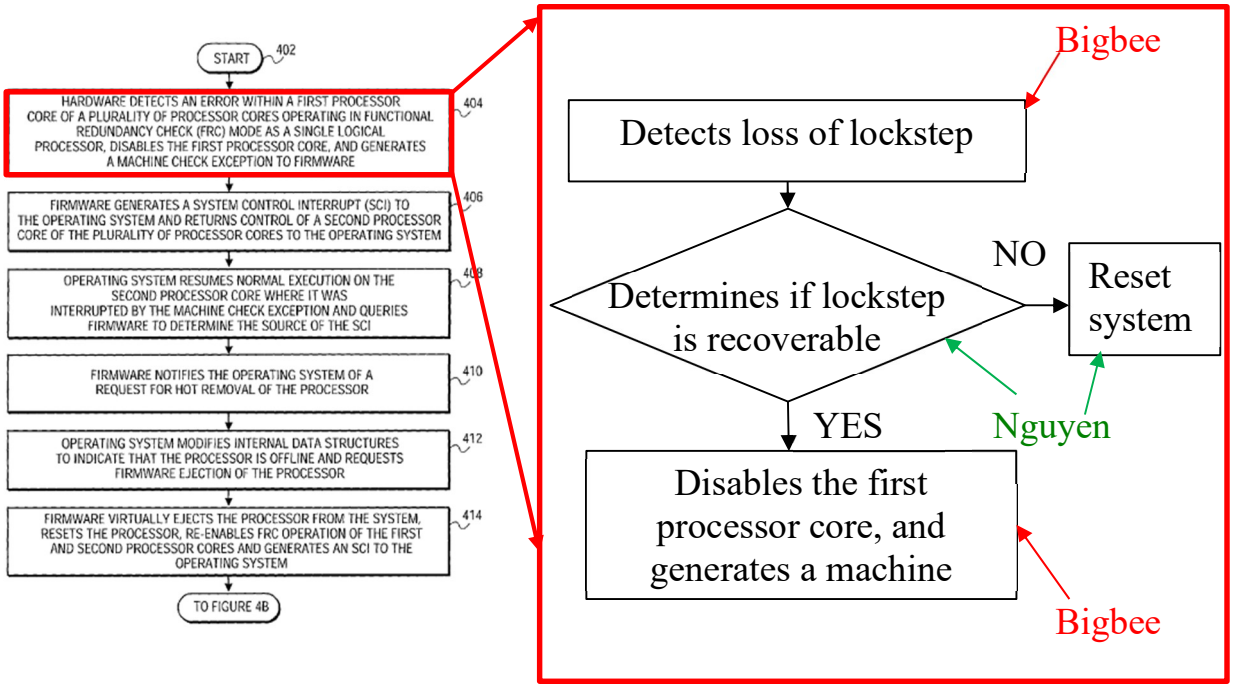


FIG. 3

Bigbee, FIG. 3 (annotated).



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

B. A Person of Ordinary Skill in the Art Would Have Found it Obvious to Combine Bigbee and Nguyen

80. At 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. A person of ordinary skill in the art 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 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.

81. 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 it only discloses a responsive routine for recovering from recoverable errors. *See above* Section VII.A. Bigbee does not disclose handling non-recoverable errors. Because Bigbee only discloses recovering from recoverable lockstep errors, it 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).

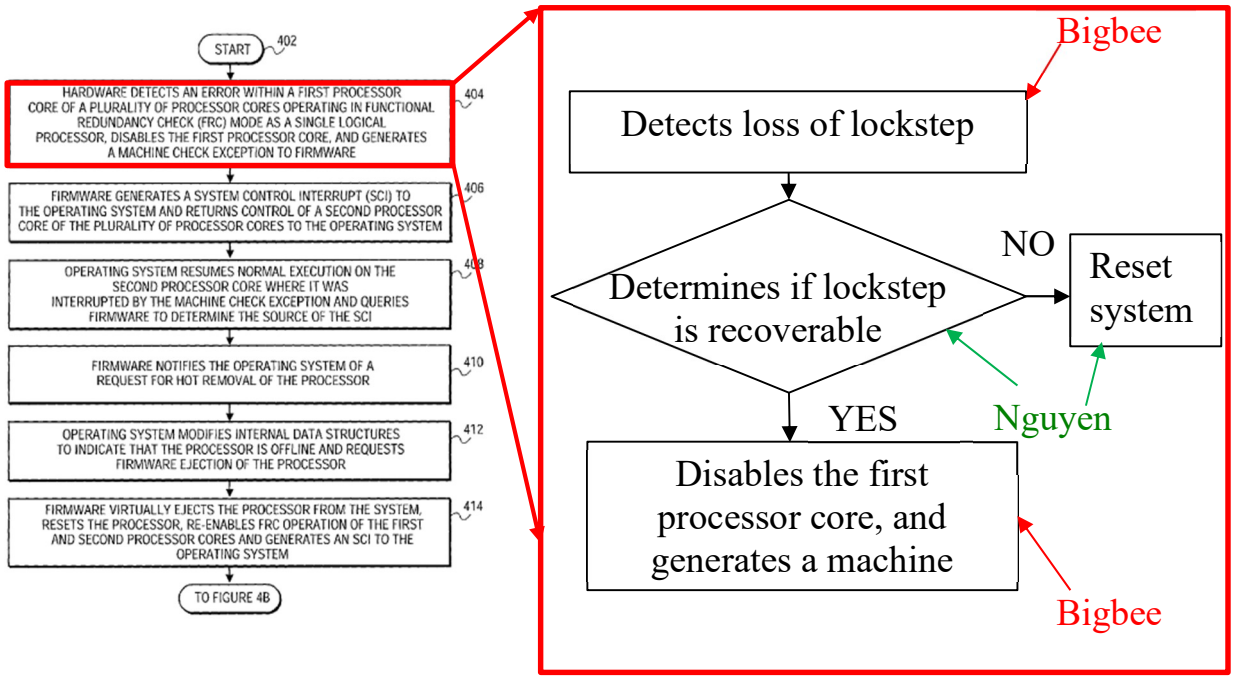
82. A person of skill in the art at the time would have recognized that Bigbee's disclosure omits the portion of the process after a non-recoverable error is detected (e.g., a mismatch error when the processor causing the error cannot be determined) and would have been motivated to add a process for handling non-recoverable errors. Nguyen is one such reference.

83. 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].

84. A person of ordinary skill in the art 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. *See* 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).

85. The Bigbee-Nguyen system 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).

86. As I stated in the preceding section, a person of ordinary skill in the art 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 person of ordinary skill in the art 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 person of skill in the art would have been motivated to add Nguyen's determining whether lockstep is recoverable from the detected error to fill a gap in Bigbee's disclosure.

87. A person of ordinary skill in the art would have been motivated to implement Nguyen's determination of whether the detected error causing the loss of lockstep is recoverable by Bigbee's firmware FRC logic to appropriately handle the detected error. Because different responsive actions should be taken to handle each of non-recoverable and recoverable errors, determining whether the error is recoverable or non-recoverable is critical to reduce the impact of 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); Shultz, [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).

88. As I explained 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. And, as I also explained in Section V.C, recoverable errors can be recovered using known recovery routines that reduce the impact on the system. A person of ordinary skill in the art would have understood that it is critical to handle recoverable errors differently from non-recoverable errors.

89. 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.*

90. 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, 1:46-50 (a data cache error in one processor, if not promptly corrected, may cause the computer system to crash).

91. 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).

92. Accordingly, a person of ordinary skill in the art would have understood that fault tolerant systems would distinguish between recoverable and

non-recoverable errors by first determining whether an error is recoverable and then responding accordingly based on the determination.

93. Based on a person of ordinary skill in the art'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 person of skill in the art would have understood how to use Bigbee's disclosed firmware to perform Nguyen's determining whether a detected error is recoverable or non-recoverable.

94. A person of ordinary skill in the art 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)).

95. A person of ordinary skill in the art 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].

96. A person of ordinary skill in the art would have known how to use ACPI Source Language (ASL) to program Bigbee's firmware code to implement Nguyen's determining step using ordinary computer programming skill. 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. Shultz, [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 Itanium® family 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).

97. A person of ordinary skill in the art 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 person of ordinary skill in the art would have been familiar with logic design and would have understood the concepts of logic delays, timing, and the possibility of race conditions as well the ability to handle such conditions using each of the alternatives.

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

C. Independent Claim 19

99. In my opinion, Bigbee in view of Nguyen renders obvious claim 19.

1. [19preamble]: "A system comprising a pair of lockstep processors and computer-executable firmware code stored to a computer-readable medium, the computer-executable firmware code comprising:"

100. Bigbee discloses [19preamble] because Bigbee discloses a data processing system 100 including hardware 102, firmware 104, and operating system 106, as shown below. Bigbee, [0014]-[0025], FIG. 1.

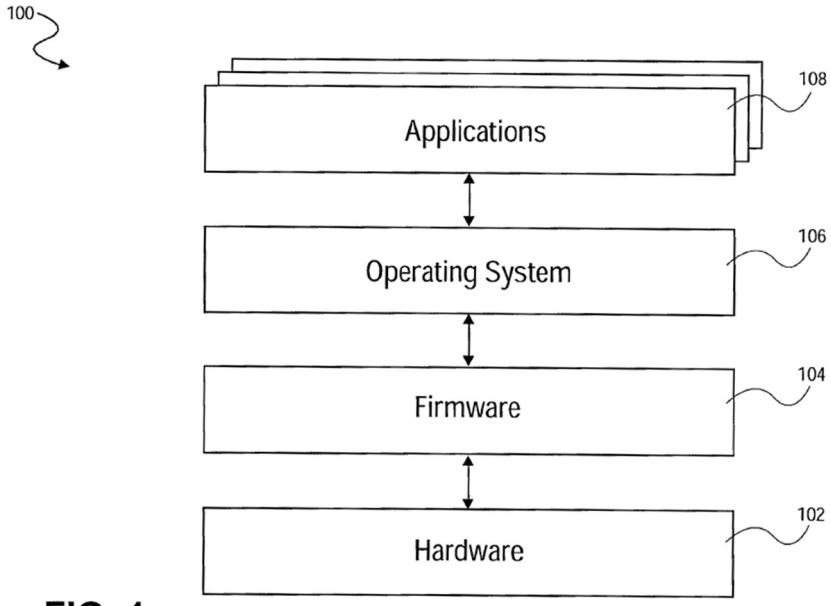


FIG. 1

Bigbee, FIG. 1. Hardware 102 includes two or more processor packages, each processor package including two or more processor cores operating in lockstep ("a pair of lockstep processors") as a single logical processor from the point of view of operating system 106. *Id.*, [0019], FIG. 3; *see above* Section VII.A. Firmware 104

includes ACPI BIOS including ACPI machine language program code ("computer-executable firmware code"). *Id.*, [0016]-[0017].

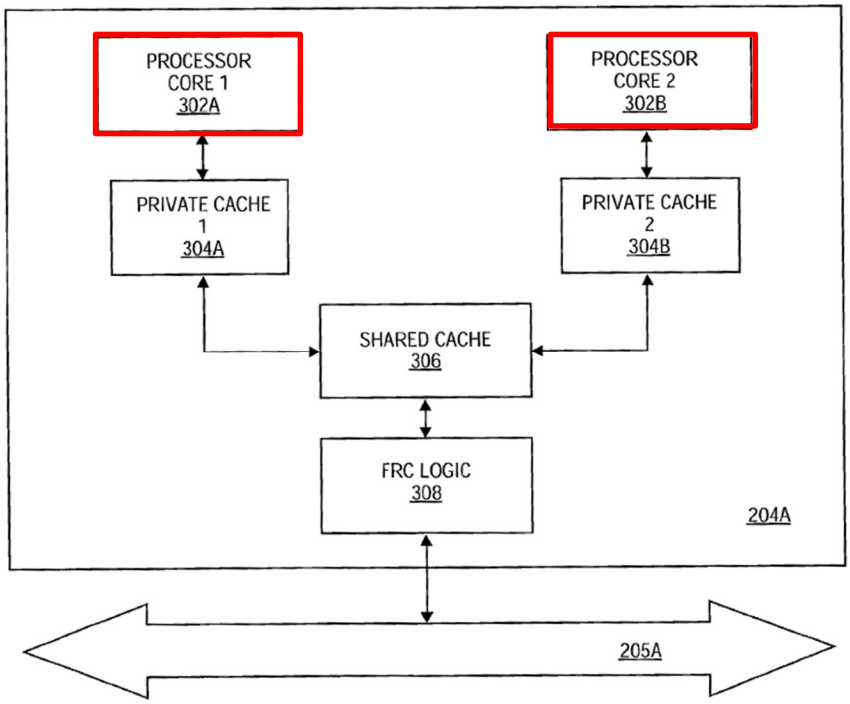


FIG. 3

Bigbee, FIG. 3.

101. As shown in FIG. 2 below, the data processing system includes two or more nodes, each including a plurality of the FRC processor packages. *Id.*, [0026]-[0027]. The system also includes a non-volatile memory 226a ("computer-readable medium") to store firmware code (e.g. SAL or ACPI BIOS program code). *Id.*, [0029], [0012]-[0013] (executable instructions embodied within a machine accessible medium, such as a ROM, accessible by a machine (e.g., a data processing system or computer)).

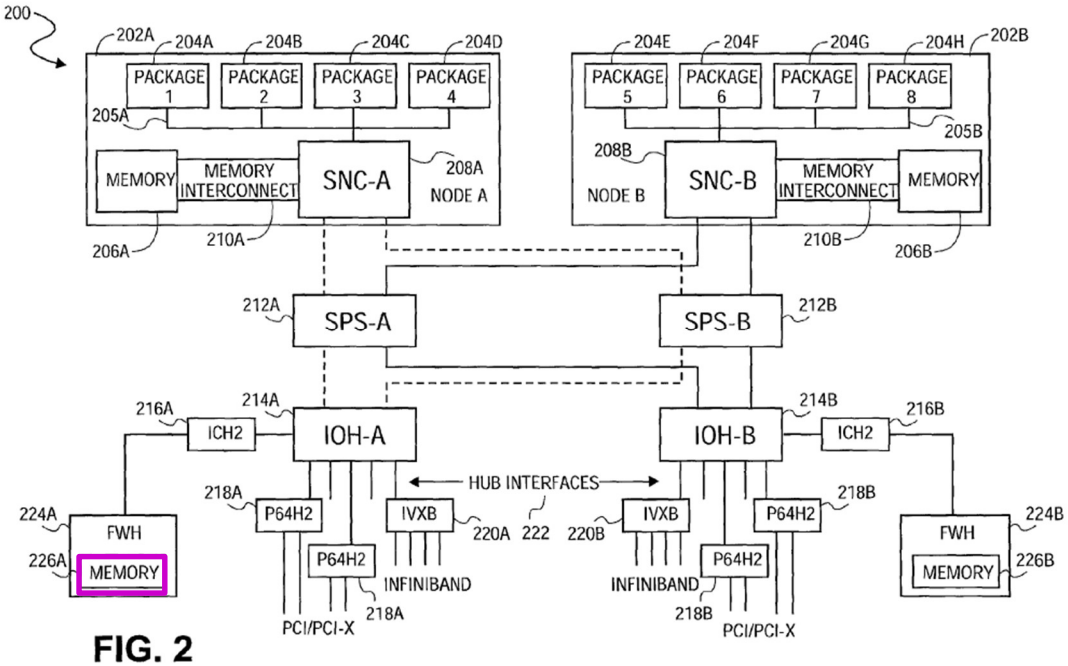


FIG. 2

Bigbee, FIG. 2 (annotated).

2. [19a]: "firmware code, responsive to detection of loss of lockstep between the pair of lockstep processors, for determining if the lockstep is recoverable for the pair of lockstep processors;"

102. Bigbee in view of Nguyen renders obvious [19a].

103. Bigbee discloses the FRC logic detects loss of lockstep between the pair of lockstep processor cores and further discloses implementing FRC logic by executable firmware. Bigbee, [0019], [0034].

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

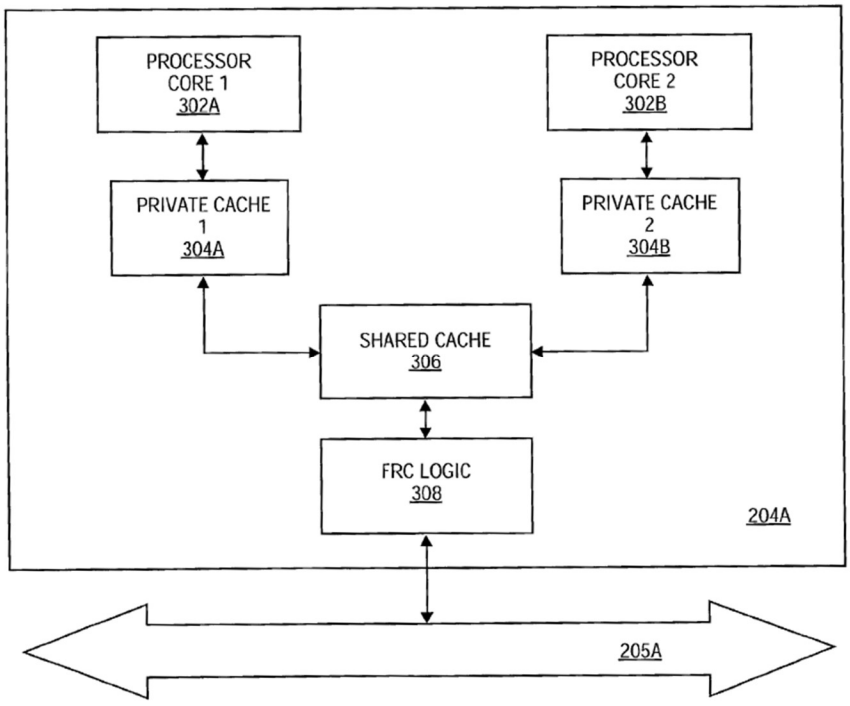


FIG. 3

Bigbee, FIG. 3.

105. Nguyen discloses determining if lockstep is recoverable in response to detection of lockstep errors. *See above* Section VII.B. Nguyen discloses responsive to the detection of an error in the processor cores ("responsive to detection of loss of lockstep between the pair of lockstep processors"), 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].

106. 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 VII.B.

107. As explained in Section IX.B.2, a person of ordinary skill in the art would have been motivated to incorporate Nguyen's determination of whether

lockstep is recoverable in Bigbee's FRC logic to reduce impact of errors and increase system fault tolerance and availability. A person of ordinary skill in the art 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 IX.B.2.

3. [19b]: "wherein said firmware code for determining if the lockstep is recoverable further comprises firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors;"

108. Bigbee in view of Nguyen renders obvious [19b].

109. Bigbee in view of Nguyen discloses "the firmware code for determining if the lockstep is recoverable" (*see above* [19a]) and further discloses "firmware code for determining if a lockstep mismatch has occurred between the pair of lockstep processors."

110. Nguyen discloses "determining if a lockstep mismatch *has occurred*" because Nguyen determines if the lockstep is recoverable or non-recoverable after detecting a mismatch error. For example, in Nguyen determining if lockstep is recoverable or non-recoverable: (1) Nguyen's FRC checker's compare unit compares the data from both cores, it sets a status flag to indicate whether the data matches or mismatches ("determining if mismatch has occurred"); and (2) determining if 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 ("determining if the lockstep is recoverable"). Nguyen, [0056]-[0057]; *see above* Section VII.B.

111. As explained in Section IX.B.2, a person of ordinary skill in the art would have been motivated to incorporate Nguyen's determination of whether lockstep is recoverable in Bigbee's firmware FRC logic to reduce impact of errors and increase system fault tolerance and availability. A person of ordinary skill in the art 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 IX.B.2.

4. [19c]: "firmware code, responsive to determining that the lockstep is recoverable, for triggering an operating system to idle the processors, attempting to recover lockstep between the pair of processors, and"

112. Bigbee in view of Nguyen renders obvious [19c].

113. As explained above, a person of ordinary skill in the art 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 IX.B. Bigbee in view of Nguyen further teaches, responsive to determining if the lockstep is recoverable from the detected error, the firmware triggers the operating system to idle the processors and recover the lockstep between the pair of processors.

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

[4] 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].

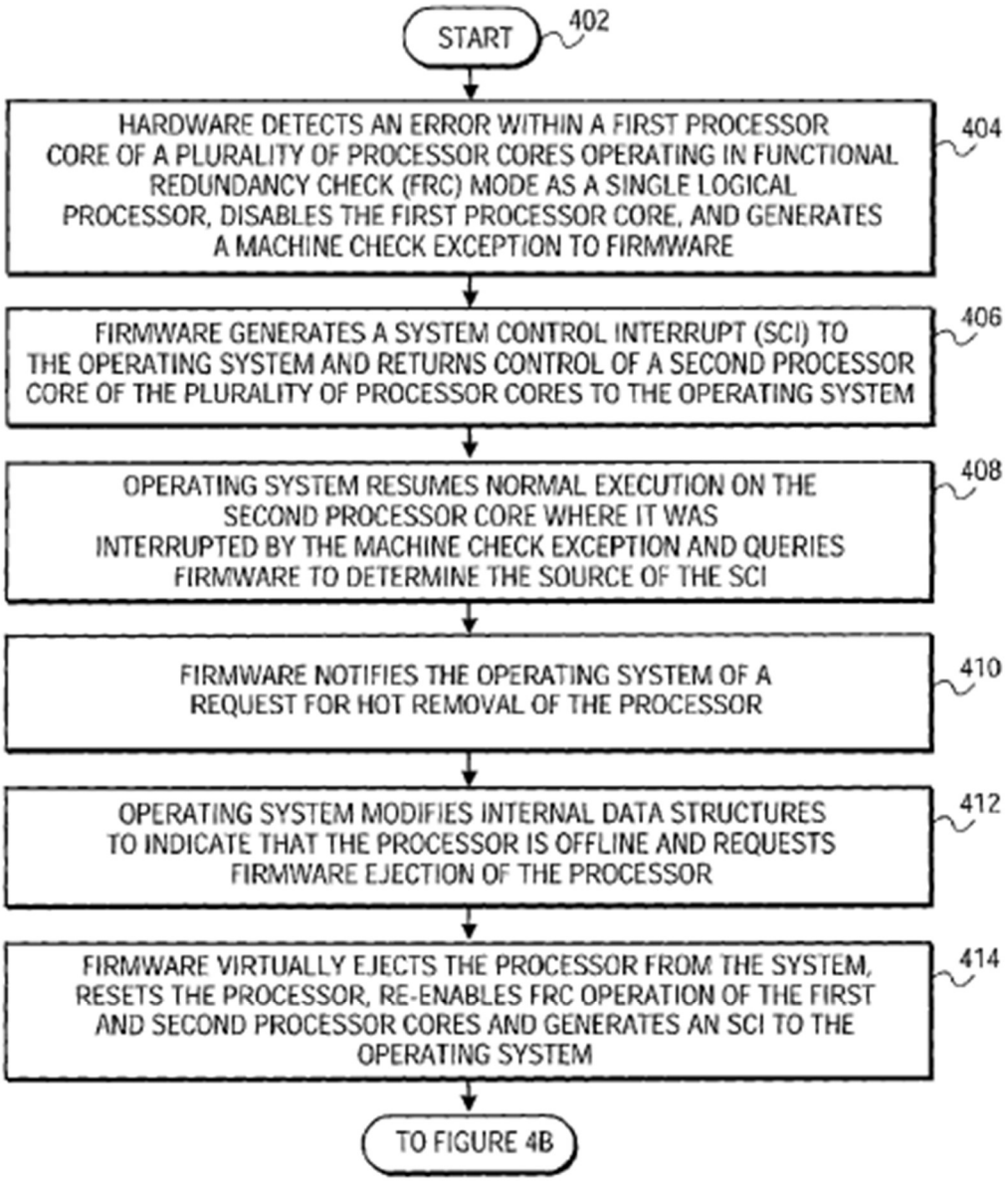


FIG. 4A

Bigbee, FIG. 4A.

115. ***Second***, Bigbee discloses "firmware code … for attempting to recover lockstep between the 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 ("attempting to recover lockstep between the pair of processors") and generates an SCI to the operating system. Bigbee, [0037], FIG. 4A (block 414).

116. 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, rests the processor pair, and recovers lockstep. Ex. 1001, 8:9-17, 6:40-42, 6:55-57.

5. [19d]: "if lockstep is successfully recovered between the pair of processors, [firmware] triggering said operating system to recognize the processors as being available for receiving instructions."

117. Bigbee discloses [19d].

118. Bigbee discloses that the firmware generates a SCI to the operating system after recovering lockstep between the pair of processors ("if lockstep is successfully recovered 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 ("triggering said 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 processors as being available for receiving instructions"). Bigbee, [0037], FIG. 4B (block 430).

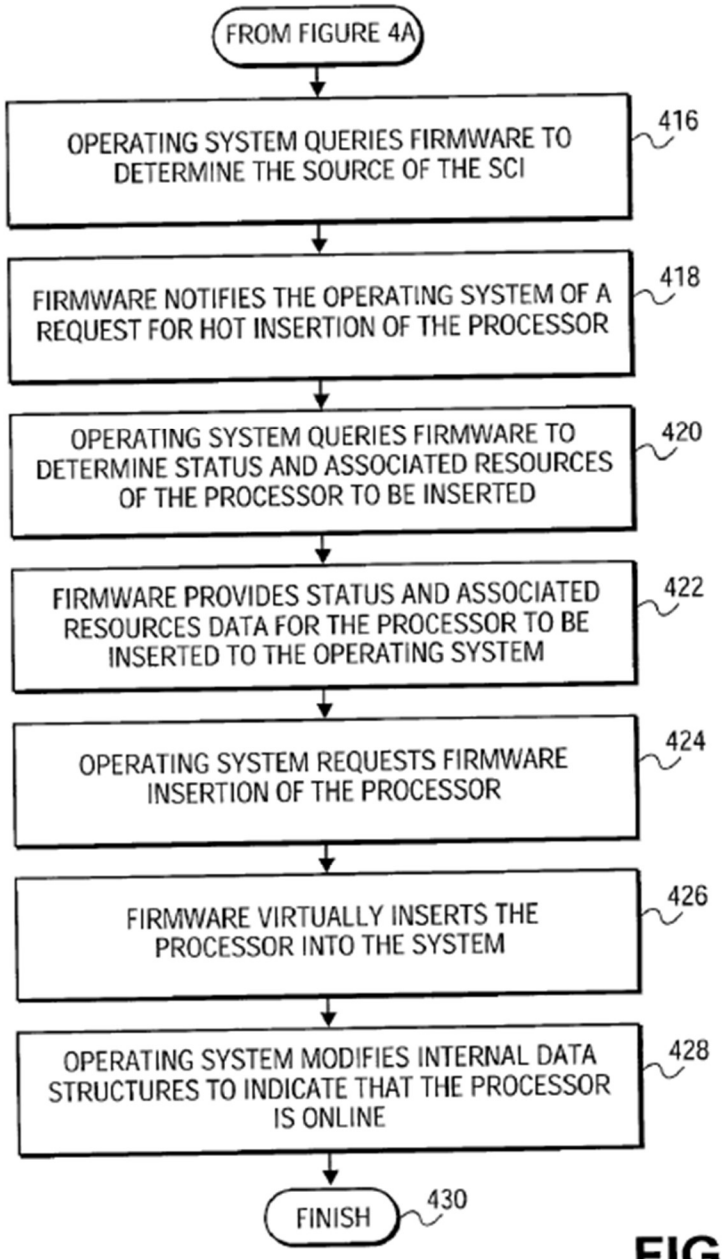


FIG. 4B

Bigbee, FIG. 4B.

119. 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. Ex. 1001, 6:55-7:12.

D. Independent Claim 1

120. Independent claim 1 is substantially similar to claim 19. Bigbee discloses claim 1's preamble because it discloses a method for lockstep recovery of a pair of lockstep processors. Bigbee, Abstract, FIGS. 4A-4B.

121. In my opinion, Bigbee in view of Nguyen renders obvious claim 1. The following chart identifies the relevant claim 19 analysis applicable for each claim 1 element.

Claim Element	Relevant Explanation
[1a] detecting loss of lockstep for a lockstep pair of processors;	Claim [19a]: Section IX.C.2
[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	Claim [19c]: Section IX.C.4 Claim [19d]: Section IX.C.5

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,	Claim [19a]: Sections IX.C.2
[1d] wherein said using said firmware to determine if lockstep is recoverable further comprises determining if a lockstep mismatch has occurred between the lockstep pair of processors.	Claim [19b]: Sections IX.C.3

E. Independent Claim 13

122. In my opinion, Bigbee in view of Nguyen renders obvious claim 13.

1. [13preamble]: "A system comprising:"

123. [13preamble] is in [19preamble]. Bigbee discloses [13preamble] for the reasons discussed in [19preamble]. *See above* Section IX.C.1.

2. [13a]: "an Advanced Configuration and Power Interface (ACPI)-compatible operating system;"

124. Bigbee discloses [13a].

125. The '958 Patent admits that its ACPI-compatible operating system makes ACPI calls or calls ACPI methods to parse the ACPI tables to discover the other processors of a multi-processor system "in a manner as is well-known in the art." Ex. 1001, 8:8-11. Bigbee discloses the use of such a well-known ACPI-compatible operating system.

126. Bigbee's data processing system comprises an ACPI-compliant data processing system. Bigbee, [0016]. The system's operating system includes operating system-directed power management (OSPM) program code, and an ACPI device driver, and its firmware comprises an ACPI BIOS including ACPI machine language (AML) program code describing what hardware devices are present within the data processing system and their configuration and interfaces, via ACPI control methods, objects, registers, and tables. *Id.* The ACPI device driver within the operating system acts as an AML interpreter to interpret and execute AML program code. *Id.* A person of ordinary skill in the art would have understood the operating system of Bigbee's ACPI-compliant data processing system is an "ACPI-compatible operating system," as described in the '958 Patent. Ex. 1001, 8:8-11.

3. [13b]: "master processor operable to process instructions scheduled by said ACPI-compatible operating system;"

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

127. In my opinion, Bigbee in view of Nguyen renders obvious [13b] and [13c].

128. Bigbee discloses its processor package includes a pair of processor cores that are "lockstepped with" each other "operable to perform redundant processing" because they operate in lockstep FRC mode as a single logical processor from the point of view of operating system software, where they are provided with identical instructions concurrently executed on identical data ("process instructions"). Bigbee, [0033]. A person of ordinary skill in the art would have understood that the identical instructions provided to Bigbee's processor cores are "scheduled by [its] ACPI-compatible operating system" because it was conventional and well-known that the operating system schedules tasks that include instructions to processors for execution.

129. It was also conventional and well-known that in a multi-processor system like Bigbee, the operating system schedules tasks among the multiple processors to make full use of processor capacity for high performance. Furthermore, both Bigbee and Nguyen disclose its processors can be the Itanium®

family processors (Bigbee, [0033], Nguyen, [0039]), which the ’958 Patent admits were “*well-known*” (Ex. 1001, 5:3-9); and it was also well-known that ACPI-compatible operating systems include schedulers and that ACPI-compatible processors, such as Itanium® processors, execute instructions scheduled by the operating system, as illustrated below. *See* Bigbee, FIG. 1, [0015]; Ex. 1023, 267-277; Schultz, [0001], FIG. 1 (reproduced below).

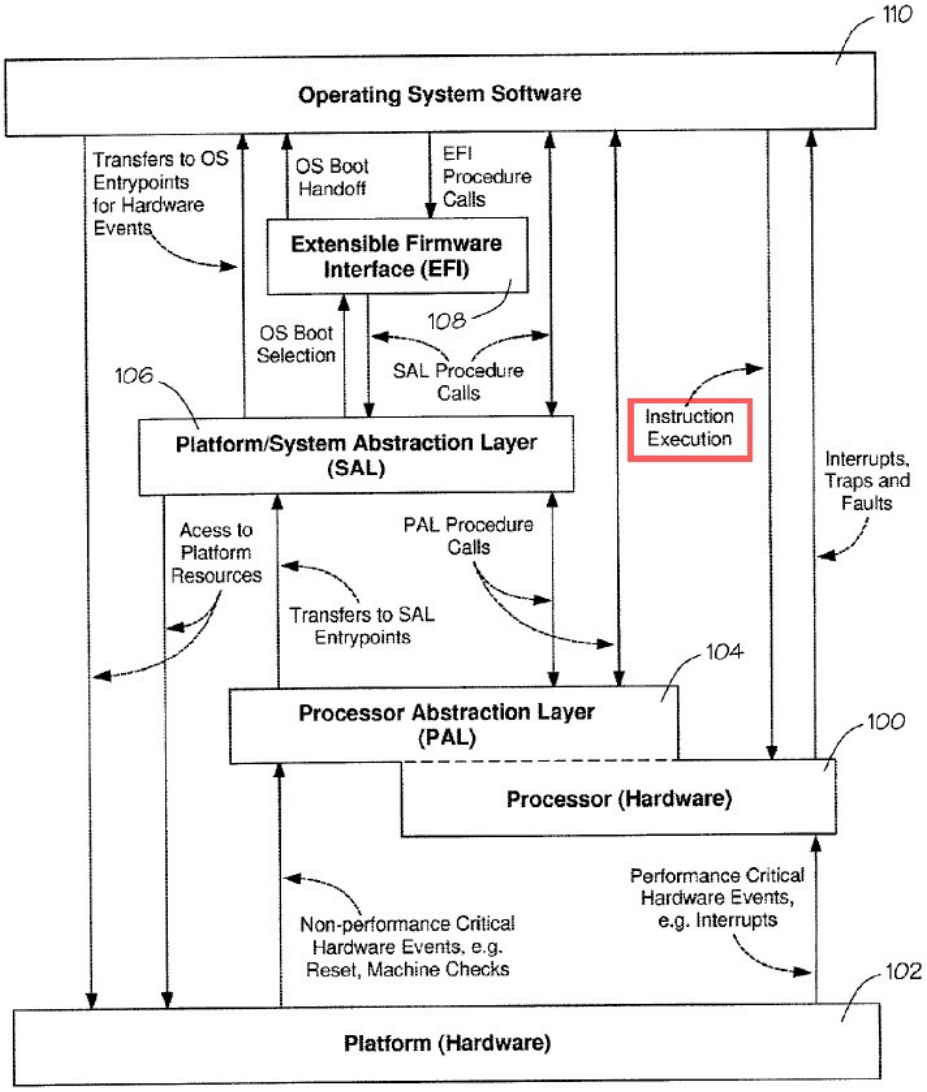


FIG. 1

Schultz, FIG. 1 (annotated).

130. Bigbee does not disclose the pair of processor cores includes a master processor and a slave processor.[5] Bigbee in view of Nguyen, however, renders obvious this organization of processors obvious.

131. Nguyen discloses a processor having a pair of lockstep processor cores (*see above* Section VII.B) and that one of the execution cores "may be designated as the master core and the other as the slave core" when the processor is operating in FRC mode ("lockstep"). Nguyen, [0027], [0047]. Nguyen also discloses a bit in a status register of each core may be used to indicate its status as master or slave, and the bit may be set when the system is booted or reset. *Id.*

132. A person of ordinary skill in the art would have found it obvious to designate one of Bigbee's lockstep pair of processor cores as the master processor and the other as the slave processor with a limited role of discrepancy checking. A person of ordinary skill in the art would have been motivated to designate a master processor and a slave processor in Bigbee's pair of processors because it would allow the single logical processor to output a single set of results by writing data

[5] Bigbee does not use the terms "master" and "slave" in his disclosure, opting instead for terms like "first" and "second" processor core. As such terminology is commonplace in the field of computer system design, a person of ordinary skill in the art would understand using these, and similar, terms to differentiate between processors.

from whichever of the processors (master or slave) is determined to be operating correctly when there is discrepancy between the processors. In such a case, the correctly operating processor would be designated as the master, and its results would be transferred to the shared resources while signals from the slave are dropped. Nguyen, [0047], [0053]; Kondo, [0086]-[0095]; Marshall, 1:11-32.

133. A person of ordinary skill in the art would have expected success in making such modification because it was conventional and well known to a person of ordinary skill in the art that 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, and that processors have status registers for recording such designations. Marshall, 1:11-16; Kondo, [0086]; Tu, [0031]. Designating Bigbee's pair of lockstepped processor cores as a master processor and a slave processor as taught in Nguyen would merely require the application of ordinary skill to generate predictable results. For example, the FRC logic typically includes a register that stores information about which processor has which designation, such as via status bit for each processor. Modifying Bigbee to add a status bit in the FRC logic register for each processor core to indicate whether it is designated as the master or the slave when the system is booted or reset as taught

in Nguyen would have required only the application of ordinary programming skill.

4. [13d]: "firmware operable, responsive to detection of loss of lockstep between the master and slave processors, to use an ACPI method to trigger the operating system to idle the master processor, attempt to recover lockstep between the master and slave processors, and"

134. In my opinion, Bigbee in view of Nguyen renders obvious [13d] for reasons discussed in [19c] and [13b]-[13c] and the additional reasons discussed below.

135. For reasons explained in [19c], a person of ordinary skill in the art would have understood that Bigbee's firmware is operable, responsive to detection of loss of lockstep between the processors, to trigger the operating system to idle the processors. And for reasons explained in [13b]-[13c], Bigbee in view of Nguyen renders obvious master and slave processors. For the reasons explained below, Bigbee in view of Nguyen renders obvious idling both the master and slave processors, and attempting to recover lockstep between the master and slave processors.

136. Bigbee's term "processor" refers to either a physical processing element or execution entity, or a logical processing element or execution entity as viewed by the operating system software. Bigbee, [0032]. Bigbee further discloses

virtually ejecting the processor from the operating system such that the processor's package, which includes both processor cores, is taken offline and to ensure that it is reset or "re-initialized." Bigbee, [0019], [0023]. In Bigbee combined with Nguyen, both the master and slave processors would have been idled in the way as taught by Bigbee. This is the same as the '958 Patent's idling the lockstep processor pair that includes both the master processor and the slave processor. Ex. 1001, 6:9-20.

137. The ACPI specification uses the term "method" to refer to certain functionality, and of course these are methods within the scope of the claim. Other functionality in the ACPI specification may also be a method under the plain and ordinary meaning of the term—even if not termed a "method." E.g., Ex. 1022, Table 5.43.

138. Bigbee further discloses the firmware uses ACPI methods to trigger the operating system to idle the processors. Bigbee discloses the firmware, via PAL (e.g., PALE_CHECK routine) and SAL (e.g., SAL_CHECK), generates a system control interrupt (SCI) to the operating system. Bigbee, [0021]-[0023]; *see above* Section IX.C.4. The operating system queries firmware (e.g., ACPI BIOS) to determine the source of the SCI using an _Lxx control method. Bigbee, [0035]-[0036], FIG. 4A (block 408). The firmware responds to the query by notifying the

operating system of a request for processor hot removal using an ACPI Notify command, and the operating system requests firmware ejection of the processor by executing an _EJx control method. *Id.*, [0036], FIG. 4A (blocks 410, 412).

139. A person of ordinary skill in the art would have understood that the _Lxx, _EJx, and ACPI Notify commands are ACPI methods as described in the ACPI 2.0 specification published over one year prior to the time of the alleged invention. Ex. 1022, 154-155, 174-175. A person of ordinary skill in the art would also have understood that these methods are executed from Bigbee's firmware so as to virtually eject the processor from the operating system ("use an ACPI method to trigger the operating system to idle the master processor"). Bigbee, [0036]-[0037], FIG. 4A (block 414). The '958 Patent similarly discloses the "firmware 15 utilizes an ACPI method 104 to 'eject' master processor 12A, thereby triggering OS 11 to idle the master processor 12A." Ex. 1001, 6:9-15.

140. For reasons explained in [19c], Bigbee's firmware is operable to attempt to recover lockstep between the pair of processors, and Bigbee combined with Nguyen's designation of a master and slave processor as explained above renders obvious "attempt[ing] to recover lockstep between the master and slave processors."

5. [13e]: "[firmware operable to] if recovery of lockstep is successful then use an ACPI method to reintroduce the master processor to the operating system."

141. In my opinion, Bigbee in view of Nguyen renders obvious [13e] for reasons explained in [19d] and the additional reasons discussed below.

142. Bigbee discloses that after resetting the processor and re-enabling FRC mode for the pair of processors ("if recovery of lockstep is successful"), the firmware generates an SCI to the operating system. Bigbee, [0037], FIG. 4B (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. Bigbee, [0037], FIG. 4B (blocks 416, 418). The operating system queries firmware to determine the status and to identify the associated resources of the processor to be inserted, and the firmware provides the operating system with the requested data by executing one or more control methods (e.g., _STA, _MAT, and _CRS). Bigbee, [0037], FIG. 4B (blocks 420, 422). Once the relevant status and resource data is received, the operating system requests insertion of the processor by calling an ACPI method (e.g., _PS0), which the firmware virtually executes to insert the processor into the operating system. Bigbee, [0037], FIG. 4B (blocks 424, 426). Once the processor has been inserted and/or activated, the operating system modifies its internal data

structures to indicate that the processor (including both processor cores) is online and available ("reintroduce the master processor to the operating system"). Bigbee, [0037], FIG. 4B (block 430).

143. A person of ordinary skill in the art would have known that the _STA, _MAT, _CRS, _PS0 control methods are ACPI methods as listed in the ACPI 2.0 specification. Ex. 1022, 158, 156, 154. A person of ordinary skill in the art would also have understood that these methods are executed from Bigbee's firmware when called by the operating system to insert/activate the processor such that the operating system knows the processor is online and available ("use an ACPI method to reintroduce the master processor to the operating system").

144. The '958 Patent discloses a similar process for reintroducing the processors to the operating system—after the processor pair is reset and lockstep is recovered, the firmware uses an ACPI method to trigger the operating system to "check for" the master processor such that the operating system recognizes the master processor as available, thereby reintroducing the master processor to the operating system. Ex. 1001, 6:55-7:12; 8:60-9:7.

F. Claim 2: "wherein said using firmware to trigger an operating system to idle the lockstep pair of processors further comprises: using the firmware to trigger the operating system to idle a master processor of the lockstep pair of processors."

145. Claim 2 depends from claim 1. In my opinion, Bigbee in view of Nguyen renders obvious claim 2 for reasons discussed in [1b], [19c], [13b], and the reasons discussed below. Bigbee in view of Nguyen discloses using firmware to trigger the operating system to idle the pair of processors, which would include a master processor and a slave processor. Bigbee, [0032], [0035]-[0037], FIG. 4A; *see above* Sections IX.D, IX.C.4, IX.E.3.

G. Claim 3: "wherein said using firmware to trigger the operating system to recognize the lockstep pair of processors having recovered lockstep further comprises: using the firmware to trigger the operating system to recognize the master of the lockstep pair of processors."

146. Claim 3 depends from claim 2. In my opinion, Bigbee in view of Nguyen renders obvious claim 3 for reasons discussed in [1b], [19d], [13b], and the additional reasons discussed below. Bigbee in view of Nguyen discloses using the firmware to trigger an operating system to recognize the pair of processors, which would include a master processor and a slave processor. *See above* Sections IX.D, IX.C.5, IX.E.3.

H. Claim 4: "wherein said detecting loss of lockstep comprises: detecting corrupt data in one of said lockstep pair of processors that would lead to a lockstep mismatch if acted upon."

147. Claim 4 depends from claim 1. In my opinion, Bigbee discloses or renders obvious claim 4.

148. In the Background section, the '958 Patent provides an example of the detection of data corruption as detecting a data cache error in one of the processors by a parity-based mechanism and/or ECC mechanism, and that corrupt data may be due to a cosmic event. Ex. 1001, 1:41-46, 2:29-37. The '958 Patent further describes that if a processor of the lockstep pair acts upon the corrupt data, the output of the two processors will not match, and a lockstep mismatch will occur because of the mismatch in the output of the two processors. Ex. 1001, 2:37-59. As admitted by the '958 Patent in the Background section, such detection of corrupt data in a processor of a lockstep pair of processors that would cause a lockstep mismatch was known.

149. Bigbee has similar disclosure to the '958 Patent and at least suggests this limitation. Bigbee discloses detecting errors within individual processor cores, the errors can be data ECC or parity errors occurring in a processor's cache, and such errors cause the processor pair to break lockstep. Bigbee, [0035], [0018]-[0019]. Bigbee also discloses that the FRC logic monitors the operations and

outputs of the processors to detect inconsistencies indicating a mismatch error, which can be caused by errors in the individual processors, such as an alpha particle impacting on one or more of the processors. Bigbee, [0033].

150. A person of ordinary skill in the art would have understood Bigbee's detecting errors within an individual processor core to be, or at least to suggest, detecting corrupt data in one of the lockstep pair of processors that would lead to a lockstep mismatch error if processors continued to use the corrupt data in processor operations or attempted to write to the shared cache ("acted upon") to detect inconsistencies indicating a mismatch error ("lead to a lockstep mismatch"). *See* Bigbee, [0033]. This is because a person of ordinary skill in the art would have understood that detecting errors within an individual processor, rather than a mismatch in the outputs of the two lockstep processors, often means an impending mismatch or loss of lockstep. Safford, 3:5-34 (error detected in an single processor core of a lockstep processor signals an *impending* loss of lockstep); *see also* Kondo, [0101], [0103], [0106]; Nguyen, [0008], [0022]; *see above* Section V.A.

151. One example of a person of ordinary skill in the art's understanding that an error in an individual processor (as disclosed by Bigbee) is an error that would lead to a lockstep mismatch down the line is evidenced by Nguyen. Nguyen discloses detecting data corruption by detecting soft errors in the data in storage

arrays of the processors via ECC or parity error detection logic for the arrays (e.g., caches). Nguyen, [0008], [0020]-[0021]. Nguyen further discloses corrupt data from one execution core and an uncorrupted version of the data from the other execution can reach and be compared by the FRC checker ("acted upon"), and if the data does not match ("lead to a lockstep mismatch"), the FRC checker sets a status flag to indicate the mismatch and triggers an FRC error. Nguyen, [0022], [0056]-[0057].

I. Claim 5: "wherein said detecting loss of lockstep comprises: detecting a precursor to lockstep mismatch."

152. Claim 5 depends from claim 1. In my opinion, Bigbee discloses or renders obvious claim 5.

153. The '958 patent describes a "precursor to lockstep" as an error occurring "before an actual lockstep mismatch has occurred." Ex. 1001, 4:60-64. It explains that "once an error (e.g., corrupt data) is detected by error detect logic 13A or 13B [(in one of the two lockstepped processors]), such error may eventually propagate to a lockstep mismatch error that is detectable" as a mismatch in the output of the two lockstepped processors. Ex. 1001, 4:65-67.

154. As explained for claim 4, Bigbee discloses or at least suggests detecting errors in individual processors, such as those that would lead to lockstep mismatch if the error propagated to the compare logic. *See above* Section IX.H.

155. It was well known to a person of ordinary skill in the art that an error detected in an individual processor of the lockstep processor pair is a precursor to lockstep mismatch because it signals an impending or imminent loss of lockstep. *See above* Sections V.A, IX.H; Safford, 3:5-34, Kondo, [0101], [0103], [0106].

J. Claim 6: "wherein said detecting loss of lockstep comprises: detecting lockstep mismatch between the lockstep pair of processors."

156. Claim 6 depends from claim 1. In my opinion, Bigbee discloses claim 6.

157. As explained in [19a], Bigbee's "detection of loss of lockstep between the pair of lockstep processors" includes the FRC logic's detection of inconsistencies in the results of the pair of processors that indicate a mismatch error ("detecting lockstep mismatch between the lockstep pair of processors"). Bigbee, [0019], [0033]; *see above* Section IX.C.2.

158. The '958 Patent also similarly discloses an error detect logic compares the outputs of the master processor and the slave processors and detects a lockstep mismatch if the outputs fail to match. Ex. 1001, 2:16-34; 4:46-55.

K. Claim 7: "if determined that a lockstep mismatch has not occurred between the lockstep pair of processors, then determining that the loss of lockstep is recoverable."

159. Claim 7 depends from claim 1. In my opinion, Bigbee in view of Nguyen renders obvious claim 7.

160. As discussed in [19a], Bigbee discloses detection of errors within the processor cores of a processor package that cause a break in the lockstep between the cores (Bigbee, [0019], [0033]), and Nguyen discloses determining that lockstep is recoverable in response to detection of an error in the processor cores before a mismatch is detected ("if determined that a lockstep mismatch has not occurred"). *See above* Sections VII.B, IX.C.2; Nguyen, [0023].

L. Claim 8: "if determined that a lockstep mismatch has occurred between the lockstep pair of processors, then determining that the loss of lockstep is not recoverable."

161. Claim 8 depends from claim 1. In my opinion, Bigbee in view of Nguyen renders obvious claim 8.

162. As discussed in [19a], Bigbee's FRC logic detects inconsistencies in the results of the processor pair that indicate an FRC mismatch error, and Nguyen discloses determining that lockstep is not recoverable from an FRC error indicating mismatch ("if determined that a lockstep mismatch has occurred"). *See above* Sections VII.B, IX.C.2; Bigbee, [0033]; Nguyen, [0007], [0022]-[0023]. Nguyen discloses, for example, the FRC checker signals an FRC error after detecting a mismatch ("if determined that a lockstep mismatch has occurred between the lockstep pair of processors") before the error detector detects the underlying soft error and resets the system. *Id.*, [0022]. Also, the FRC checker signals an FRC

error and triggers resetting the system if the error detector does not detect an error within a timeout interval after the mismatch detection ("if determined that a lockstep mismatch has occurred between the lockstep pair of processors"). *Id.*, [0056]-[0058]. In Bigbee combined with Nguyen, Bigbee's FRC logic would have determined that the FRC mismatch error for which it was not known which processor core was responsible as not recoverable. *See above* Sections V.B; Bigbee, [0033].

M. Claim 14: "error detection logic for detecting loss of lockstep between the master and slave processors."

163. Claim 14 depends from claim 13. In my opinion, Bigbee in view of Nguyen renders obvious claim 14 for reasons discussed in [19a], [13b] and [13c], and below.

164. As discussed in [19a], Bigbee discloses error detection logic for detecting loss of lockstep between a lockstep pair of processors, and as discussed in [13b]-[13c], Bigbee and Nguyen discloses the lockstep pair includes a master and a slave processors. *See above* Sections IX.E.3, IX.C.2, VII.A-VII.B. Thus, Bigbee combined with Nguyen teaches error detection logic for detecting loss of lockstep between the master and slave processors.

N. Claim 15: "error detection logic for detecting a precursor to loss of lockstep."

165. Claim 15 depends from claim 13. In my opinion, Bigbee in view of Nguyen renders obvious claim 15 for reasons discussed in [19a], claim 5, and below.

166. For reasons discussed in [19a], Bigbee discloses error detection logic for detecting loss of lockstep between a lockstep pair of processors; and for reasons discussed in claim 5, such detection of loss of lockstep includes detecting a precursor to lockstep mismatch. *See above* Sections IX.C.2, IX.I.

O. Claim 16: "wherein said precursor to loss of lockstep is treated as said detection of loss of lockstep."

167. Claim 16 depends from claim 15. In my opinion, Bigbee in view of Nguyen renders obvious claim 16. For reasons discussed in claim 5, Bigbee discloses detecting errors before detecting a mismatch in the output of the two lockstep processors (a "precursor to lockstep mismatch"). *See above* Sections IX.I, IV.C.1. Bigbee discloses treating such precursor to lockstep mismatch as detection of loss of lockstep because Bigbee discloses handling such errors through a recovery process that recovers lockstep of the processor pair. Bigbee, [0024], [0037].

P. Claim 17: "wherein said error detection logic comprises a parity-based mechanism."

168. Claim 17 depends from claim 15. Bigbee discloses or renders obvious claim 17.

169. The '958 Patent admits that its error detection logic includes "known parity-based mechanisms and ECC mechanisms." Ex. 1001, 4:44-46. Its Background section describes that "[p]arity-based mechanisms are often employed in processors" and "Parity/ECC mechanisms have been employed extensively for caches, memories, and similar data storage arrays," for detecting data errors in processors. Ex. 1001, 1:57-2:6. Likewise, Bigbee discloses detecting errors in the individual processors ("precursor to loss of lockstep") via well-known parity or ECC error detection logic, as evidenced by Nguyen. Bigbee, [0018]-[0019]; Nguyen, [0008], [0020]-[0021], [0039]; *see above* Section IX.N.

Q. Claim 18: "further comprising error detection logic for detecting lockstep mismatch between the master and slave processors."

170. Claim 18 depends from claim 13. Bigbee in view of Nguyen renders obvious claim 18.

171. For reasons discussed in [19a] and claim 6, Bigbee's FRC logic detects lockstep mismatch between a pair of lockstep processors. *See above* Sections IX.C.2, IX.J. And for reasons discussed in [13b]-[13c], Bigbee in view of

Nguyen discloses the pair of lockstep processors include a master processor and a slave processor. *See above* Section IX.E.3. A person of ordinary skill in the art would have understood that in Bigbee combined with Nguyen, Bigbee's FRC would detect lockstep mismatch between the master and slave processors.

R. Claim 20: "wherein the code for attempting to recover lockstep further comprises: code for resetting the pair of processors.

172. Claim 20 depends from claim 19. In my opinion, Bigbee discloses claim 20.

173. As discussed in [19c], Bigbee discloses "firmware code … for attempting to recover lockstep between the pair of processors." *See above* Section IX.C.4. As shown below, Bigbee's "firmware code … for attempting" to recover lockstep includes resets the processor ("code for resetting the pair of processors"). Bigbee, [0037], FIG. 4A (block 414).

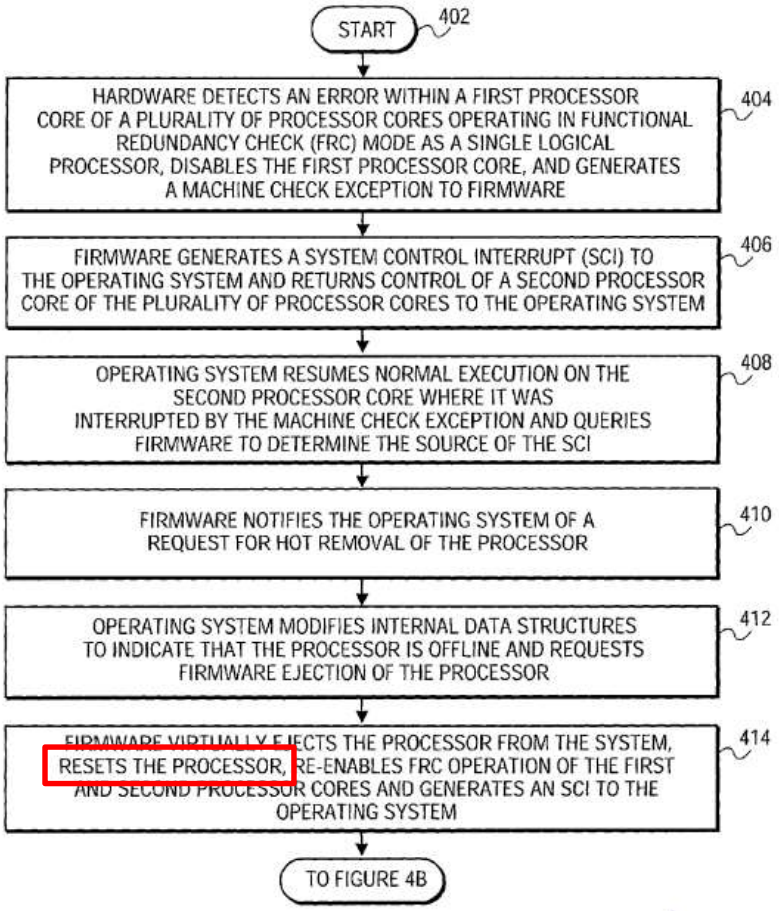


FIG. 4A

Bigbee, FIG. 4A (annotated).

S. Claim 21

174. Claim 21 depends from claim 19. In my opinion, Bigbee discloses claim 21.

1. [21a]: "wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system,"

175. For reasons discussed in [13a], Bigbee discloses its operating system is an ACPI-compatible operating system. *See above* Section IX.E.2.

2. [22b]: "said code for triggering the operating system to idle the processors comprises: code for using an ACPI method to idle the processors."

176. For reasons discussed in [19preamble], Bigbee discloses firmware code, and for reasons discussed in [19c] and [13d], Bigbee discloses firmware code for using an ACPI method to trigger the operating system to idle the processors. *See above* Sections IX.C.1, IX.C.4, IX.E.4.

T. Claim 22

177. Claim 22 depends from claim 19. In my opinion, Bigbee discloses claim 22.

1. [22a]: "wherein said operating system is an Advanced Configuration and Power Interface (ACPI)-compatible operating system,"

178. For reasons discussed in [13a], Bigbee discloses its operating system is an ACPI-compatible operating system. *See above* Section IX.E.2.

2. [22b]: "said code for triggering said operating system to recognize the processors as being available for receiving instructions comprises: code for using an ACPI method to cause the operating system to check for the processors."

179. For reasons discussed in [19preamble], Bigbee discloses firmware code; for reasons additionally discussed in [19d], Bigbee discloses firmware code for using an ACPI method to recognize the processors as being available for receiving instructions. *See above* Sections IX.C.1, IX.C.5. And during this process, Bigbee discloses the firmware 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) ("code for using an ACPI method"). Bigbee, [0037], FIG. 4B (block 422). Once the status and resource data is received, the operating system then requests insertion of the processor by calling an ACPI method (e.g., _PS0) ("to cause the operating system to check for the processors"). 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. Bigbee, [0037], FIG. 4B (block 430).

180. The '958 Patent similarly discloses a similar process—the firmware uses an ACPI method to trigger the operating system to "check for" the master processor, and as a result, the operating system recognizes the master processor as available. Ex. 1001, 6:55-7:12; 8:60-9:7.

X. Conclusion

181. For the reasons set forth above, in my opinion, a person of ordinary skill in the art as described in paragraph 29 would have considered the Challenged Claims of the '958 Patent to be obvious over the prior art discussed above.

182. 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 this case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

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

Dated: September 8, 2023 By: 

Michael C. Brogioli, Ph.D.