

succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy taught this claim limitation.).

181. *[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor* (I agree that, even if Kalyanasundharam did not explicitly teach selecting the processor coupled to the common communication bus as the spare processor, Kalyanasundharam as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, I agree that Kennedy taught selecting one or more alternate (*e.g.*, spare) boot processors by installing those alternate boot processors into specific slots corresponding to that alternate boot role on the backplane bus. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to improve the reliability of Kalyanasundharam's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, Kalyanasundharam as modified by Kennedy taught this claim limitation.).

182. **Claim 13:** *The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare* (I agree that Kalyanasundharam taught identifying a spare processor and holding the spare processor in reserve as a hot spare for at least the same reasons provided above in connection with claim 1. I also agree that, even if Kalyanasundharam did not explicitly teach doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors...may be used by the operating system." *Kennedy* (Ex. 1005) at 9:1-5. "[T]he CPU1 120 controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "The **alternate boot processors...**can execute the boot code in the central **boot ROM 150**". *Id.* at 10:42-45. I agree that the BIOS was a very well-known example of firmware.

In view of the above citations and this common knowledge, I agree that a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to achieve some cost savings by performing these tasks with firmware instead of using additional hardware. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA.).

183. **Claim 14:** *The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare* (I agree that Kalyanasundharam taught booting the spare processor as an active spare for at least the same reasons provided above in connection with claim 6. I also agree that, even if Kalyanasundharam did not explicitly teach doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors...**may be used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors...can execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. I agree that the BIOS was a very well-known example of firmware. In view of the above citations and this common knowledge, I agree that a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to enable Kalyanasundharam's system to **dynamically modify the activity of spare processors** (*e.g.*, between active spare and hot spare) to increase efficiency and/or throughput. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to

succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA.).

184. **Claim 15:** *The method of claim 9 wherein system firmware performs said changing operation* (I agree that Kalyanasundharam taught performing the changing operation for at least the same reasons provided above in connection with claim 9. I agree that, even if Kalyanasundharam did not explicitly teach doing so via system firmware, Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA. For example, Kennedy teaches that each CPU includes a slot selection (sometimes referred to as “SS”) logic for selecting which CPU is designated as the boot processor. “In general, the slot select logic 148 on the default boot CPU allows this CPU to boot.” *Kennedy* (Ex. 1005) at 10:29-31, FIG. 3. I agree that Kennedy’s **input/output services module (“IOSM”)** is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs’ slot selection logic. *See id.* at 6:36-38, FIG. 3. For example, “[a]ny other slot selection circuits on CPUs installed in the system have a corresponding slot selection line from the **IOSM 108.**” *Id.* As demonstrated by FIG. 3 of Kennedy reproduced below, Kennedy’s IOSM includes the **BIOS**.

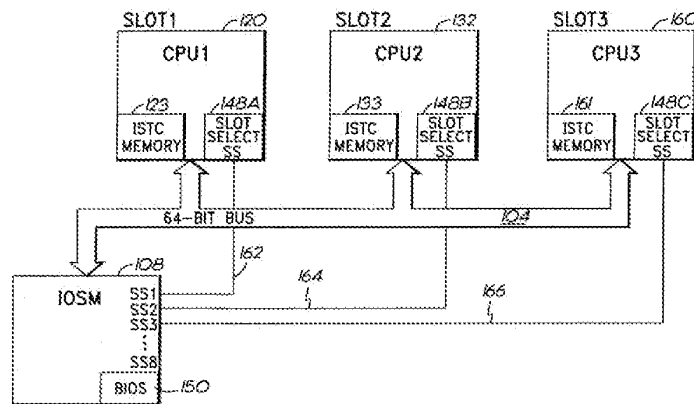


FIG. 3

I agree that the BIOS was a very well-known example of firmware. In view of the above citations and this common knowledge, I agree that a POSITA would have understood that that Kennedy’s BIOS, which constitutes firmware, is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs’ slot selection logic. In my opinion, a POSITA would have been motivated to combine these teachings from Kalyanasundharam and Kennedy to achieve some cost savings by performing the change operation with firmware instead of using

additional hardware. In my opinion, it was well within the skill of a POSITA to modify Kalyanasundharam's system by Kennedy in this manner, and thus this combination of Kalyanasundharam and Kennedy would have been expected to succeed. Accordingly, I agree that Kalyanasundharam as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA.).

2. Ground 2: Safford in view of Kennedy

185. I agree that Safford in view of Kennedy renders claims 1-6 and 9-15 obvious to a POSITA for the reasons set forth in the Request. I identify evidence supporting my opinions regarding the obviousness of certain claims and/or limitations below. It is worth noting that certain claim limitations analyzed for obviousness in the Request are omitted from the claim set below for the purpose of brevity. In my opinion, the Request's obviousness analysis of those claim limitations is sufficient and does not require my support to demonstrate a POSITA's understanding of the prior art relative to those claim limitations.

186. My opinions should be evaluated in view of the corresponding motivations to combine discussed above and the citations referenced in the Request. I have also reviewed and assisted in the preparation of Appendix B, which provides a detailed mapping of Safford and Kennedy to claims 1-6 and 9-15 that accurately reflects my opinions regarding obviousness.

187. **Claim 1: [1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors** (I agree that Safford described assigning the roles of master and checker to certain processors. *See Safford* (Ex. 1004) at 2:55-57. For at least the reasons provided in this Declaration above and in the Request, I agree that a POSITA would have understood Safford as disclosing a boot processor and failing over the same to a hot spare processor. I agree that, even if Safford did not explicitly identify which processor was the boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In my opinion, Safford taught data storage that stores the architected states, such as role assignments (*e.g.*, master/checker), of paired processors, and Kennedy taught that one such assignment could be the role of boot processor and that data storage holds configuration information (*e.g.*, boot processor assignment). I agree that a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford's

system by storing information that assigns the roles of boot processor and spare processor in case of a reboot. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors.).

188. *[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor* (I agree that the '781 Patent notes that a firmware component known as a processor abstraction layer (PAL) "copies all the needed information from memory into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor." Ex. 1001 at 17:57-61. I agree that Safford disclosed this limitation through an identical process for transferring the role of boot processor to the spare processor. *See Safford* (Ex. 1004) at 32:29-32. For example, Safford taught "moving an architected state of the processor unit experiencing the loss of lock step to the spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system." *Id.* I agree that, even if Safford did not explicitly identify the failed processor as having been the boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In view of the above citations, I agree that Safford taught logic that causes its system to replace a failed processor with a hot spare processor in response to a loss of lockstep, and Kennedy taught logic that causes its system to fail over the boot processor to an alternate processor. I agree that a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy taught logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *Id.*).

189. **Claim 2:** *The system of claim 1 wherein said data storage is non-volatile data storage* ("Finally, the architected states of each of the microprocessors 22 and 24 is copied from

the **memory 25** into the microprocessors 22 and 24, respectively.” *Safford* (Ex. 1004) at 2:2-4. “For example, the node controller 130 may store the architected state of the processors 111, 113, 121, 123, 125, and 127, either internally in the **node controller 130**, or in **another component** of the computer system 100.” *Id.* at 4:6-10. In my opinion, it was well known in the art that the term “memory” broadly refers to both volatile and non-volatile data storage. Indeed, I understand that non-volatile memory, including battery-backed SRAM (sometimes simply called CMOS) and Flash memory were commonly used at the time to store critical information. For at least this reason, I agree that a POSITA would have understood memory 25 of Safford as including non-volatile data storage. Accordingly, I agree that Safford as modified by Kennedy would have rendered the subject matter of claim 2 obvious to a POSITA.).

190. **Claim 3:** *The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor* (I agree that, even if Safford did not explicitly teach a user interface that receives input specifying the processor assigned the role of boot processor, Safford as modified by Kennedy would have rendered the subject matter of claim 3 obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, “**other processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art.” *Id.* In my opinion, it was well known in the art that the act of selecting in computer systems is often performed via user interfaces. Based on this common knowledge in the art, I agree that a POSITA would have understood Kennedy’s teaching about selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. In my opinion, a POSITA would have been motivated to modify Safford by Kennedy and this common knowledge in the art to enable a user to select which processor is to boot the system via a user interface. In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA.).

191. **Claim 4:** *The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor* (I agree that, even if Safford did not explicitly teach a user interface that receives

input specifying the processor assigned the role of spare processor, Safford as modified by Kennedy would have rendered the subject matter obvious to a POSITA. For example, Kennedy taught selecting which processor among multiple processors is to boot the system. *See Kennedy* (Ex. 1005) at 4:33-34. Specifically, “other **processor** types may be **selected to boot the system** with a corresponding change in the boot ROM 150, as well known in the art.” *Id.* at 4:33-34. It was well known in the art that the act of selecting in computer systems is often performed via user interfaces. Based on this common knowledge in the art, I agree that a POSITA would have understood Kennedy’s teaching about selecting the processor to boot the system as including a spare processor selection (*e.g.*, one of the alternate processors not selected as the boot processor) made by input received via a user interface. In my opinion, a POSITA would have been motivated to modify Safford by Kennedy and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA. *Id.*).

192. **Claim 5:** *The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor* (“To improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a **master/check pair.**” *Safford* (Ex. 1004) at 2:55-57. “**The processors** operate as pairs when in lock step (*i.e.*, the processors 111 and 113 are a pair; the processors 121 and 123 are a second pair ; and the processors 125 and 127 are a third pair). From the node controller’s perspective, **each pair of processors** appears as a **single (logical) processor.**” *Id.* at 3:57-63. In view of these citations, I agree that a POSITA would have understood the processors disclosed in Safford to constitute master and slave processors. Accordingly, I agree that Safford as modified by Kennedy taught a plurality of processor modules that each include a master processor and a slave processor.).

193. **Claim 6:** *The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor* (“In block 210, the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step** and that the first processor pair 111/113 should be taken ‘off-line.’” *Safford* (Ex. 1004) at 4:37-40. “In block 230,

the node controller 130 **‘reboots’ the processors 111 and 113**, and the processors 111 and 113 become the **new ‘hot standby’ processor pair** on the system 100.” *Id.* at 4:57-60. Safford also taught that “the spare processor unit becomes an active processor unit in the computer system” *Id.*, claim 1. In view of the above citations, I agree that a POSITA would have understood Safford as teaching that a spare processor can be active before a role transfer. Moreover, in my opinion, a POSITA would have understood that the role at issue could be the boot processor assignment in view of Safford as modified by Kennedy for at least the same reasons provided in the Request and in connection with claim 1. I also agree that a POSITA would have understood the term “good” processor to refer to an active spare processor. In my opinion, a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford’s system by providing high availability through active and/or inactive spare processors. *Id.* In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, I agree that Safford as modified by Kennedy taught that the spare processor is an active spare before being transferred the role of boot processor.).

194. **Claim 9: [9.1] assigning a role of boot processor to one of a plurality of processors in a system** (I agree that Safford described assigning the roles of master and checker to certain processors. *See Safford* (Ex. 1004) at 2:55-57. For at least the reasons provided in this Declaration above and in the Request, I agree that a POSITA would have understood Safford as disclosing a boot processor and failing over the same to a hot spare processor. I agree that, even if Safford did not explicitly identify which processor was the boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13. In my opinion, Safford taught assigning roles (*e.g.*, master/checker) to paired processors, and Kennedy taught that one such assignment could be the role of boot processor. I agree that a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford’s system by assigning the role of boot processor to a certain processor in case of a reboot. In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy taught assigning a role of boot processor to one of a plurality of processors in a system.).

195. *[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor* (I agree that Safford described replacing a failed processor with a hot spare processor in response to a loss of lockstep. *Safford* (Ex. 1004) at 3:27-30, 4:37-54. I also agree that the '781 Patent notes that a firmware component known as a processor abstraction layer (PAL) “copies all the needed information from memory into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor.” Ex. 1001 at 17:57-61. I agree that Safford disclosed this claim limitation through an identical process for transferring the role of boot processor to the spare processor. *Safford* (Ex. 1004) at 32:29-32. I further agree that, even if Safford did not explicitly identify the failed processor as having been the boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. *See, e.g., Kennedy* (Ex. 1005) at 7:5-13). In my opinion, Safford taught replacing a failed processor with a hot spare processor in response to a loss of lockstep, and Kennedy taught failing over the boot processor to an alternate processor. I agree that a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford’s system by providing high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy taught changing, during system runtime, the assigned role of boot processor to the spare processor in response to loss of lockstep for the boot processor.).

196. *[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system* (I agree that, even if Safford did not explicitly teach an operating system that recognizes the spare processor as the boot processor without requiring shutdown, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, “a **central system boot ROM 150** accessible by all processors...may be used by the **operating system.**” *Kennedy* (Ex. 1005) at 9:1-5. “[T]he CPU1 120 controls the system again and begins the **operating system boot process.**” *Id.* at 9:53-55. “The **alternate boot processors**...can execute the **boot code in the central boot ROM 150**”. *Id.* at 10:42-45. In my opinion, a POSITA would have understood that,

because Kennedy's central boot ROM 150 is used by the operating system and stores the boot code executed in conjunction with the operating system to boot the system, the operating system would recognize the spare processor as the boot processor after failover without requiring shutdown of the operating system, especially if Kennedy is to achieve its objective of providing fault tolerance and high availability. *See Kennedy* (Ex. 1005) at 9:1-5, 9:53-55, 10:42-45. I believe that this conclusion of a POSITA's understanding is also supported by Safford's objective of avoiding system crashes. *See Safford* (Ex. 1004) at 3:21-22, 1:45-47. I agree that a POSITA would have been motivated to combine these teachings from Safford and Kennedy to improve the reliability of Safford's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy would have rendered obvious to a POSITA an operating system that recognizes the spare processor as the boot processor without requiring shutdown.).

197. **Claim 12:** *The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor* (I agree that, even if Safford did not explicitly teach determining a processor that is coupled to a common communication bus with the boot processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "the system has a centrally accessible memory, a default boot central processing unit installed in a first slot on the backplane bus, and may also have one or more alternate boot central processing units installed in a second slot on the backplane bus." *Kennedy* (Ex. 1005) at 2:34-38. In view of these citations, I agree that Kennedy taught that the boot processor and the alternate boot processors are installed in specific slots along the same backplane bus. Based on this teaching from Kennedy, I agree that a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. As a result, I agree that any form of identifying such an alternate processor would constitute a determination of an alternate processor coupled to a common communication bus with the boot processor. In my opinion, a POSITA would have been motivated to combine these teachings from Kennedy with Safford to improve the communication capabilities of Safford's system for fault tolerance and high

availability. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. *Id.* Accordingly, I agree that Safford as modified by Kennedy taught this claim limitation.).

198. *[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor* (I agree that, even if Safford did not explicitly teach selecting the processor coupled to the common communication bus as the spare processor, Safford as modified by Kennedy would have rendered this claim limitation obvious to a POSITA. For example, according to Kennedy, "the system has a centrally accessible memory, a default boot central processing unit installed in a first slot on the backplane bus, and may also have one or more alternate boot central processing units installed in a second slot on the backplane bus." *Kennedy* (Ex. 1005) at 2:34-38. In my opinion, a POSITA would have been motivated to combine these teachings from Kennedy with Safford to improve the reliability of Safford's system by providing fault tolerance and high availability in the event that the boot processor fails before a reboot occurs. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, Safford as modified by Kennedy taught this claim limitation.).

199. **Claim 13:** *The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare* (I agree that, even if Safford did not explicitly teach doing so via system firmware, Safford as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the basic input/output system (BIOS) instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 microprocessor in order to boot the system." *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system boot ROM 150 accessible by all processors...may be used by the operating system." *Id.* at 9:1-5. "[T]he CPU1 120 controls the system again and begins the operating system boot process." *Id.* at 9:53-55. "The alternate boot processors...can execute the boot code in the central boot ROM 150". *Id.* at 10:42-45. I agree that the BIOS was a very well-known example of firmware. In view of the above citations and this

common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. In my opinion, a POSITA would have been motivated to combine these teachings from Safford and Kennedy to achieve some cost savings by performing these tasks with firmware instead of using additional hardware. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy would have rendered the subject matter of claim 13 obvious to a POSITA.).

200. **Claim 14:** *The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare* (I agree that, even if Safford did not explicitly teach doing so via system firmware, Safford as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA. For example, according to Kennedy, "central boot ROM 150 comprises a read-only-memory with the **basic input/output system (BIOS)** instruction set for an INTEL 80486 or 80386 microprocessor. Accordingly, in the present embodiment, at least one CPU connected to the 64-bit bus is, or emulates, an INTEL 80486 or 80386 **microprocessor** in order to **boot the system.**" *Kennedy* (Ex. 1005) at 4:25-33. "[A] central system **boot ROM 150** accessible by all processors...**may be used by the operating system.**" *Kennedy* (Ex. 1005) at 9:1-5. "[T]he **CPU1 120** controls the system again and begins the **operating system boot process.**" *Id.* at 9:53-55. "**The alternate boot processors...can execute the boot code** in the central **boot ROM 150**". *Id.* at 10:42-45. I agree with the BIOS was a very well-known example of firmware. In view of the above citations and this common knowledge, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. In my opinion, a POSITA would have been motivated to combine these teachings from Safford and Kennedy to enable Safford's system to dynamically modify the activity of spare processors (*e.g.*, between active spare and hot spare) to increase efficiency and/or throughput. In my opinion, it was well within the skill of a POSITA to modify Safford's system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy would have rendered the subject matter of claim 14 obvious to a POSITA.).

201. **Claim 15:** *The method of claim 9 wherein system firmware performs said changing operation* (Safford taught performing the changing operation for at least the same

reasons provided above in connection with claim 9. I agree that, even if Safford did not explicitly teach doing so via system firmware, Safford as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA. For example, Kennedy teaches that each CPU includes a slot selection (sometimes referred to as “SS”) logic for selecting which CPU is designated as the boot processor. “In general, the slot select logic 148 on the default boot CPU allows this CPU to boot.” *Kennedy* (Ex. 1005) at 10:29-31, FIG. 3. I agree that Kennedy’s **input/output services module (“IOSM”)** is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs’ slot selection logic. *See id.* at 6:36-38, FIG. 3. For example, “[a]ny other slot selection circuits on CPUs installed in the system have a corresponding slot selection line from the **IOSM 108.**” *Id.* As demonstrated by FIG. 3 of Kennedy reproduced below, Kennedy’s IOSM includes the BIOS.

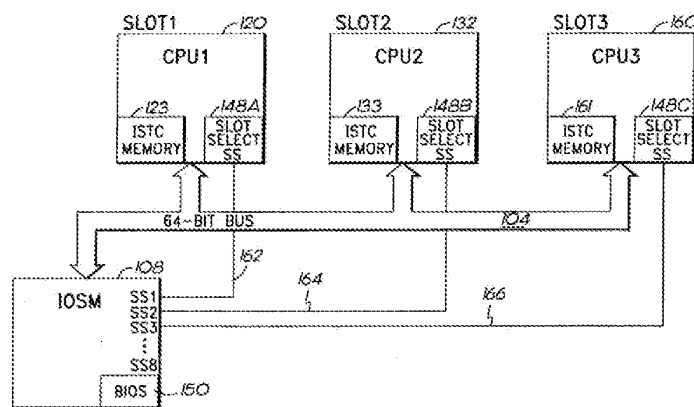


FIG. 3

I agree that the BIOS was a very well-known example of firmware. In view of the above citations and this common knowledge, I agree that a POSITA would have understood that that Kennedy’s BIOS, which constitutes firmware, is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs’ slot selection logic. In my opinion, a POSITA would have been motivated to combine these teachings from Safford and Kennedy to achieve some cost savings by performing the change operation with firmware instead of using additional hardware. In my opinion, it was well within the skill of a POSITA to modify Safford’s system by Kennedy in this manner, and thus this combination of Safford and Kennedy would have been expected to succeed. Accordingly, I agree that Safford as modified by Kennedy would have rendered the subject matter of claim 15 obvious to a POSITA.).

3. **Ground 3: Kennedy in view of Nguyen and Fox**

202. I agree that Kennedy in view of Nguyen and Fox renders claims 1-30 obvious to a POSITA for the reasons set forth in the Request. I identify evidence supporting my opinions regarding the obviousness of certain claims and/or limitations below. It is worth noting that certain claim limitations analyzed for obviousness in the Request are omitted from the claim set below for the purpose of brevity. In my opinion, the Request's obviousness analysis of those claim limitations is sufficient and does not require my support to demonstrate a POSITA's understanding of the prior art relative to those claim limitations.

203. My opinions should be evaluated in view of the corresponding motivations to combine discussed above and the citations referenced in the Request. I have also reviewed and assisted in the preparation of Appendix C, which provides a detailed mapping of Kennedy, Nguyen, and Fox to claims 1-30 that accurately reflects my opinions regarding obviousness.

204. **Claim 1: [1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors** (I agree that Kennedy taught memories that store CPU configuration information that includes the roles of boot processor and alternate processor in the system. *See, e.g., Kennedy* (Ex. 1005) at 7:9-13, 2:17-21, 9:16-21, 7:33-35. I also agree that Kennedy taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors.).

205. **[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor** (I agree that Nguyen described detecting a loss of lockstep. *See, e.g., Nguyen* (Ex. 1008) ¶ [0008], [0024], [0054]-[0055]. I also agree that Kennedy taught logic that causes its system to replace the boot processor with an alternate processor in response to a failure of the boot processor, and Nguyen taught detecting an error (*i.e.*, a loss of lockstep). *See, e.g., Kennedy* (Ex. 1005) at 7:9-13; *see also Nguyen* (Ex. 1008) ¶ [0008], [0024], [0054]-[0055]. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the reliability of Kennedy's system by providing fault tolerance and high availability in the event that the boot processor fails through a loss of lockstep. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed. Accordingly, I agree that Kennedy

as modified by Nguyen taught logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor.).

206. **[1.4] wherein said another of said plurality of processors that is assigned said role of spare processor is held in reserve as a hot spare processor until said loss of lockstep is detected for said boot processor** (I agree that Kennedy as modified by Nguyen and Fox disclosed, or at least rendered obvious, transferring the role of boot processor to a hot spare processor during system runtime in response to detecting loss of lockstep for the boot processor. *See, e.g., Fox* (Ex. 1006), Abstract. I also agree that a POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to decrease delays in failing over from the boot processor to the spare processor. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught that the processor assigned the role of spare is held in reserve as a hot spare processor until the loss of lockstep is detected for the boot processor.).

207. **Claim 2: The system of claim 1 wherein said data storage is non-volatile data storage** (Kennedy taught that "[t]he CPU1 120 has a corresponding non-volatile memory 123 which contains configuration information and CPU specific initialize and self test code (ISTC) for the CPU1 120." *Kennedy* (Ex. 1005), 5:15-18. I agree that, based on Kennedy's disclosure, a POSITA would have understood that the data storage that stores the information assigning the roles of boot processor and spare processor included non-volatile memory. Accordingly, I agree that Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 2 obvious to a POSITA.).

208. **Claim 3: The system of claim 1 further comprising: a user interface operable to receive input specifying said one of said plurality of processors to be assigned said role of boot processor** (I agree that it was well known in the art that the act of selecting processor roles in computer systems (*e.g.*, master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. I also agree that, based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure of selecting the processor to boot the system as including a boot processor selection made by input received via a user interface. In my opinion, a POSITA would have been motivated to modify Kennedy by Nguyen, Fox, and this common knowledge in the art to enable a user to

select which processor is to boot the system via a user interface. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen, Fox, and the common knowledge in this manner, and thus this combination of Kennedy, Nguyen, Fox, and the common knowledge would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen, Fox, and this common knowledge in the art would have rendered the subject matter of claim 3 obvious to a POSITA.).

209. **Claim 4:** *The system of claim 1 further comprising: a user interface operable to receive input specifying said another of said plurality of processors to be assigned said role of spare processor* (I agree that it was well known in the art that the act of selecting processor roles in computer systems (e.g., master/slave designations, distinct duty assignments or permissions in asymmetric multiprocessor configurations, etc.) is often performed via user interfaces. I also agree that, based on this common knowledge in the art, a POSITA would have understood Kennedy's disclosure of selecting the processor to boot the system as including a spare processor selection (e.g., one of the alternate processors not selected as the boot processor) made by input received via a user interface. In my opinion, a POSITA would have been motivated to modify Kennedy by Nguyen, Fox, and this common knowledge in the art to enable a user to select, via a user interface, which processor is to serve as a spare to the boot processor. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen, Fox, and the common knowledge in this manner, and thus this combination of Kennedy, Nguyen, Fox, and the common knowledge would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen, Fox, and this common knowledge in the art would have rendered the subject matter of claim 4 obvious to a POSITA.).

210. **Claim 5:** *The system of claim 1 wherein each of said plurality of processors is a processor module that includes a master processor and a slave processor* (I agree that Kennedy as modified by Nguyen and Fox taught multicore processor modules that each include master and slave execution cores. See, e.g., Nguyen (Ex. 1008) ¶ [0002], [0020], [0047]. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to improve the spatial efficiency of Kennedy's system by consolidating the amount of space occupied by the processors. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. Accordingly, I agree that Kennedy as

modified by Nguyen and Fox taught that each processor is a processor module including a master processor and a slave processor.).

211. **Claim 6:** *The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor* (I agree that Kennedy as modified by Nguyen and Fox taught that the alternate processor is an active spare before being transferred the role of boot processor. *See, e.g., Fox* (Ex. 1006) at 7:8-10. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy, Nguyen, and Fox to improve the usage and/or efficiency of the processors in the system. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught that the spare is active before being transferred the role of boot processor.).

212. **Claim 7:** *The system of claim 1 further comprising: an operating system, wherein, responsive to the logic transferring the role of boot processor to the spare processor, the operating system recognizes the spare processor as the boot processor* (I agree that a POSITA would have understood that, because Fox's failing processor automatically fails over to a hot spare processor without affecting the operating system and then control is returned to the operating system which updates the "active" processor list, the operating system would recognize the spare processor as having assumed the role of the failing processor after the role transfer. *See, e.g., Fox* (Ex. 1006) at Abstract, 4:16-25, 8:10-14, 3:28-31, 6:64-67. I also agree that, at the very least, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 7 obvious to a POSITA due at least in part to Fox's failing processor automatically being replaced by a hot spare processor without affecting the operating system and then having control returned to the operating system which updates the "active" processor list. In my opinion, it was well within the skill of a POSITA to modify Nguyen's system by Kennedy and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed.).

213. **Claim 8:** *The system of claim 1 wherein said plurality of processors are Itanium Processor Family (IPF) processors* (I agree that Kennedy as modified by Nguyen and Fox taught that the processors would have been IPF processors. *See, e.g., Nguyen* (Ex. 1008) ¶ [0045]. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy,

Nguyen, and Fox to improve the spatial efficiency of the system by consolidating the processors into the multicore processor package used by the IPF. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught this claim limitation).

214. **Claim 9: [9.1] assigning a role of boot processor to one of a plurality of processors in a system** (I agree that Kennedy taught assigning the role of boot processor to one of the processors in the system. *See, e.g., Kennedy* (Ex. 1005) at 7:9-13, 2:17-21, 1:14-15.).

215. **[9.2] assigning a role spare processor to another of said plurality of processors** (I agree that Kennedy taught assigning the role of spare processor to one of the processors in the system. *See, e.g., Kennedy* (Ex. 1005) at 7:9-13, 2:17-21, 1:14-15.).

216. **[9.3] responsive to detecting loss of lockstep for said boot processor, changing, during system runtime, the assignment of the role of boot processor to said spare processor** (I agree that Kennedy taught replacing the boot processor with an alternate processor in response to a failure of the boot processor, and Nguyen taught detecting an error (*i.e.*, a loss of lockstep). *See, e.g., Nguyen* (Ex. 1008) ¶ [0008], [0024], [0054]-[0055]. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the reliability of Kennedy's system by providing fault tolerance and high availability in the event that the boot processor fails through a loss of lockstep. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen taught changing the assignment of the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor.).

217. **[9.4] responsive to changing the assignment of the role of boot processor to said spare processor, an operating system of the system recognizing said spare processor as the boot processor without requiring shutdown of said operating system** (I agree that a POSITA would have understood that, because Fox's failing processor automatically fails over to a hot spare processor without affecting the operating system and then control is returned to the operating system which updates the "active" processor list, the operating system would recognize the spare processor as having assumed the role of the failing processor after the role transfer. *See, e.g., Fox* (Ex. 1006) at Abstract, 4:16-25, 8:10-14, 3:28-31, 6:64-67. I also agree that, at the very least,

Kennedy as modified by Nguyen and Fox would have rendered this subject matter obvious to a POSITA due at least in part to Fox's failing processor automatically being replaced by a hot spare processor without affecting the operating system and then having control returned to the operating system which updates the "active" processor list. In my opinion, it was well within the skill of a POSITA to modify Nguyen's system by Kennedy and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed.).

218. **Claim 12:** *The method of claim 9 further comprising: [12.1] determining at least one processor that is coupled to a common communication bus with said one of said plurality processors assigned to the role of boot processor* (I agree that a POSITA would have understood that any alternate processor installed in one of the slots of the backplane bus would have been coupled to a common communication bus with the boot processor. *See, e.g., Kennedy* (Ex. 1005) at 2:34-38. In my opinion, any form of identifying such an alternate processor would constitute a determination of an alternate processor coupled to a common communication bus with the boot processor. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught this claim limitation.).

219. *[12.2] selecting said another of said plurality of processors to be assigned the role of spare processor from said determined at least one processor* (I agree that Kennedy as modified by Nguyen and Fox taught determining that the spare processor is coupled to the common communication bus for at least the same reasons provided above in connection with this claim. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught selecting a processor coupled to the common communication bus to be assigned the role of spare.).

220. **Claim 13:** *The method of claim 9 further comprising: during system boot-up, system firmware identifying the spare processor and holding said spare processor in reserve as a hot spare* (I agree that the BIOS was a very well-known example of firmware. In my opinion, a POSITA would have understood that Kennedy's BIOS constitutes firmware responsible for identifying a spare processor and holding the same in reserve as a hot spare. Accordingly, I agree that Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 13 obvious to a POSITA.).

221. **Claim 14:** *The method of claim 9 further comprising: during system boot-up, system firmware booting the spare processor as an active spare* (I agree that the BIOS was a very well-known example of firmware. In my opinion, a POSITA would have understood that

Kennedy's BIOS constitutes firmware responsible for booting the spare processor as an active spare. Accordingly, I agree that Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 14 obvious to a POSITA.).

222. **Claim 15:** *The method of claim 9 wherein system firmware performs said changing operation* (I agree that the BIOS was a very well-known example of firmware. In my opinion, a POSITA would have understood that that Kennedy's BIOS, which constitutes firmware, is responsible for changing the role of boot processor to the spare processor in conjunction with the CPUs' slot selection logic. Accordingly, I agree that Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 15 obvious to a POSITA.).

223. **Claim 16:** *[16.1] a plurality of lockstep processor modules that each includes a master processor and a slave processor that is in lockstep with said master processor* (I agree that Kennedy as modified by Nguyen taught multicore processor modules that each include master and slave execution cores in lockstep. See, e.g., Kennedy (Ex. 1005) at 1:14-19; see also Nguyen (Ex. 1008) ¶ [0002], [0006], [0008], [0020], [0024], [0047]. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen to improve the spatial efficiency of Kennedy's system by consolidating the amount of space occupied by the processors. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen in this manner, and thus this combination of Kennedy and Nguyen would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen taught a plurality of lockstep processor modules that each include a master processor and a slave processor in lockstep with each other.).

224. *[16.2] a first communication bus with which at least a first two of said plurality of lockstep processor modules are communicatively coupled* (I agree that Kennedy as modified by Nguyen taught that the boot processor and the alternate processor are installed in slots on the backplane bus. See, e.g., Kennedy (Ex. 1005) at 2:34-38. In my opinion, Kennedy's backplane bus is communicatively coupled to the boot processor and alternate processor. Accordingly, I agree that Kennedy as modified by Nguyen taught a first communication bus with which at least a first two of the plurality of lockstep processor modules are communicatively coupled.).

225. *[16.3] a second communication bus with which at least a second two of said plurality of lockstep processor modules are communicatively coupled* (I agree that Kennedy as modified by Nguyen and Fox taught a plurality of processors that are interconnected to each over

via two or more different buses (e.g., system bus 110, processor-to-processor buses 203, and individual control buses 205). *See, e.g., Fox* (Ex. 1006) at 5:9-12, 5:41-45, FIG. 1. In my opinion, a POSITA would have been motivated to combine these disclosures from Kennedy and Nguyen and Fox to provide multiple communication channels and/or mechanisms for communicating with the processors. In my opinion, it was well within the skill of a POSITA to modify Kennedy's system by Nguyen and Fox in this manner, and thus this combination of Kennedy, Nguyen, and Fox would have been expected to succeed. Accordingly, I agree that Kennedy as modified by Nguyen and Fox taught a second communication bus with which at least a second two of the plurality of lockstep processor modules are communicatively coupled.).

226. ***[16.4] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors, wherein said spare processor is communicatively coupled to the same one of said first and second communication buses as said boot processor*** (I agree that Kennedy as modified by Nguyen and Fox taught memories that store CPU configuration information that includes the roles of boot processor and alternate processor, which are assigned to processors coupled to the same communication bus. *See, e.g., Kennedy* (Ex. 1005) at 7:9-13, 2:17-21, 2:34-38, 9:16-21, 7:33-35. Accordingly, Kennedy as modified by Nguyen and Fox taught data storage storing information that assigns the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors, wherein the boot processor and the spare processor are coupled to the same communication bus.).

227. **Claim 21: The system of claim 20 wherein said user interface presents for selection as said spare processor at least one processor that is communicatively to the same one of said first and second communication buses as said boot processor** (I agree that, for at least the same reasons provided above in connection with claims 4 and 16, Kennedy as modified by Nguyen and Fox would have rendered the subject matter of claim 21 obvious to a POSITA.).

B. Proposed § 102 Rejections for the Challenged Claims

1. Ground 1: Safford in light of Cepulis

228. I agree that Safford discloses each and every limitation of claims 1 and 6, especially in light of Cepulis' clarifying explanation of "logical CPU 0," for the reasons set forth in the Request. I identify evidence supporting my opinions of invalidity of claims 1 and 6 under 35 U.S.C.

§ 102 below. It is worth noting that certain claim limitations analyzed under 35 U.S.C. § 102 in the Request are omitted from the claim set below for the purpose of brevity. In my opinion, the Request's § 102 analysis of those claim limitations is sufficient and does not require my support to demonstrate a POSITA's understanding of the prior art relative to those claim limitations.

229. I have also reviewed and assisted in the preparation of Appendix D, which provides a detailed mapping of Safford in light of Cepulis to claims 1 and 6 that accurately reflects my opinions of invalidity under 35 U.S.C. § 102.

230. **Claim 1: [1.2] data storage storing information that assigns a role of boot processor to one of said plurality of processors and assigns a role of spare processor to another of said plurality of processors** (I agree that Safford disclosed this limitation, whether expressly or inherently, by teaching about memory that stores the architected states of microprocessors, including the designation of a hot standby processor, and by teaching about the logical CPU 0 designation. For at least the reasons provided in this Declaration above and in the Request, I agree that a POSITA would have understood Safford as disclosing a boot processor and failing over the same to a hot spare processor. I agree that, even if Safford did not explicitly identify which processor was the boot processor, **Cepulis shed light on Safford's use of a boot processor.** *See, e.g., Cepulis* (Ex. 1013) at 2:52-64. In my opinion, a POSITA would have also recognized "CPU 0" and "CPU 1" as being physical designations used by Safford's computing system to identify processors 111 and 113. I also agree that the use of both logical and physical designators by a computing system is generally necessary when the role of boot processor can be assigned and reassigned to a processor regardless of its physical position within a system. *See Safford* (Ex. 1007) at 4:52-53 ("The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100"). Thus, I agree that a POSITA would have understood "logical CPU 0" to be the boot-processor designation used by an operating system to determine which of its processors is the boot processor and/or to distinguish its boot processor from its non-boot processors. Furthermore, because the architected states of Safford's processors are copied from memory, I agree that a POSITA would have understood the memory to be storing the boot-processor designation (*e.g.*, logical CPU 0) and the hot-standby designation for the processors so assigned by Safford's system. *See id.* In view of these citations and the knowledge that a boot processor was a common component identified as "logical CPU 0" in computer systems (*e.g.*, Safford's system), I agree that a POSITA would have understood Safford as disclosing data storage that stores information

assigning the role of boot processor to one of the processors and assigns the role of spare processor to another one of the processors.).

231. *[1.3] logic operable, responsive to detecting loss of lockstep for said boot processor, for transferring, during system runtime, the role of boot processor to said spare processor* (I agree that Safford disclosed this limitation, whether expressly or inherently, by teaching about failing over a processor to a hot standby processor in response to a loss of lockstep. I agree that the '781 Patent notes that a **firmware component known as a processor abstraction layer (PAL)** “copies all the needed information from memory into the internal registers of the hot spare processor so that the hot spare processor becomes a new active processor, transforming the role of the hot spare to boot processor.” Ex. 1001 at 17:57-61. In my opinion, Safford disclosed this limitation through an identical process for transferring the role of boot processor to the spare processor. *See Safford* (Ex. 1004) at 32:29-32. For example, Safford taught “**moving an architected state of the processor unit experiencing the loss of lock step to the spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system.**” *Id.* As explained above, I agree that a POSITA would have understood “logical CPU 0” to be the boot-processor designation used by an operating system to determine which of its processors is the boot processor and/or to distinguish its boot processor from its non-boot processors. In view of these citations and the knowledge that a boot processor was a common component identified as “logical CPU 0” in computer systems (*e.g.*, Safford’s system), I believe a POSITA would have understood Safford as disclosing failover of the boot processor (*e.g.*, logical CPU 0) to a spare processor in response to a loss of lockstep for the boot processor, especially in light of Cepulis’ explanation of the meaning of the “logical CPU 0” boot-processor designator. Accordingly, I agree that Safford disclosed logic operable for transferring the role of boot processor to the spare processor during system runtime in response to detecting loss of lockstep for the boot processor.).

232. **Claim 6:** *The system of claim 1 wherein said another of said plurality of processors that is assigned said role of spare processor is an active spare before being transferred said role of boot processor* (I agree that Safford disclosed this limitation by teaching about a spare processor being active before a role transfer. For example, Safford taught that “the processor 111 signals the node controller 130 that **the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken ‘off-line.’**” *Safford* (Ex. 1004)

at 4:37-40. "In block 230, the node controller 130 'reboots' the processors 111 and 113, and the processors 111 and 113 become the new 'hot standby' processor pair on the system 100." *Id.* at 4:57-60. Safford also taught that "the spare processor unit becomes an active processor unit in the computer system" *Id.*, claim 1. In view of the above citations, I agree that a POSITA would have understood Safford as teaching that a spare processor can be active before a role transfer. Moreover, I agree that a POSITA would have understood that the role at issue could be the boot processor assignment in view of Safford for at least the same reasons provided above in connection with claim 1. *Id.* Accordingly, I agree that Safford disclosed that the spare processor is an active spare before being transferred the role of boot processor. *Id.*).

C. Secondary Considerations

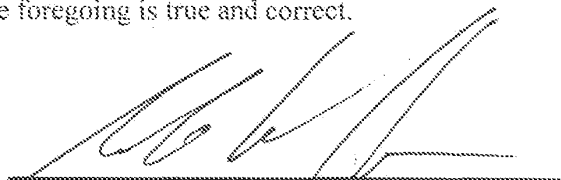
233. It is my opinion that the Challenged Claims of the '781 Patent are unpatentable as obvious in view of the prior art references. I have considered the secondary considerations that I am aware of at this time and determined that they do not impact my opinions.

IX. DECLARATION IN LIEU OF OATH

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

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

August 30, 2023



Andrew Wolfe, Ph.D.

EXHIBIT

1004



US00708595B2

(12) **United States Patent**
Safford

(10) **Patent No.:** **US 7,085,959 B2**

(45) **Date of Patent:** **Aug. 1, 2006**

(54) **METHOD AND APPARATUS FOR
RECOVERY FROM LOSS OF LOCK STEP**

(75) Inventor: **Kevin David Safford**, Fort Collins, CO
(US)

(73) Assignee: **Hewlett-Packard Development
Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 651 days.

(21) Appl. No.: **10/187,833**

(22) Filed: **Jul. 3, 2002**

(65) **Prior Publication Data**
US 2004/0006722 A1 Jan. 8, 2004

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

(52) U.S. CL. **714/11**

(58) Field of Classification Search 714/10,
714/11, 12, 13
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,589,066 A 5/1986 Lam et al.
5,732,209 A * 3/1998 Vigil et al. 714/39
6,065,135 A 5/2000 Marshall et al.
6,148,348 A 11/2000 Garnett et al.
6,263,452 B1 7/2001 Jewett et al.
6,378,021 B1 * 4/2002 Okazawa et al. 710/317

6,615,366 B1 * 9/2003 Grochowski et al. 714/10
6,751,749 B1 * 6/2004 Hofstee et al. 714/11
6,772,368 B1 * 8/2004 Dhong et al. 714/11
2002/0073357 A1 * 6/2002 Dhong et al. 714/19
2002/0144177 A1 * 10/2002 Kondo et al. 714/13
2003/0126498 A1 * 7/2003 Bighsee et al. 714/10
2004/0019771 A1 * 1/2004 Quach 712/229
2004/0153731 A1 * 8/2004 Aino et al. 714/6
2004/0153857 A1 * 8/2004 Yamazaki et al. 714/43

OTHER PUBLICATIONS

Tamir, Yuval, "Self-Checking Self-Repairing Computer
Nodes Using the Mirror Processor", IEEE Journal Of Solid-
State Circuits, vol. 27, No. 1, Jan. 1992.

Search Report issued on Dec. 30, 2003 in counterpart
foreign application in GB under application No. 315295.6.

* cited by examiner

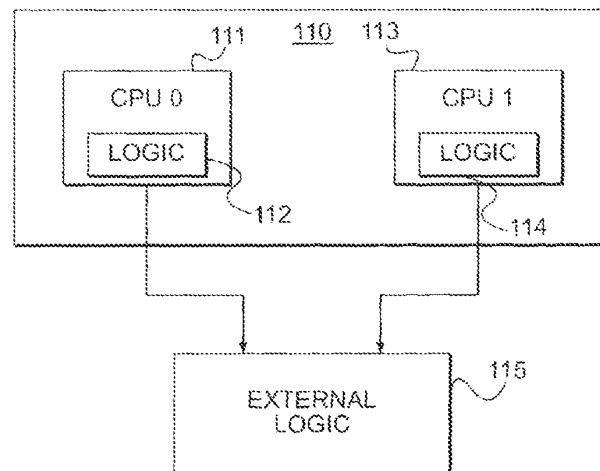
Primary Examiner—Bryce P. Bonzo

(57) **ABSTRACT**

An apparatus, operating on an advanced multi-core proces-
sor architecture, and a corresponding method, are used to
enhance recovery from loss of lock step in a multi-processor
computer system. The apparatus for recovery from loss of
lock step includes multiple processor units operating in the
computer system, each of the processor units having at least
two processor units operating in lock step, and at least one
idle processor unit operating in lock step; and a controller
coupled to the two processor units operating in lock step and
the idle processor unit. The controller includes mechanisms
for copying an architected state of each of the two lock step
processor units to the idle processor unit.

14 Claims, 4 Drawing Sheets

100



10

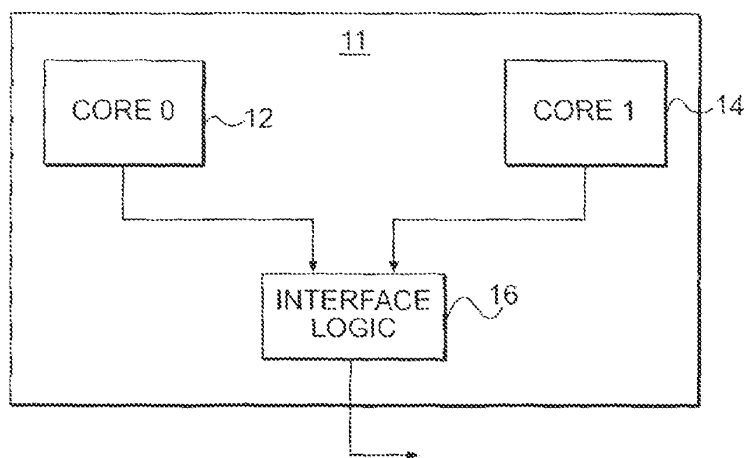


FIG. 1A
PRIOR ART

18

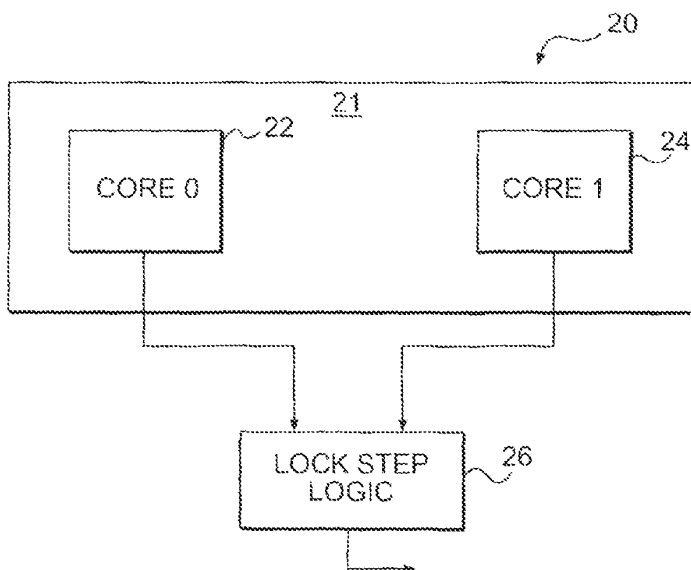


FIG. 1B
PRIOR ART

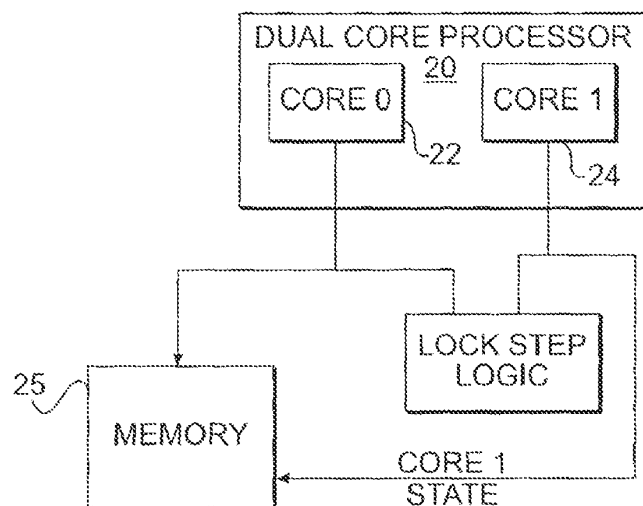


FIG. 1C
PRIOR ART

100

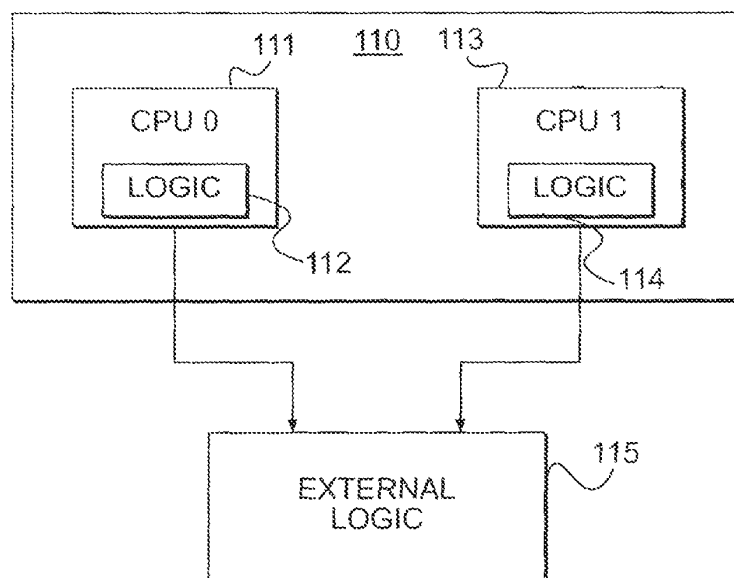


FIG. 2

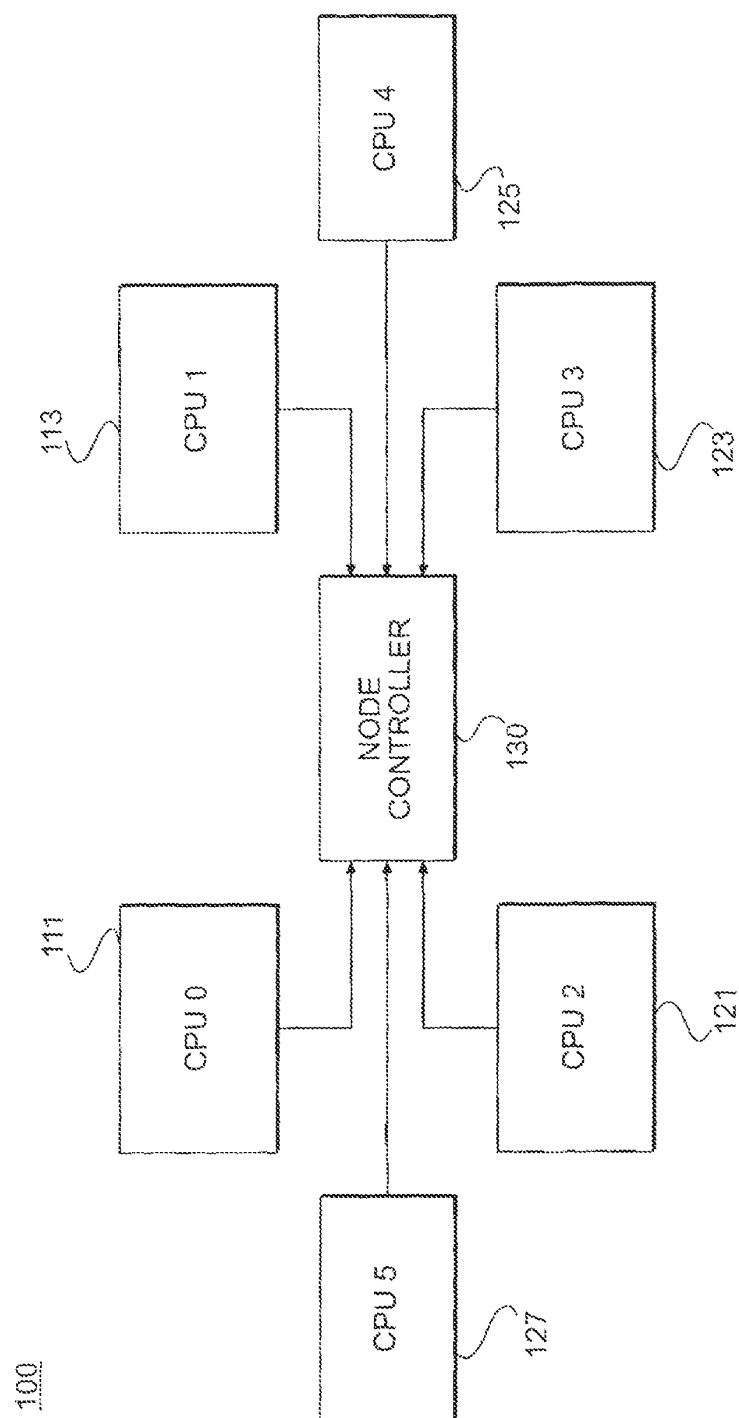
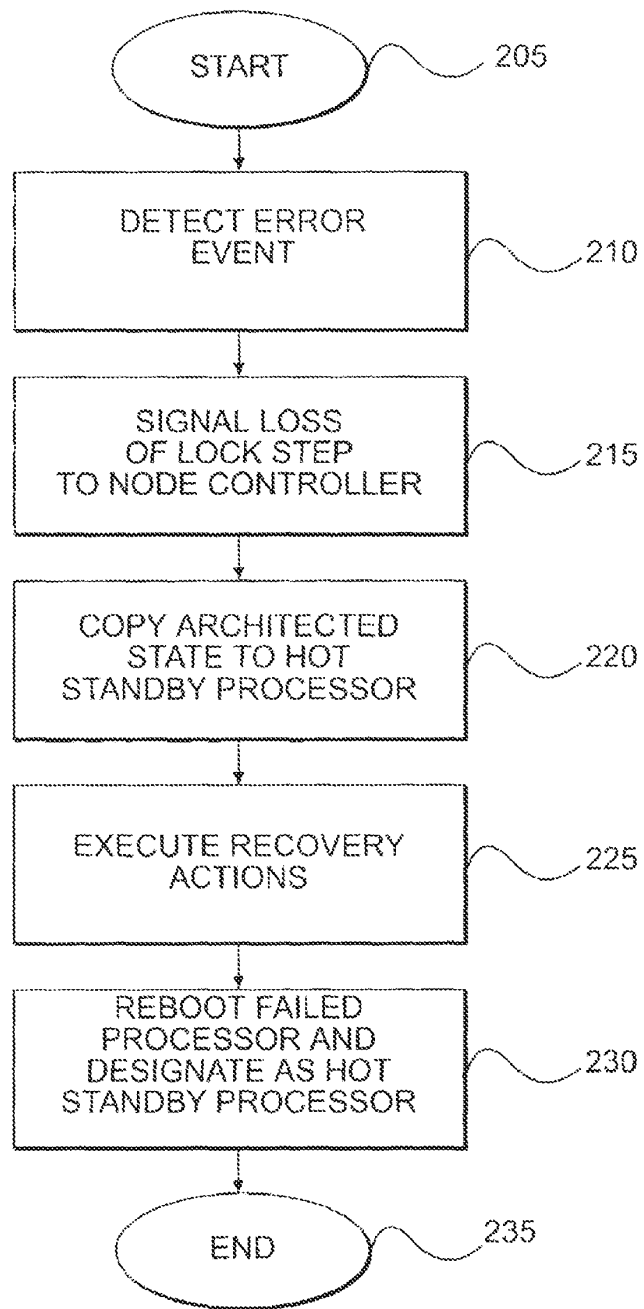


FIG. 3

**FIG. 4**

METHOD AND APPARATUS FOR RECOVERY FROM LOSS OF LOCK STEP

TECHNICAL FIELD

The technical field is computer systems employing lock stepped microprocessors.

BACKGROUND

Advanced computer architectures may employ multiple microprocessors. Some advanced computer architectures may employ multiple microprocessors on one silicon chip. In a typical application, two microprocessors may be implemented on a single silicon chip, and the implementation may be referred to as a dual core processor. Two or more of the multiple microprocessors may operate in a lock step mode, meaning that each of the lock stepped microprocessors process the same code sequences, and should, therefore, produce identical outputs. FIG. 1A illustrates a typical implementation of a dual core processor. A dual core processor 10 includes a silicon chip 11 having microprocessor core 12 (core 0) and microprocessor core 14 (core 1). The microprocessor cores 12 and 14 are coupled to an interface logic 16 that monitors external communications from the microprocessor cores 12 and 14. In the dual core processor 10, the microprocessor cores 12 and 14 operate as independent entities. While the dual core processor 10 has advantages in terms of size and processing speed, the reliability of the dual core processor 10 is not significantly better than that of two single core processors.

To enhance reliability, the dual core processor, or other multiple microprocessor architected computer systems, may employ lock step features. FIG. 1B is a diagram of a prior art dual core processor that uses lock step techniques to improve overall reliability. In FIG. 1B, a computer system 18 includes a dual core processor 20 having a single silicon chip 21, on which are implemented microprocessor core 22 and microprocessor core 24. To employ lock step, each of the microprocessor cores 22 and 24 process the same code streams. To ensure reliable operation of the dual core processor 20, each of the microprocessors 22 and 24 may operate in "lock step." An event that causes a loss of lock step can occur on either or both of the microprocessor cores 22 and 24. An example of such an event is a data cache error. A loss of lock step, if not promptly corrected, may cause the computer system 18 to "crash." That is, a failure of one microprocessor core may halt processing of the dual core processor 20, and the computer system 18, even if the other microprocessor core does not encounter an error.

To detect a loss of lock step, a lock step logic 26, which may be external to the chip 21, compares outputs from the microprocessor cores 22 and 24. A difference in processing detected by the lock step logic 26 is by definition a loss of lock step. A drawback to the dual core processor architecture shown in FIG. 1B is that the logic to determine loss of lock step is external to the chip. This configuration imposes delays in determining loss of lock step, and requires additional architectural features.

The dual core processor 20 also makes recovery from a loss of lock step difficult and time-consuming. FIG. 1C illustrates a current methodology for recovering from a loss of lock step. In FIG. 1C, the dual core processor 20 is shown coupled to memory 25. Should the dual core processor 20 suffer a loss of lock step, recovery may be initiated by the memory 25 saving the architected state of one of the microprocessors 22 and 24 (i.e., the microprocessor that is

considered "good"). Then, both microprocessors 22 and 24 are reset and reinitialized. Finally, the architected states of each of the microprocessors 22 and 24 is copied from the memory 25 into the microprocessors 22 and 24, respectively. This prior art methodology for recovery from a loss of lock step makes the microprocessors 22 and 24 unavailable for an amount of time. If the amount of time required for recovery is too long, the computer system 18 employing the dual core processor 20 may "crash."

SUMMARY

An apparatus, operating on an advanced multi-core processor architecture, and a corresponding method, are used to enhance recovery from loss of lock step in a computer system. In an embodiment, the apparatus for recovery from loss of lock step comprises a plurality of processor units operating in the computer system, each of the processor units comprising at least two processor units operating in lock step, and at least one idle processor unit operating in lock step; and a controller coupled to at least the at least two processor units operating in lock step and the at least one idle processor unit, the controller comprising means for copying an architected state of each of the at least two processor units to the idle processor unit.

The method comprises receiving a loss of lock step signal from a processor unit; receiving a notice from the processor unit experiencing the loss of lock step to take the processor unit off line; and moving an architected state of the processor unit experiencing the loss of lock step to a spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system.

DESCRIPTION OF THE DRAWINGS

The detailed description will refer to the following figures, in which like numbers refer to like elements, and in which:

FIG. 1A is a diagram of a prior art dual-core processor;

FIG. 1B is a diagram of a prior art dual-core processor employing lock step;

FIG. 1C is a diagram illustrating prior art recovery from loss of lock step;

FIG. 2 is a diagram of a computer system that uses an improved, multi-core processor employing lock step processing;

FIG. 3 illustrates additional architectural features for use in recovery from loss of lock step for the computer system of FIG. 2; and

FIG. 4 is a flowchart illustrating a process for recovery from loss of lock step in the computer system of FIG. 3.

DETAILED DESCRIPTION

To improve reliability of processing assets, a computer system employs lock stepped processor cores that operate in a master/checker pair. Each of two processors in the pair processes the same code sequences, and the resulting outputs of the processors are compared by a logic circuit located near external interfaces of the two processors. Any difference in the processor outputs indicates the existence of an error. The logic circuit may then initiate a sequence of steps that halt operation of the two processors. FIG. 2 shows a computer system 100 that employs processors 111 (central processor unit (CPU) 0) and 113 (CPU 1), which, in an embodiment, may be located on a common silicon chip or substrate 110. Alternatively, the processors 111 and 113 may

be implemented on separate substrates. The processors 111 and 113 may operate in an independent mode, or in a lock step mode. When operating in a lock step mode, the processors 111 and 113 will appear to the computer system 100 to be a single processor core, or a logical CPU 0. The processor 111 may include error detection and signaling logic 112, and the processor 113 may include error detection and signaling logic 114. The error detection and signaling logic will be described later.

External logic circuit 115 monitors outputs of the processors 111 and 113 and may be used to detect any differences in the outputs. As noted above, such differences are indicative of a potential error in at least one of the processors 111 and 113. However, which of the processors 111 and 113 is subject to an error condition may not be known. On rare occasions, both the processors 111 and 113 may be subject to an error condition. Such an error condition may lead to a halt in processing of the processors 111 and 113 until the error can be corrected. In other words, any difference in the outputs causes a loss of lock step, and a halt to processing.

To improve availability of the processors assets of the computer system 100, additional features, such as means for detecting and signaling occurrence of errors, may be incorporated into the computer system 100. For example, the error detection and signaling logic 112 and 114 may be included in the processors 111 and 113, respectively, or in other parts of the computer system 100, to signal an impending loss of lock step. Using the impending loss of lock step signal, the computer system 100 may continue operating (processing) using one of the processors 111 and 113 that did not experience an error. In particular, certain events within either of the lock stepped processors 111 and 113 may be used by the processors 111 and 113, respectively, to indicate the impending loss of lock step. As an example, and possibly due to completely random circumstances, a data cache error for a cache associated with the processor 111 may occur. Such an error can be completely corrected (i.e., the processor 111 does not need to be replaced), but will guarantee that the processors 111 and 113 will break lock step at some future time because the data cache error causes timing differences between the processors 111 and 113. The processor 111 may detect the data cache error, and use the detection of this data cache error to signal the logic circuit 115 that the processor 111 is experiencing an error that will cause a loss of lock step, and that the processor 111 is "bad." The logic circuit 115 may then "turn off," thereby ending lock step operations, and processing may continue using the "good" processor 113. At some future time, recovery from the loss of lock step (and correction of the data cache error) is executed to restore lock step operation of the processors 111 and 113.

FIG. 3 illustrates further architectural details for recovery from loss of lock step in the computer system 100 of FIG. 2. In FIG. 3, the computer system 100 is shown with additional processors 121, 123, 125, and 127, as well as the processors 111 and 113. The processors 111, 113, 121, 123, 125, and 127 are coupled to node controller 130. The processors operate as pairs when in lock step (i.e., the processors 111 and 113 are a first pair; the processors 121 and 123 are a second pair; and the processors 125 and 127 are a third pair). From the node controller's perspective, each pair of processors appears as a single (logical) processor. The processor pairs, or processor units, are coupled to a lockstep logic, such as the lockstep logic 115 shown in FIG. 2, and the lockstep logic is then connected to the node controller 130. The node controller 130 provides means for copying the architected state of a processor to another

processor. In an embodiment, the node controller 130 has available at all time a current architected state of the processors to which the node controller 130 is coupled. In another embodiment, the node controller 130 simply provides means for communication among the processors 111, 113, 121, 123, 125, and 127. For example, the node controller 130 may store the architected state of the processors 111, 113, 121, 123, 125, and 127, either internally in the node controller 130, or in another component of the computer system 100. Alternatively, the node controller 130 may allow one processor (e.g., the processor 111) to copy the architected state of the processor 111 to another processor (e.g., the processor 125). In yet another alternative embodiment, the node controller 130 may allow a processor that has broken lock step to copy, as part of the process for recovering from loss of lock step, the architected state of the processor to the node controller 130, which will in turn copy the architected state to a "hot standby" processor.

The six processors 111, 113, 121, 123, 125, and 127 operate in lock step (i.e., are processing code sequences). For example, the processor 111 operates in lock step with the processor 113, and the processor 121 operates in lock step with the processor 123 and the processor 125 operates in lock step with the processor 127.

The processor 125 may be designated as a "hot standby," and is sitting idle in lock step mode with the processor 127. Should one of the processors 111, 113, 121, and 123 suffer an error, the hot standby processors 125, 127 may be used to speed recovery from the resulting loss of lock step.

FIG. 4 is a flow chart illustrating a process 200 for recovery from a loss of lock step using the computer system 100 shown in FIG. 3. The process 200 will be shown with an error condition in the first processor pair 111/113. The operation 200 begins in block 205 with the system 100 operating in a normal lock step fashion. In block 210, the processor 111 detects an error event that indicates an impending loss of lock step. In block 215, the processor 111 signals the node controller 130 that the first processor pair 111/113 has broken lock step and that the first processor pair 111/113 should be taken "off-line." In block 220, the node controller 130 copies the architected state of the first processor pair 111/113 to the hot standby processor pair 125/127. In an embodiment, the architected state of the first processor pair 111/113 may be stored in the node controller 130, and to facilitate recovery, the node controller 130 copies the stored state to the third processor pair 125/127. Alternatively, the node controller 130 may copy the state of the first processor pair 111/113 directly from the processors 111 and 113 to the processors 125 and 127 without any intermediate storage of the architected state in the node controller 130, or other component of the computer system 100. The processor pair 125/127 then becomes the logical CPU 0 in the computer system 100, and the computer system 100 operates without a hot standby processor pair. In block 225, recovery actions are executed on the first processor pair 111/113 (e.g., all caches are flushed on the processors 111 and 113). In block 230, the node controller 130 "reboots" the processors 111 and 113, and the processors 111 and 113 become the new "hot standby" processor pair on the system 100. In block 235, the operation 200 ends, with the computer system 100 operating the processors 121, 123, 125, and 127 in lock step, and with the processors 111/113 idle and in hot standby.

The terms and descriptions used herein are set forth by way of illustration only and are not meant as limitations. Those skilled in the art will recognize that many variations are possible within the spirit and scope of the invention as

5

defined in the following claims, and there equivalents, in which all terms are to be understood in their broadest possible sense unless otherwise indicated.

The invention claimed is:

1. A method for recovery from loss of lock step in a computer system, comprising:
 - receiving a loss of lock step signal from a processor unit;
 - receiving a notice from the loss of lock step processor unit to take the loss of lock step processor unit off line; and
 - moving an architected state of the loss of lock step processor unit to a spare processor unit, wherein the spare processor unit becomes an active processor unit in the computer system.
2. The method of claim 1, further comprising correcting an error condition in the loss of lock step processor unit.
3. The method of claim 2, further comprising rebooting the loss of lock step processor unit, whereby the rebooted processor unit becomes a new spare processor unit.
4. The method of claim 1, wherein moving the architected state comprises copying the architected state from a node controller that couples the loss of lock step processor unit and the spare processor unit.
5. The method of claim 1, wherein moving the architected state comprises copying the architected state directly from the loss of lock step processor unit to the spare processor unit.
6. The method of claim 1, wherein moving the architected state comprises:
 - copying the architected state from the loss of lock step processor unit to a node controller; and
 - copying the architected state from the node controller to the idle processor unit.
7. An apparatus to recover from a loss of lock step in a multiprocessor computer system, wherein two or more processor units operate in lock step, comprising:
 - in each of the two or more processor units operating in lock step:
 - means for detecting a loss of lock step initiating event,
 - and

6

means for signaling an impending loss of lock step; and means for moving an architected state of a processor unit having the loss of lock step initiating event to a spare processor unit, the spare processor unit operating idle in lock step.

8. The apparatus of claim 7, further comprising means for taking the loss of lock step processor unit off line.

9. The apparatus of claim 8, further comprising means for rebooting the loss of lock step processor unit, whereby the rebooted processor unit is designated as a new spare processor unit.

10. The apparatus of claim 7, wherein the moving means comprises:

- means for copying the architected state; and
- means for storing the architected state.

11. The apparatus of claim 10, wherein the means for storing the architected state comprises a node controller coupled to the two or more processor units and the spare processor unit.

12. The apparatus of claim 7, wherein the moving means comprises means for copying the architected state directly from the loss of lock step processor unit to the spare processor unit.

13. A method for recovering from a loss of lock step operation in a multi-processor computer system, comprising:

detecting a loss of lock step initiating event in a processor; signaling an impending loss of lock step; copying an architected state of the loss of lock step processor to a spare processor; and idling the loss of lock step processor.

14. The method of claim 13, further comprising: correcting the loss of lock step event in the loss of lock step processor; and rebooting the loss of lock step processor, whereby the rebooted processor is a new spare processor.

* * * * *