UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,

Petitioner

v.

BITMICRO LLC,

Patent Owner

Case No.: IPR2023-00787

U.S. Patent No. 10,120,694

PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 10,120,694

TABLE OF CONTENTS

Page

I. INTRODUCTION 1

II. GROUNDS FOR STANDING 2

III. IDENTIFICATION OF CHALLENGE 2

A. Prior Art 2

B. § 325(d): The Art and Arguments Were Not Previously Presented 3

C. Factors for Discretionary Denial are Not Present 5

D. Grounds for Challenge 5

IV. TECHNICAL BACKGROUND 5

A. Power-On Reset and Boot 6

B. System Registers 8

C. Direct Memory Access ("DMA") 9

V. CHALLENGED U.S. PATENT NO. 10,120,694 11

A. Overview of the '694 Patent 11

B. Prosecution History 14

C. Claim Construction 17

D. Level of Ordinary Skill in the Art 17

VI. BASIS OF THE UNPATENTABILITY CHALLENGE 17

A. Overview of Prior Art 17

1. Post Disclosed a Process for Copying Boot Data from Nonvolatile Memory to RAM and Verifying Integrity of the Boot Data 17

2. Kao Disclosed a DMA Engine and DMA Techniques for Booting an Embedded System 22

3. Wilson Disclosed Techniques for Initializing Registers of an Embedded System During a Boot Process 23

TABLE OF CONTENTS
(continued)

Page

4. So Disclosed a Reset Controller within an SoC for an Internet Phone that Generated Reset Signals 25

5. Overview of Combinations 27

B. Claim 1: the Combinations of Post with Kao and Wilson and Post with Kao, Wilson, and So render claim 1 obvious.....27

1. Claim 1[pre]. An apparatus, comprising:.....27

2. Claim 1[a]. an embedded system comprising one or more processors, a reset controller, a storage device controller, one or more direct memory access (DMA) controllers, a random access memory (RAM) and a memory controller, a nonvolatile memory and a nonvolatile memory controller, a debug interface, and.....28

3. Claim 1[b]. a power-on reset (POR) sequencer configured to detect, in the nonvolatile memory, a descriptor used in a boot process and configured to copy the descriptor from the nonvolatile memory to the RAM, ...41

4. Claim 1[c]. wherein the POR sequencer facilitates a boot process and verifies an integrity of loaded data and code in the RAM,.....53

5. Claim 1[d]. wherein the descriptor comprises a POR sequencer descriptor, and wherein the POR sequencer uses the POR sequencer descriptor which is a preassembled descriptor that is stored in the nonvolatile memory, ...56

6. Claim 1[e]. wherein the POR sequencer descriptor includes register information and DMA controller descriptors, and 60

7. Claim 1[f]. wherein the POR sequencer reads and uses the register information to update configuration registers of the embedded system.....65

TABLE OF CONTENTS
(continued)

Page

C. Claim 5. The apparatus of claim 1, further comprising: a CPU bus in the embedded system. 67

D. Claims 6 and 9 69

E. Claim 12 69

1. Claim 12[pre]. An article of manufacture, comprising: 69

2. Claim 12[a]. a non-transitory computer-readable medium having stored thereon instructions to permit an apparatus to perform a method comprising: 70

3. Claim 12[b]-12[f] 71

VII. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(B) 71

A. Real Party in Interest 71

B. Related Matters 71

C. Lead and Back-Up Counsel 72

VIII. CONCLUSION 72

EXHIBIT LIST

Exhibit	Description
INTEL-1001	U.S. Patent 10,120,694 to Manlapat ("the '694 patent")
INTEL-1002	File History of U.S. Patent 10,120,694
INTEL-1003	Declaration of Dr. Donald Alpert
INTEL-1004	U.S. Patent 7,490,177 to Kao ("Kao")
INTEL-1005	U.S. Patent 8,799,555 to Post et al. ("Post")
INTEL-1006	Notice of Service upon Intel of BiTMICRO's Complaint on April 1, 2022
INTEL-1007	Microsoft Computer Dictionary (5th Ed.)
INTEL-1008	J. Catsoulis, Designing Embedded Hardware (2005)
INTEL-1009	J. Ganssle et al., Embedded Hardware (2008) ("Ganssle")
INTEL-1010	Q. Li, Real-Time Concepts for Embedded Systems (2003)
INTEL-1011	Federal Judiciary Statistics prepared by Administrative Office of the U.S. Courts
INTEL-1012	P. Barry et al., Modern Embedded Computing (2012)
INTEL-1013	U.S. Patent 7,861,073 to Wilson ("Wilson")
INTEL-1014	U.S. Patent No. 7,921,237 ("Holland")
INTEL-1015	M. Mintz et al., Hardware Verification with SystemVerilog (2007)
INTEL-1016	T. Noergaard, Embedded Systems Architecture (2005)
INTEL-1017	AHB Example AMBA System Technical Reference Manual, ARM, 1999
INTEL-1018	J. Lenk, Simplified Design of Microprocessor Supervisory Circuits (1998)
INTEL-1019	U.S. Patent App. No. 2005/0273588 ("Ong")
INTEL-1020	Power-On Reset and Related Supervisory Functions, Application Note 327, Maxim Integrated, July 8, 2004
INTEL-1021	U.S. Patent 7,526,679 ("So")

I. INTRODUCTION

Intel Corporation ("Intel" or "Petitioner") petitions for *inter partes* review (IPR) of claims 1, 5, 6, 9, and 12 of U.S. Patent No. 10,120,694 (INTEL-1001). The '694 patent describes methods, apparatuses, and articles of manufacture for booting up an embedded system using a descriptor that the patentees called a "POR [power-on reset] sequencer descriptor." (*Id*., Abstract, 2:46-52, claims 1-13.)

The '694 patent specification and challenged claims describe the POR sequencer descriptor according to its components: "The POR Sequencer descriptor contains register information and direct memory access (DMA) controller descriptors." (*Id*., 2:50-52.) At system power-on, the POR sequencer descriptor is copied from non-volatile memory to RAM. (*Id*., 3:12-23.) The embedded system uses register information in the POR sequencer descriptor to program configuration registers (*id*., 3:26-32) and then processes the DMA descriptor to load additional information or code from the nonvolatile memory (*id*., 3:32-43).

The phrase "POR sequencer descriptor"—which, during prosecution of the related U.S. Patent No. 9,858,084, was added to the claims to overcome prior art—was coined by the applicants. But there is nothing in the recited elements of the POR sequencer descriptor, or in the recited functionality of the embedded system, that was not already known in the art. Instead, the claims recite conventional embedded system components, boot functionality, and Direct Memory Access techniques that

were known years before the '694 patent's priority date. This Petition, supported by the declaration of Dr. Donald Alpert, demonstrates that the challenged claims were obvious in view of the cited prior art and are unpatentable.

II. GROUNDS FOR STANDING

Petitioner certifies the '694 patent is available for IPR and that Petitioner is not barred or estopped from requesting IPR on the grounds in this Petition. (INTEL-1006.)

III. IDENTIFICATION OF CHALLENGE

A. Prior Art

The '694 patent issued from application 14/217,365 filed on March 17, 2014. Each reference applied in this Petition was filed, issued, or was published before March 15, 2012.

1. U.S. Patent 8,799,555 to Post et al. (INTEL-1005), was filed April 14, 2011 and issued August 5, 2014. It is prior art under 35 U.S.C. §§ 102(a)(2) and pre-America Invents Act ("AIA") 35 U.S.C § 102(e).

2. U.S. Patent 7,861,073 to Wilson (INTEL-1013), was filed January 3, 2007 and issued December 28, 2010. It is prior art under 35 U.S.C. §§ 102(a)(1), 102(a)(2) and pre-AIA 35 U.S.C §§ 102(a), (b) and (e).

3. U.S. Patent 7,490,177 to Kao (INTEL-1004), was filed January 23, 2006, and issued February 10, 2009. It is prior art under 35 U.S.C. §§ 102(a)(1), 102(a)(2) and pre-AIA 35 U.S.C §§ 102(a), (b) and (e).

4. U.S. Patent 7,526,679 to So (INTEL-1021), was filed July 18, 2005, and issued April 28, 2009. It is prior art under 35 U.S.C. §§ 102(a)(1), 102(a)(2) and pre-AIA 35 U.S.C §§ 102(a), (b) and (e).

Kao was identified in an IDS during examination of the '694 patent. (*See* INTEL-1002.) The Examiner, however, does not appear to have considered Kao in combination with other references. The Post, Wilson, and So references were not presented and could not have been considered with Kao.

B. § 325(d): The Art and Arguments Were Not Previously Presented

To deny a petition "[u]nder § 325(d), the art and arguments must have been previously presented to the Office during proceedings pertaining to the challenged patent." *Advanced Bionics, LLC v. MED-EL Elektromedizinische Geräte GmbH*, IPR2019-01469, Decision, Paper No. 6 at 7. The Board's two-part framework asks: "(1) whether the same or substantially the same art previously was presented to the Office or whether the same or substantially the same arguments previously were presented to the Office; and (2) if either condition of first part of the framework is satisfied, whether the petitioner has demonstrated that the Office erred in a manner material to the patentability of challenged claims." *Id.* at 8.

The combination of Kao with any reference was not cited or examined by the Patent Office during the prosecution of the ’694 patent or related U.S. Patent No. 9,858,084. Thus, the Office has not previously decided patentability of these claims, or claims comparable to them, over the combinations presented here.

The prior art considered during prosecution is not substantially the same as that presented in this petition. At various points during the prosecution, the Examiner rejected applicant’s claims over published application US 2007/0180223 (“Tanaka”) and U.S. Patents No. 7,676,640 to (“Chow”) and No. 7,921,237 (“Holland”). Patentee distinguished Chow on the grounds that it did not use a power-on reset sequencer as part of its boot process (INTEL-1002, 68), and the Examiner allowed the current claims because the combination of Tanaka and Holland did not use power-on sequencer descriptors containing register information to update configuration registers. (*Id*., 118 (allowing then-pending claim “5” which recited these features).) Additionally, as detailed herein, the combinations described incorporate the features the Examiner found lacking in Chow and the combination of Tanaka and Holland.

Because the arguments for obviousness of the ’694 claims over a combination of Post, Wilson, and So were not previously considered by the PTO, the second part of the test for denying a petition also fails.

C. Factors for Discretionary Denial are Not Present

In addition to the compelling merits of the Petition, the *Fintiv* factors weigh heavily against discretionary denial. The '694 patent was asserted against Defendant in the Western District of Texas in early April 2022. The case was transferred to the Northern District of California ("NDCA") in February 2023. The median-time-to-trial of all civil cases in NDCA is 32.9 months. (*See* INTEL-1011, 66.) Assuming that timing from the original filing date, the NDCA trial date is December 2025, over 2 months after the final written decision. Moreover, the co-pending district court litigation is in its early stages. The court has not issued a scheduling order and claim construction, expert discovery, summary judgment and trial all remain in the future.

D. Grounds for Challenge

Ground	35 U.S.C. §	Claims	Combination
1	103	1, 5, 6, 9 and 12	Post, Wilson, and Kao
2	103	1, 5, 6, 9 and 12	Post, Wilson, Kao, and So

IV. TECHNICAL BACKGROUND

The '694 patent states that it "relates generally to computers having a single or multiple Central Processing Units (CPU), and more particularly, to a method and device for booting such a system directly from a storage device." (INTEL-1001, 1:14-18.) At the time of the '694 patent's priority date, booting processes for such systems typically involved a number of conventional steps.

A. Power-On Reset and Boot

In a processor-based system, a *boot process* is used to start the system up. System boot-up takes place after a power-on or reset condition has occurred. (INTEL-1016, 555 ("System boot-up means that some type of power-on or reset source … has occurred."); INTEL-1019, ¶ 0001 ("Upon being powered up, or in response to a reset condition, conventional microcomputers, and other devices controlled by a programmable microprocessor or microcontroller, undergo an initialization process known as "booting.").) When the system first powers on, it enters a power-on reset state in which internal hardware are reset to a known starting point. (INTEL-1018, 2 ("Power-On Reset. When power is applied to a microprocessor, the internal registers 'come up' in arbitrary states that contain random data. Applying a reset command to the microprocessor overcomes this chaos by resetting all internal circuits at a predictable starting point.").) (INTEL-1003, ¶ 31.)

Start-up firmware called "boot code" is then loaded, typically from system Read-Only Memory. (INTEL-1016, 555 ("When power is applied to an embedded board (because of a reset), start-up code, also referred to as boot code… in the system's ROM is loaded and executed by the master processor."); INTEL-1008, 18 ("The primary purpose of ROM within a system is to hold the code (and sometimes data) that needs to be present at power-up. Such software is generally known as

firmware and contains software to initialize the computer by placing I/O devices into a known state."), 3 ("At the lowest level, there are programs that are run by the processor when the computer first powers up. These programs initialize the other hardware subsystems to a known state and configure the computer for correct operation").) (INTEL-1003, ¶ 32.)

Boot code "length and functionality" differs depending on the particulars of the system. (INTEL-1016, 555.) Regardless of these particulars, however, boot code typically performs the "same (minimal) general functions" that include "initializing the hardware, … initializing buses, setting the master and slave processors in a specific state, and initializing memory." (*Id*.; INTEL-1013, 1:25-28 ("The boot-up process allows a microcontroller to perform various actions to prepare the microcontroller for ordinary operation, such as load initialization values in its registers, load instructions in its instruction memory, etc.").) Once the hardware is initialized, software, including an operating system, may be loaded to complete system setup. (INTEL-1016, 555 ("After the hardware initialization sequence … the remaining system software, if any, is then initialized."); INTEL-1008, 3 ("The bootloader … reads the operating system from disk … and places it in memory so that the processor may then run it.").) (INTEL-1003, ¶ 33.)

B. System Registers

Processor hardware includes *registers*, which are a type of volatile memory. (INTEL-1009, 70-71 ("*Register set*. A register is simply fast programmable memory normally used to store operands that are immediately or frequently used. A processor's set of registers is commonly referred to as the *register set* or the *register file*."); INTEL-1016, 137 (same); INTEL-1013, 2:54-56 ("Register file 102 can include various registers which are used by the processor 101 in the ordinary operation of the microcontroller.").) Different processors will have different register sets; "the number of registers in [processor] sets varies between very few to several hundred (even over a thousand)." (INTEL-1009, 71; INTEL-1016, 137 (similar); INTEL-1008, 32 ("Registers are the internal (working) storage for the processor. The number of registers varies significantly among processor architectures ... Some processors also have one or more control registers consisting of configuration bits that affect processor operation and the operating modes of various internal subsystems.").) (INTEL-1003, ¶ 34.)

Registers must be initialized to a known value before they can be used by the processor. (INTEL-1020, 1 ("One task of the power-on reset (POR) is ensuring that the processor starts at a known address when power is first applied.… The POR's second task is to keep the processor from starting its operation from that known address until … the internal registers have been properly loaded."); INTEL-1013,

2:56-59 ("The microcontroller can require that certain specific predefined initial values are stored in one or more of the registers of the register file before the microcontroller assumes ordinary operation.").) Register initialization values may be included in the boot data and loaded into the register set as part of the boot process. (INTEL-1013, 2:59-63 ("Since the register file is usually composed of volatile memory, the initial values can be stored in the register file as part of an initialization or boot operation … [and] can be a part of boot data that is loaded in the microcontroller during a boot-up procedure.").) (INTEL-1003, ¶ 35.)

C. Direct Memory Access ("DMA")

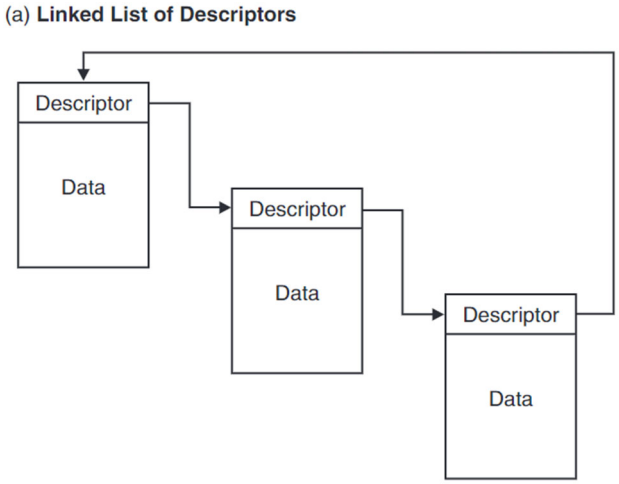
The core processor itself may be responsible for performing "data transfer between internal and external memory spaces." (INTEL-1009, 214; INTEL-1008, 20.) These memory transfer operations require significant processing workload and divert the processor from its various other tasks, thus reducing optimum performance. (INTEL-1009, 214 ("in reality you can only achieve optimum performance in your application if data can move around without constantly bothering the core to perform the transfers").) (INTEL-1003, ¶ 36.)

Direct Memory Access ("DMA") techniques were one way to offload this work from the core system processor. (INTEL-1003, ¶¶ 37-39.) DMA controllers were "peripheral[s] devoted to moving data around a system" that operated independently from the core processor. (INTEL-1009, 216; INTEL-1016, 280

("DMA (direct memory access) circuitry can be used to bypass the master CPU entirely. DMA has the ability to manage data transmissions or receptions directly to and from main memory and an I/O device. On some systems, DMA is integrated into the master processor, and on others there is a separate DMA controller. Essentially, DMA requests control of the bus from the master processor."); INTEL-1003, ¶¶ 38-39.) A DMA controller could be configured to transfer data or code stored in one location in memory to another, without requiring the processor to perform the work itself. (INTEL-1009, 215 ("You can program the DMA controller to move data in parallel with the core while the core is doing its basic processing tasks—the jobs on which it's supposed to be focused."); INTEL-1008, 20 ("Using DMA bypasses the processor by setting up a channel between the I/O device and the memory. Thus, data is read from the I/O device and written into memory without the need to execute code to perform the transfer on a byte-by-byte (or word-by-word) basis."); INTEL-1003, ¶ 38.)

DMA controllers could use *descriptors* stored in memory to initiate a DMA sequence. (INTEL-1009, 231 ("DMA transfers that are descriptor-based require a set of parameters stored within memory to initiate a DMA sequence."); INTEL-1008, 21 ("Data-chaining transfers [a]llow DMA transfers to be performed as specified by a linked-list in memory. Data chaining is started by specifying a pointer to a descriptor in memory. The descriptor is a table specifying byte count, source

address, destination address, and a pointer to the next descriptor.").) DMA "descriptors allow[ed] the chaining together of multiple DMA sequences," where another DMA transfer could be configured to automatically start when a current DMA transfer completed. (INTEL-1009, 231; INTEL-1008, 21.) For example, DMA descriptors could be configured in a "Linked List" format, with each descriptor pointing to the address of the next:



(INTEL-1009, 233 (Fig. 5.28); INTEL-1008, 21.) A "descriptor-based" DMA model thus "provide[d] the most flexibility in managing a system's DMA transfers." (INTEL-1009, 231.) (INTEL-1003, ¶ 41.)

V. CHALLENGED U.S. PATENT NO. 10,120,694

A. Overview of the '694 Patent

The '694 patent describes a system "having a single or multiple Central Processing Units (CPU), and more particularly, … a method and device for booting such a system directly from a storage device." (INTEL-1001, 1:15-19.) The patent

explains that in a "conventional embedded system"—a term the patent uses to "refer to a single CPU or a multiprocessor system"—a "nonvolatile memory … is used to store firmware that is read and executed by a CPU to initialize the system." (*Id*., 1:26-31.) The patent notes that reducing the size of this firmware would reduce the size of the required nonvolatile memory. (*Id*., 2:10-14.) It also recognizes that the boot process should be able to verify the integrity of data and initialize configuration registers of the embedded system. (*Id*., 2:24-19.)

The patent aims to address these issues through use of a "Power-On Reset ("POR") sequencer descriptor" that can be distributed across multiple storage devices. Figure 2 illustrates components of the '694 patent's embedded system, which include one or more CPUs (shown in red, below), RAM (in green), a Direct Memory Access ("DMA") controller (in yellow), a "POR sequencer" (in blue) and various fragments of a POR sequencer descriptor (in orange):

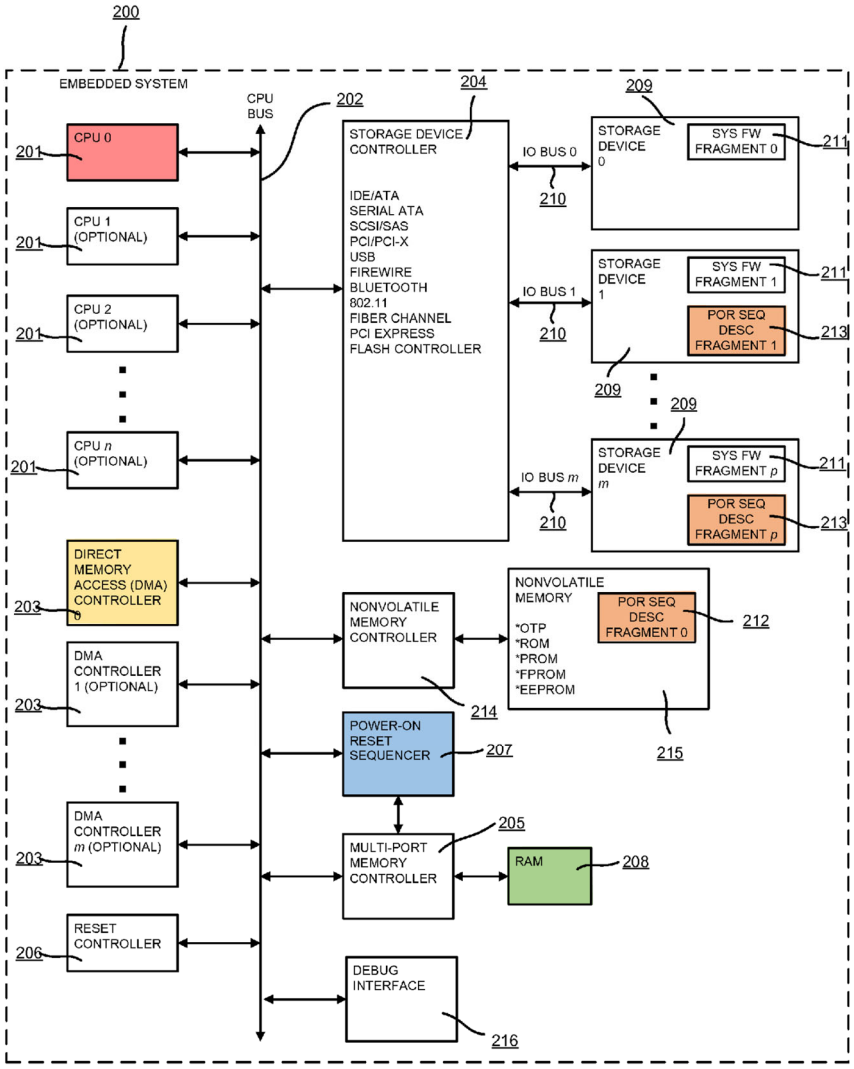


FIG. 2

(*Id*., Fig. 2, 2:29-39; 4:47-58; Fig. 2.) (INTEL-1003, ¶ 43.)

The '694 patent explains that the "POR sequencer descriptor" is "a preassembled descriptor that is stored in a nonvolatile memory of the system." (*Id*., 2:47-50.) The POR sequencer descriptor contains "register information … to update the configuration registers of the system" and "DMA descriptors" to be "loaded … to the DMA controllers of the system." (*Id*., 2:50-56.) The "head" of the POR sequencer descriptor is stored in nonvolatile memory 215. (*Id*., 5:24-26; *id*., Fig. 2

(fragment 212).) If the POR sequencer descriptor does not fit within that nonvolatile memory, it "is divided into fragments and distributed across the nonvolatile memory and storage devices attached to the system." (*Id*., 6:34-37; *id*., Fig. 2 (fragments 213).) (INTEL-1003, ¶¶ 43-44.)

A "Power-On Reset Sequencer" component "controls the overall boot process of the embedded system." (*Id*., 2:35-36.) At system power-on, the POR sequencer and various other system components are brought out of reset. (*Id*., 6:56-61.) The POR sequencer "detects the presence of POR sequencer descriptor 212 in the nonvolatile memory [215]." (*Id*., 6:62-63; *id*., Fig. 2.) It then copies POR sequencer descriptor 212 from nonvolatile memory 215 to RAM and "checks the integrity of POR sequencer descriptor 212." (*Id*., 6:64-7:1.) The POR sequencer then performs a "register initialization phase" by "read[ing] the first register information … of the POR sequencer descriptor" from the RAM and "updat[ing] the corresponding configuration register based on the contents of the register information." (*Id*., 7:17-24.) The POR sequencer repeats these steps until all register information in the POR sequencer descriptor is read and loaded into registers. (*Id*., 7:24-27.) (INTEL-1003, ¶ 45.)

B. Prosecution History

The '694 patent application was filed on March 17, 2014 and examined under the first inventor to file provisions of the AIA. (INTEL-1002, 33-36; *id*., 44, 113.)

The first office action imposed a restriction requiring Applicant to select between apparatus claims to an "embedded system" and method claims to a "method of booting up an embedded system." (INTEL-1002, 44.) Applicant elected to pursue the apparatus claims. (*Id*., 48-51.)

In the first action on the merits, the Examiner rejected the independent claims as being anticipated by Chow, U.S. Patent No. 7,676,640. (*Id*., 57-58.) In its response, Applicant amended independent claim 1 to add the limitation of "[a power-on reset (POR) sequencer] configured to detect, in the nonvolatile memory, a descriptor used in a boot process and configured to copy the descriptor from the nonvolatile memory to the RAM." (*Id*., 62.) Applicant argued that Chow et al did not disclose the claimed "power-on reset (POR) sequencer" of amended claim 1 and that new claims 21-28 incorporated the limitations of the currently pending claims. (*Id*., 68-69.) The Examiner then allowed the pending claims 1-9 and 21-28. (*Id*., 76.)

Applicant filed two requests for continued examination ("RCE"), and the Examiner allowed the claims after each. (*Id*., 79-83, 89, 94-96.) Applicant then filed a third RCE with an amendment adding new claims. (*Id*., 104-107.) The Examiner then rejected all claims for non-statutory obviousness-type double patenting over the co-pending application that issued as U.S. Patent No. 9,858,084, and also rejected most claims as obvious over the combination of US 2007/0180223

A1 to Tanaka and U.S. Patent No. 7,921,237 to Holland et al. (*Id*., 114-118.) However, the Examiner stated that certain pending dependent claims recited patentable material relating to the "POR sequencer descriptor" element and would be allowable if presented independently. (*Id*., 118.) Applicant, therefore, added this text at the end of pending claims 1 and 29 (which became issued independent claims 1 and 12):

> [1[d]] wherein the descriptor comprises a POR sequencer descriptor, and wherein the POR sequencer uses the POR sequencer descriptor which is a preassembled descriptor that is stored in the nonvolatile memory,
>
> [1[e]] wherein the POR sequencer descriptor includes register information and DMA controller descriptors, and
>
> [1[f]] wherein the POR sequencer reads and uses the register information to update configuration registers of the embedded system.

(*Id*., 122, 127].) The claims were then allowed, and issued, as amended.

As Petitioner demonstrates below, none of the elements recited by the claims was new to the art. The use of register information to update configuration registers in an embedded system, which the Examiner did not find in the art then available to him, was entirely conventional, as this Petition demonstrates. And the other elements of the claims, reciting things such as preassembled register information and DMA controller descriptors, were also already known in the art. The Examiner's rejection of claims that included only those elements was correct.

C. Claim Construction

A claim in an IPR is construed "in accordance with the ordinary and customary meaning of such claim." 37 C.F.R. §42.100(b). Petitioner does not believe any term requires construction to resolve the patentability disputes in this proceeding. *See Vivid Techs., Inc. v. Am. Sci. & Eng'g,* 200 F.3d 795, 803 (Fed. Cir. 1999). The Petition thus applies the plain and ordinary meaning to the language of the challenged claims.

D. Level of Ordinary Skill in the Art

A person having ordinary skill in the art ("POSITA") would possess a Bachelor of Science degree in electrical engineering or computer science, and at least two years of experience in the field of embedded system design, including booting of such systems, or an equivalent combination of education, work and/or experience. (INTEL-1003, ¶ 20.)

VI. BASIS OF THE UNPATENTABILITY CHALLENGE

A. Overview of Prior Art

1. Post Disclosed a Process for Copying Boot Data from Nonvolatile Memory to RAM and Verifying Integrity of the Boot Data

Post described a method for booting an embedded system within an electronic device, such as an iPhone, desktop or laptop computer, or "any other suitable type of electronic device or system." (INTEL-1005, 2:60-67.) Figure 1 of Post shows

electronic device 100 that included system on chip (SoC) 110 and non-volatile memory ("NVM") 120:

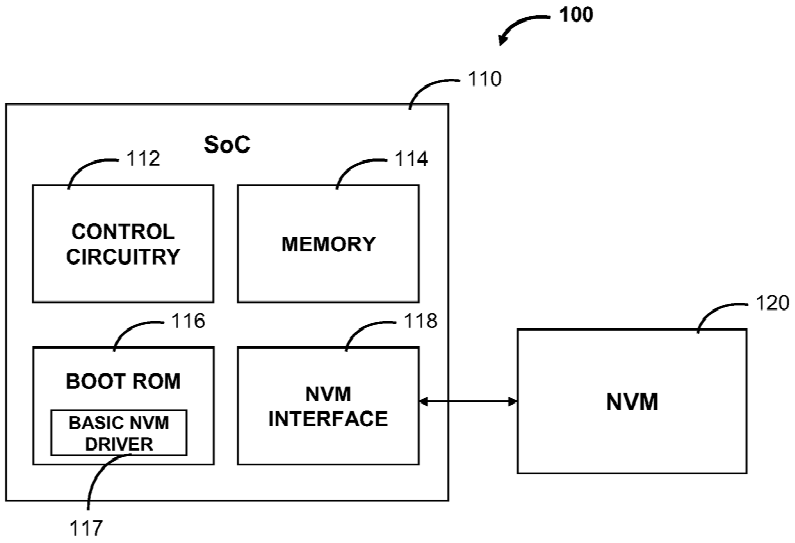


FIG. 1

SoC 110 had control circuitry 112 that "control[led] the general operations and functions of SoC 110 and electronic device 100 in general." (*Id*., 3:34-36.) Control circuitry 112 "included any suitable components, circuitry, or logic, such as one or more processors." (*Id*., 3:36-38.) SoC 110 further included memory 114 that was "any suitable type of volatile memory," such as RAM. (*Id*., 3:33-34, 3:44-49.) (INTEL-1003, ¶ 53.)

Post described a process for organizing pages of boot data in nonvolatile memory 120 using a linked-list format and retrieving this boot data from nonvolatile memory during system boot. Nonvolatile memory 120 included a "boot data storage area" made up of "any suitable number of integrated circuits, blocks, or pages." (*Id*.,

4:27-35.) The pages of nonvolatile memory containing boot data (*i.e.*, "boot data pages" (*id.*, 5:6-8)) could be nonconsecutive. (*Id.*, 1:45-48 ("[T]he boot data may be stored in only a subset of the memory locations in the boot data storage area … includ[ing] nonconsecutive pages in one or more blocks of the non-volatile memory."); *id.*, 1:45-48.) (INTEL-1003, ¶ 54.)

Boot page metadata included "a next page pointer 302 and a prior page pointer 304," such that "the boot data pages … form a first linked list using the next page pointers … [and] form a second linked list using the prior page pointers." (*Id.*, 6:29-35.) Post's Figure 3 illustrated boot data pages (color-coded in orange, below), with metadata including next-page pointers 302 and prior-page pointers 304, within blocks of nonvolatile memory 120:

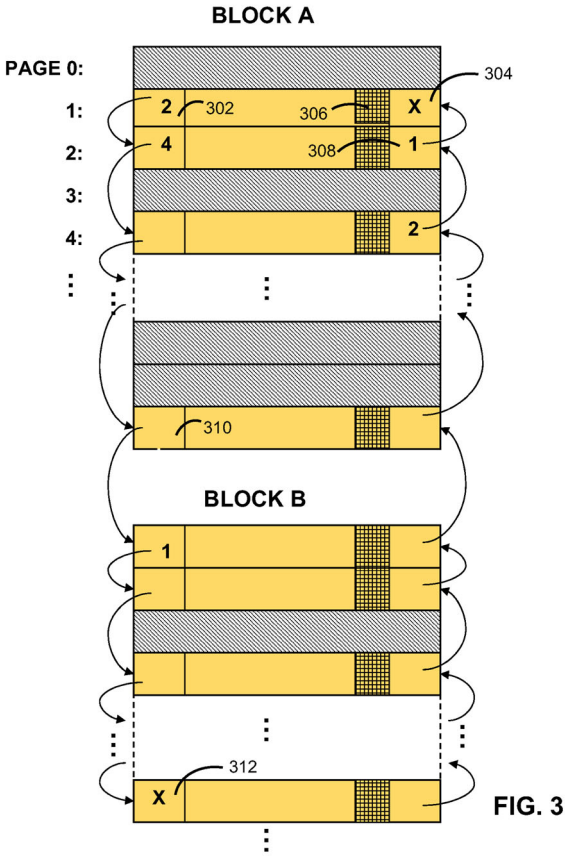


FIG. 3

(*Id.*; INTEL-1003, ¶ 54.)

At system boot-up, the control circuitry 112 executed program code stored in boot ROM 116. (*Id.*, 3:53-55, 3:41-43.) The control circuitry then "load[ed] additional information from NVM 120 into memory 114." (*Id.*, 3:55-57.) This information included "firmware, such as a first stage bootloader," which is what Post "referred to as 'boot data.'" (*Id.*, 3:57-59, 3:63-65; INTEL-1003, ¶ 55.) To load the boot data, the control circuitry read pages from the nonvolatile memory until it identified a first page containing boot data. (*Id.*, 8:15-19, 8:29-35, 8:43-49, Fig. 4; INTEL-1003, ¶ 56.) The control circuitry loaded that boot data into memory. (*Id.*,

9:3-6, Fig. 4 (step 420, "Maintain the Boot Data in Memory"), Fig. 5 (step 510, same; INTEL-1003, ¶ 56.) It then examined the next-page pointer to determine whether there were more pages of boot data. If there were, it saved the next-page pointer "for use in processing subsequent pages" and read the next page into memory:

> Following step 510, the electronic device can determine at step 512 whether the current page contains the last page of boot data. For example, the electronic device can examine the next page pointer to determine whether the pointer references a subsequent page of the non-volatile memory, or whether the pointer indicates that there are no more pages of boot data. …. If the current page is not the last page, process 500 may instead continue to step 516 ["Save the Page's Next Page Pointer"], where the electronic device can save the current page's next page pointer for use in processing subsequent pages. Process 500 may then return to step 504, where the electronic device can continue reading pages from the non-volatile memory to obtain the rest of the boot data.

(*Id*., 9:47-62; *id*., Fig. 5.) By using pointers to identify additional pages of boot data and iterating through this process until "there are no more pages of boot data," the control circuitry loaded all boot data into memory. (*Id*., 9:14-15, 9:52-56, Fig. 5; INTEL-1003, ¶ 56.)

Post's control circuitry incorporated integrity checking into its boot data transfer process. The prior-page and next-page pointers were used to "verify the integrity of boot data storage." (INTEL-1005, 6:54-55; *id*., 7:21-23.) The step of determining whether a page included boot data involved "employing an error

correcting code" on the page to detect uncorrectable errors in the boot data overall. (*Id.*, 8:34-38, 6:60-65.) Post's process thus provided a resilient boot up sequence that could determine if critical boot data was missing and abort the boot sequence as necessary. (*Id.*, 6:64-65; *id.* 8:62-66; INTEL-1003, ¶ 62.)

2. Kao Disclosed a DMA Engine and DMA Techniques for Booting an Embedded System

Kao described an "embedded system 100" (INTEL-1004, 3:47-48, 4:34) that included a host processor 102, DRAM device 110 including DMA engine 120, and a nonvolatile NAND flash memory 130 that stored boot data (including boot codes, controller codes, and OS/Application codes) for the embedded system. (*Id.*, 3:46-51, 3:60-4:3, 1:17-21, Fig. 1.)

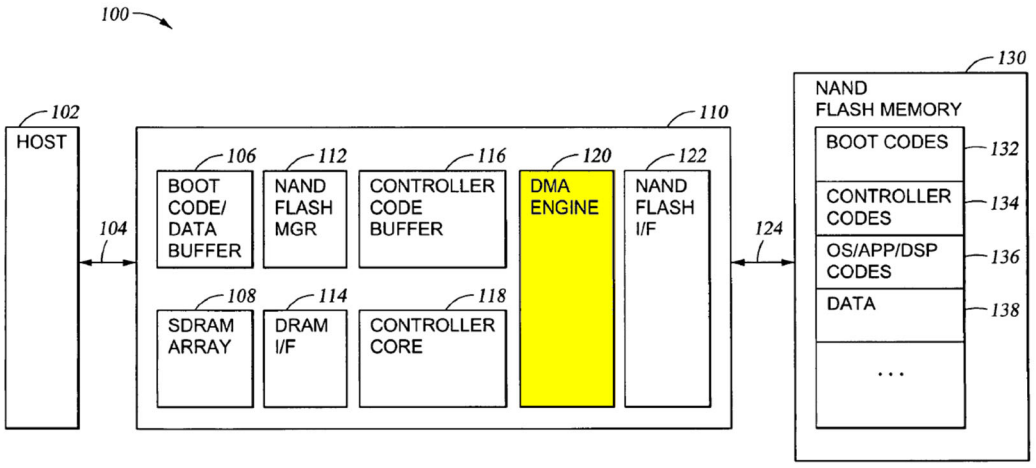


Fig. 1

(INTEL-1003, ¶ 65.)

Kao described using the DMA engine instead of the host processor to copy boot data from a nonvolatile NAND flash memory to a boot code buffer. (*Id*., 8:26-29, 8:39-45.) When the system was powered up, DMA engine circuitry loaded boot codes (and other controller or OS codes) from NAND flash memory 130 into boot code buffer 106. (*Id*., 4:61-66.) Kao taught that this DMA transfer used techniques "known to those skilled in the art" that resulted in "automatic transfer of the boot code … without external control or input, e.g., by the host 102 or any external controller circuitry." (*Id*., 4:66-5:3.) After the boot codes were copied from the NAND flash memory, the DMA circuitry copied desired code into SDRAM 108. (*Id*., 4:24-30.) (INTEL-1003, ¶¶ 66-70.)

Kao taught that by using the DMA engine to load boot codes, workload that would otherwise have been performed by the processor was offloaded. (INTEL-1004, 3:3-7.) This "reduc[ed] the complexity (and therefore the overall cost) of the host processor and the embedded system." (INTEL-1004, 3:7-9; INTEL-1003, ¶¶ 67-68.)

3. Wilson Disclosed Techniques for Initializing Registers of an Embedded System During a Boot Process

Wilson described techniques for booting a "microcontroller [that] is used to control a portable device, such as, for example a mobile phone featuring a multifunction touch screen." (INTEL-1013, 3:35-37.) Wilson's microcontroller included a processor 101, volatile memory 103, and a "[r]egister file 102 …. [with]

various registers which are used by used by the processor 101 in the ordinary operation of the microcontroller." (*Id*., 2:53-56, Fig. 1.) Wilson's microcontroller also included RLA engine 104 that "decompress[ed] [compressed boot data] into various register initialization values" and "load[ed] these values into their respective registers." (*Id*., 6:4-7, 5:47-51, Fig. 1, Fig. 4; INTEL-1003, ¶ 63.)

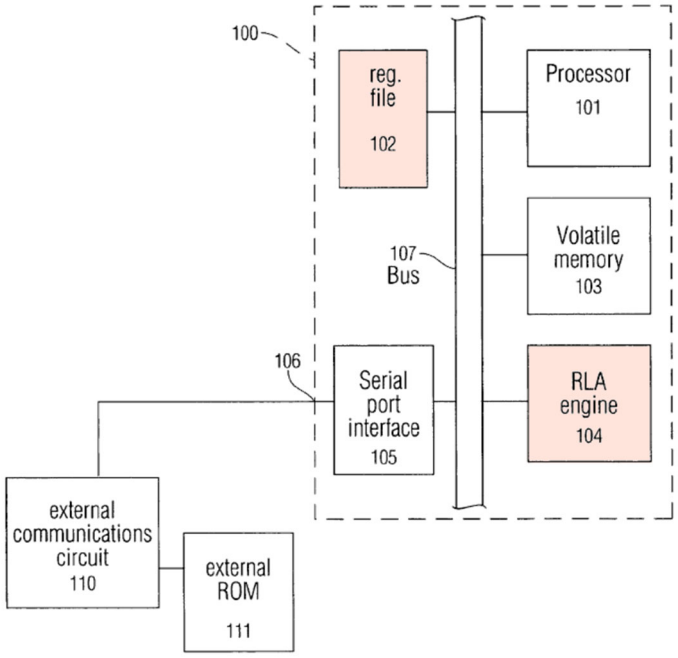


Fig. 1

Wilson taught that "certain specific predefined initial values" were required to be "stored in one or more of the registers of the register file before the microcontroller assumes ordinary operation." (*Id*., 2:56-59; *id*., 4:59-61.) These initial register values were thus "part of boot data that [wa]s loaded in the microcontroller during a

boot-up procedure" and "stored in the register file as part of an initialization or boot operation." (*Id.*, 2:60-63; INTEL-1003, ¶¶ 63, 66.)

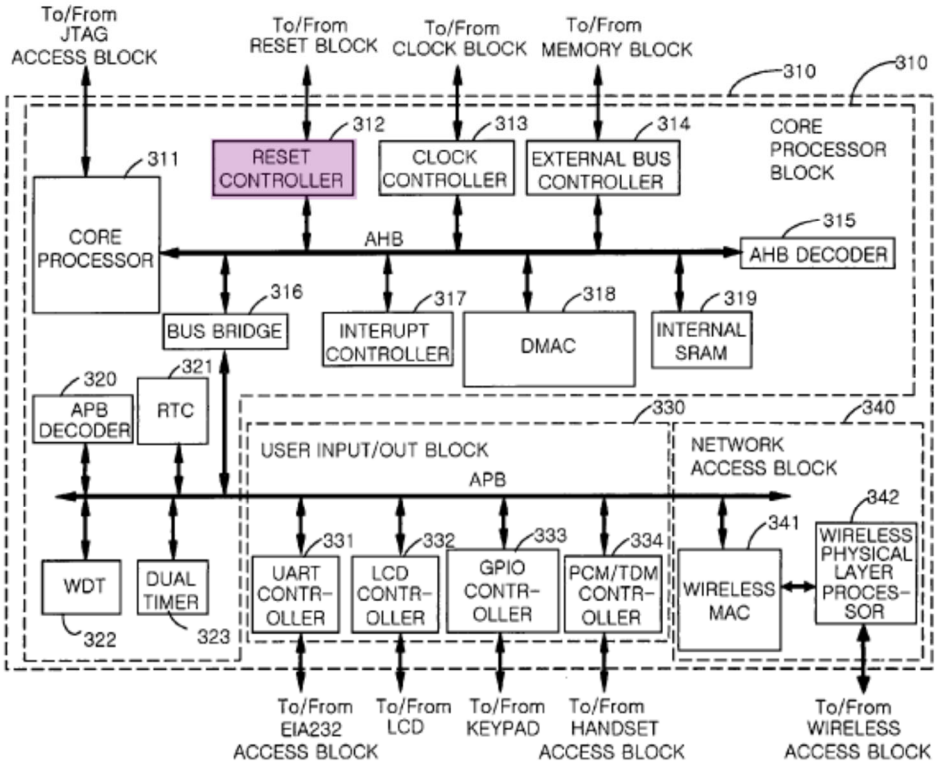
Wilson described a process for copying initial register values, along with other boot data, from external ROM to volatile memory 103. A "set of compressed boot data" stored in nonvolatile memory (such as ROM 111) was sent to the microcontroller in a single block. (*Id.*, 5:31-38, 5:46, Fig. 4.) The block included a header identifying the block as a boot data block and a data field with boot data. (*Id.*, 5:46-50, Fig. 4.) The boot data included initial register values and, optionally, initial instructions and memory values. (*Id.*, 6:42-44; INTEL-1003, ¶¶ 64, 65.)

The microcontroller received the data block and "stor[ed] data field 402 of the data block in volatile memory." (*Id.*, 5:56-61.) A Register Load Assist ("RLA") engine, implemented as a generally programmable processor, decompressed the boot data "into various register initialization values." (*Id.*, 6:3-7, 6:13-20.) The RLA engine then loaded "these values … into their respective registers" by sending each register value to the microcontroller's register file with instructions to load them into registers. (*Id.*, 6:4-7, 7:4-18, Fig. 5; INTEL-1003, ¶ 65.)

4. So Disclosed a Reset Controller within an SoC for an Internet Phone that Generated Reset Signals

So described an apparatus for developing and verifying a system-on-chip ("SoC") for an Internet phone. (INTEL-1021, Abstract.) Figure 3 illustrated the SoC. (*Id.*, 2:37-39, 3:53-55, Fig. 3; INTEL-1003, ¶ 71.)

FIG. 3



As shown in Figure 3, So's SoC included a "core processor block 310" that included a core processor 311, connected to an Advanced High-Performance Bus ("AHB"). Core processor block 310 further included reset controller 312 (highlighted in purple, above). (*Id*., 3:62-63.) So's reset controller was "connected to the external reset block" and "initializ[d]s all controllers according to the reset signal inputted from the external reset block, generate[d] the reset signal required in each controller and provide[s] the signal to the AHB." (*Id*., 4:4-8.; INTEL-1003, ¶ 72.)

5. Overview of Combinations

As and for the reasons described in detail below, the combinations of Post, Kao, and Wilson and Post, Kao, Wilson, and So disclosed all limitations of the challenged claims. In the resulting combination of Post, Kao, and Wilson, Post's control circuitry 112 would be modified to include Wilson's register file and circuitry to perform the register loading functionality of Wilson's RLA engine as well as a DMA controller that performed the boot data transfer functionality of Kao's DMA engine; and Post's boot data included initial register values, as taught by Wilson. In the resulting combination of Post, Kao, Wilson, and So, Post's SoC 110 would be further modified to include So's reset controller. (INTEL-1003, ¶ 73.)

B. Claim 1: the Combinations of Post with Kao and Wilson and Post with Kao, Wilson, and So render claim 1 obvious

1. Claim 1[pre]. An apparatus, comprising:

Intel does not concede that the preamble is limiting. To the extent it is, Post disclosed an "electronic device 100" (an "apparatus") such as "a cellular telephone (e.g., an iPhone[])," "a desktop computer, a laptop computer, and any other suitable type of electronic device or system." (INTEL-1005, 2:60-67.) Device 100 included a system-on-chip 110 (*id*., 3:1-2), which included control circuitry 112 that "control[led] the general operations and functions of SoC 110 and electronic device 100" and "include[d] any suitable components, circuitry, or logic, such as one or more processors." (INTEL-1005, 3:33-38; INTEL-1003, ¶ 75.)

2. Claim 1[a]. an embedded system comprising one or more processors, a reset controller, a storage device controller, one or more direct memory access (DMA) controllers, a random access memory (RAM) and a memory controller, a nonvolatile memory and a nonvolatile memory controller, a debug interface, and

An embedded system. The '694 patent describes an "embedded system" as including a system with one or more processors. (INTEL-1001, 1:26-28 ("embedded system will be used to refer to a single CPU or a multiprocessor system in this paper").) Post's system-on-chip ("SoC") 110 and nonvolatile memory 120 were collectively *an embedded system*. (INTEL-1003, ¶¶ 76-77; INTEL-1005, 3:33-38 (control circuitry 112 within SoC 110 "include[d] any suitable components, circuitry, or logic, such as one or more processors").)

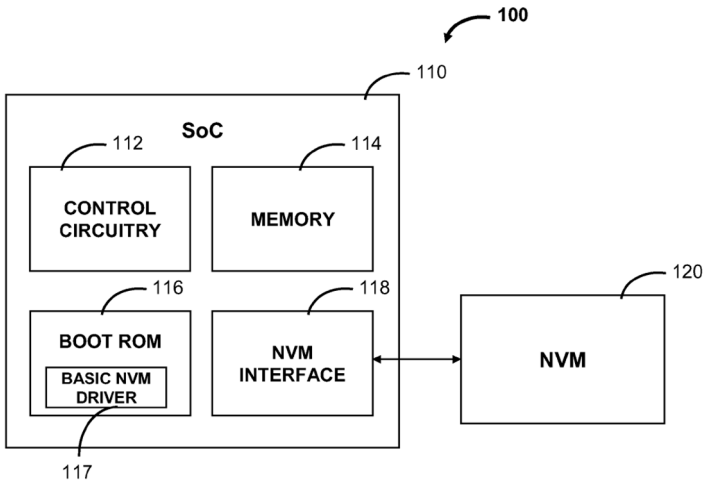


FIG. 1

(*Id*., Fig. 1.)

The remaining elements of limitation 1[a] are conventional embedded system components. (INTEL-1003, ¶¶ 79-102.) A POSITA would have understood that

Post's embedded system included these components, or would have found it obvious to incorporate these components into Post's SoC 110, in the manner and for the reasons discussed below.

Comprising one or more processors. Post disclosed that SoC 110 "include[d] control circuitry 112 … [which] control[led] the general operations and functions of SoC 110 and electronic device 100 in general." (INTEL-1005, 3:33-36, Fig. 1.) Control circuitry 112 included "one or more processors." (*Id*., 3:36-38 ("[c]ontrol circuitry 112 can include any suitable components, circuitry, or logic, such as one or more processors").) Post referred to control circuitry 112 interchangeably as "processor 112" (*id*., 4:5-6), and this Petition will do the same. (INTEL-1003, ¶ 80.)

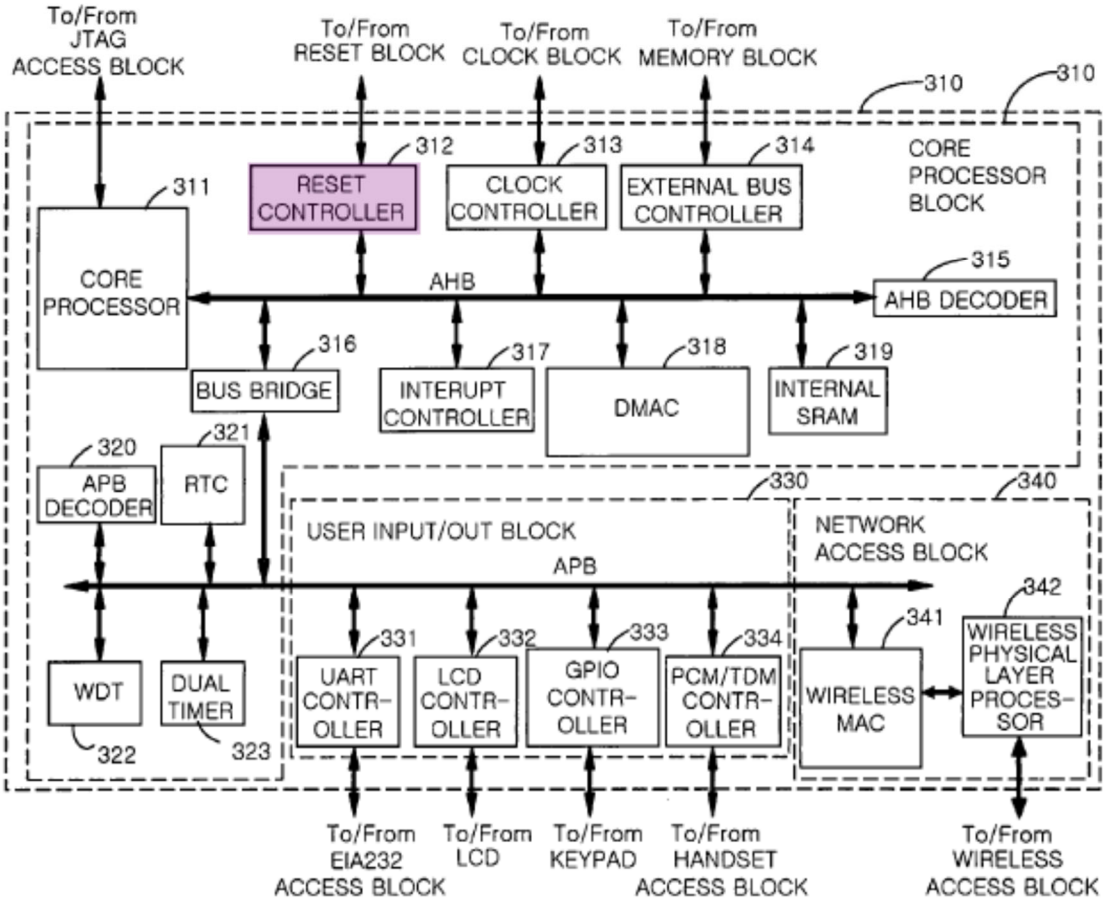
A reset controller. When a system is first powered on, components of the system are initially held in a reset state. (INTEL-1003, ¶ 81; INTEL-1004, 1:32-34 ("when the embedded system is powered up (e.g., when the embedded system enters a reset state)").) For example, the '694 patent describes a "process of releasing … CPUs 201 from their reset state" where each CPU is released from reset before executing system firmware. (INTEL-1001, 8:24-39, 3:61-67.)

Post described a process for booting up an electronic device in response to a "startup or reboot command" received by control circuitry 112. (INTEL-1005, 1:16-18, 8:15-27; *id*., 8:9-12, 9:7-11.) A POSITA would have understood Post's

disclosure of control circuitry 112 receiving a "startup or reboot command" to indicate the presence of a reset controller in SoC 110 that generated these commands. (INTEL-1003, ¶¶ 82-83; INTEL-1016, 555 ("System boot-up means that some type of power-on or reset source … has occurred."); *Realtime Data*, *LLC v. Iancu*, 912 F.3d 1368, 1372-73 (Fed. Cir. 2019) (affirming Board decision relying on general teachings in prior art as demonstrating POSITA's understanding of what prior art reference disclosed).) A POSITA would have understood power-on reset to be one of "[t]he most common microprocess-supervisory functions" (INTEL-1018, 1) and, at a minimum, would have viewed Post's disclosure of control circuitry 112 receiving startup or reboot commands as providing a suggestion to incorporate a conventional reset controller into SoC 110. (INTEL-1003, ¶ 83; INTEL-1008, 340 (illustrating an integrated reset controller).)

Additionally, So disclosed a reset controller 312 within a core processor block of a system-on-chip for an Internet phone (an "embedded system"). (INTEL-1021, 3:53-55, 3:62-63; *id*., Abstract; .)

FIG. 3



(*Id*., Fig. 3 (reset controller 312).) So disclosed that reset controller 312 "initialize[d] all controllers according to the reset signal inputted from the external reset block, generate[d] the reset signal required in each controller and provide[d] the signal to the [high-performance bus]." (*Id*., 4:4-8; INTEL-1003, ¶¶ 84-85.)

A POSITA would have found it obvious to modify Post in view of So's teachings, such that Post's SoC 110 included So's reset controller 312. So described a system on chip (SoC) for inclusion in a device like a phone, similar to Post's SoC 110 (which was included in, *e.g.*, an iPhone). (INTEL-1021, Abstract, 1:7-8, Figs.

1, 3; INTEL-1005, 2:60-64, 3:33-41, Fig. 1.) So taught a reset controller that was included in the SoC and that generated reset signals for different controllers in the SoC according to an external reset signal. (*Id*., 4:4-8.) Because Post described control circuitry 112 as receiving this type of signal at boot up (INTEL-1005, 8:25-27, 8:9-12), but was silent about how it was generated, a POSITA would have looked to So for implementation details. (INTEL-1003, ¶ 86.)

Additionally, a POSITA would have understood this combination to be obvious because So taught a known technique for solving a known problem (a reset controller for providing reset signals for initializing embedded system components); and incorporating a reset controller into Post's SoC 110 would have been within the skill of an ordinary artisan, who would be generally familiar with reset controllers (INTEL-1008, 340; INTEL-1016, 556), and because So showed that a POSITA knew how to do so for a SoC structurally similar to Post's SoC. (INTEL-1003, ¶ 87.) *Intel Corp. v. PACT XXP Schweiz AG*, -- F.4th --, 2023 WL 2469631, *5 (Fed. Cir. Mar. 13, 2023) ("It's enough [for obviousness] … to show that there was a known problem … in the art, that [the cited prior art teaching] helped address that issue, and that combining the teachings of [the cited art] wasn't beyond the skill of an ordinary artisan").

A storage device controller. Post disclosed that SoC 110 included a nonvolatile memory interface 118, which included "any suitable circuitry or

components that enable[d] SoC 110 to access NVM 120 and its memory locations." (INTEL-1005, 4:14-16.) Post further disclosed that these nonvolatile "memory locations" could include "multiple integrated circuits." (*Id*., 3:12-13.) A POSITA would have understood Post's disclosure that NVM interface 118 included "circuitry or components" that "enable[d] SoC 110 to access" the memory locations of nonvolatile memory 120's distinct circuits to indicate that NVM interface 118 included a "storage device controller." (INTEL-1003, ¶ 88; INTEL-1001, 2:40-41 ('694 patent describing "a storage device controller which provides interface to the storage device").)

One or more direct memory access (DMA) controllers. Kao disclosed a "DMA engine 120" within embedded system 100. (INTEL-1004, 3:46-51, 3:61-65; *id*., 1:17-24, 3:22-37.)

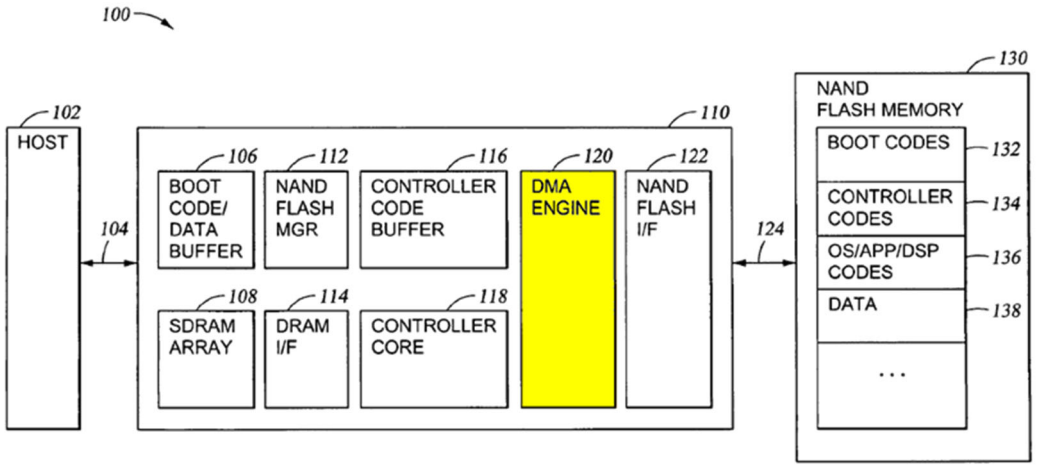


Fig. 1

(*Id*., Fig. 1.) Kao taught that "the source address and size of the boot code [to be loaded] may be provided" to DMA engine circuitry 120 by, for example, an external device. (*Id*., 5:12-14.) DMA engine circuitry 120 then automatically loaded boot code from nonvolatile NAND flash memory 130 to a RAM memory device 110. (*Id*., 4:20-26 ("[T]he DMA engine circuitry 120 may be used to automatically load boot codes 132 from the boot code images 232 into the boot code buffer 212" of DRAM memory device 110); *id*., 5:5-7 ("The DMA engine circuitry 120 may utilize the NAND flash interface 122 … to access the NAND flash memory 130.").) Kao's DMA engine circuitry 120 was thus a "DMA controller." (INTEL-1003, ¶¶ 89-90.)

A POSITA would have found it obvious to combine Post with Kao's teachings of using a DMA controller and DMA techniques to transfer boot data from nonvolatile memory to RAM; in the resulting combination, Post's control circuitry 112 would be modified to include a DMA controller (*e.g*., Kao's DMA engine 120) that performed Post's described functions of, at least, transferring boot data pages subsequent to the first identified page from nonvolatile memory 120 to memory 114. (INTEL-1003, ¶ 91.) A POSITA would have understood Kao to provide a teaching, suggestion, or motivation to make this combination because Kao taught that using DMA techniques to perform memory transfers offloads work from the processor, thus reducing the processor's workload in booting an embedded system and reducing system complexity and cost. (INTEL-1004, 4:31-34 ("by offloading work from the

host 102 to the DMA engine 120 and embedded microcontroller 118 in the DRAM memory device 110, design and operation of the host 102 and embedded system 100 may be simplified"), 3:3-9 ("By utilizing the DMA engine located on the volatile memory chip to load boot and controller codes, the processing workload necessary to boot the embedded system may be offloaded from a host processor to the volatile memory chip, thereby reducing the complexity (and therefore the overall cost) of the host processor and the embedded system."), INTEL-1003, ¶ 91.)

Kao's teachings regarding the recognized benefits of DMA were also consistent with other teachings in the art indicating that DMA techniques improved embedded system performance by offloading workload from the core processor. (INTEL-1009, 214 ("you can only achieve optimum performance in your application if data can move around without constantly bothering the core to perform the transfers [between internal and external memory]. This is where a direct memory access (DMA) controller comes into play."), *id*. ("[I]t is much more efficient to employ a DMA controller that operates independently from the core.").) A POSITA would have understood reducing processor workload to be a universal motivation within the field of embedded system design, and this would have further motivated a POSITA to combine Post with Kao's teaching of using a DMA controller to perform transfers. (INTEL-1003, ¶ 92.) *Intel*, 2023 WL 2469631, *4 ("universal motivations known in a particular field to improve technology provide a motivation

to combine prior art references *even absent any hint of suggestion* in the references themselves") (quotations omitted, emphasis in original).

A POSITA would also have had a reasonable expectation of success in making this combination. Post taught that its control circuitry 112 "can include any suitable components, circuitry, or logic" (INTEL-1005, 3:36-38) and Kao demonstrated that a POSITA would have known how to implement DMA transfers of boot data in an embedded system (Kao's) that was similar to the system of Post. (INTEL-1004, 5:25-32, 4:20-24, 10:28-50; *id*., 1:17-24 (embedded system "typically contains a computer processor … [that] include[s] a central processing unit (CPU) … or direct memory access (DMA) data transmission device"); INTEL-1003, ¶¶ 93-94.

Additionally, the combination of Post with Kao's teachings would have been obvious because Kao disclosed that its DMA controller and techniques had been used to improve one device (Kao's embedded system 100, which included one or more processors and NVM 130 (INTEL-1004, 3:22-27, 3:50-51)), and a POSITA would have recognized that Kao's DMA controller and techniques would improve Post's similar device (embedded system component SoC 110, which included one or more processors and NVM (INTEL-1005, 3:33-38, 2:60-3:2)) in the same way (by offloading processor workload and reducing processor complexity and cost). *See KSR Int'l Co. v. Teleflex Inc*., 550 U.S. 398, 417 (2007) ("[I]f a technique has been used to improve one device, and a person of ordinary skill in the art would

recognize that it would improve similar devices in the same way, using the technique is obvious unless its actual application is beyond his or her skill."). (INTEL-1004, 4:31-34, 3:3-9; INTEL-1009, 214; INTEL-1003, ¶ 94.) Additionally, both Post and Kao addressed the problem of transferring boot data from NVM during boot of an embedded system, and Kao's DMA controller and DMA techniques were a known way to address that problem. *Intel*, 2023 WL 2469631, at *5 ("That [prior art references] Kabemoto and Bauman address the same problem and that Bauman's cache was a known way to address that problem is precisely the reason that there's a motivation to combine under *KSR* and our precedent.").

Further, a POSITA would have viewed the combination of Post and Kao as involving the simple substitution of one known element (DMA controller and DMA techniques, as suggested by Kao) for another (using the core processor to transfer boot data from nonvolatile memory to RAM, as taught by Post) and would have understood that Post's processor-based system was ready for the improvements of reduced processor workload, system complexity, and cost. *KSR*, 550 U.S. at 417 ("simple substitution of one known element for another or the mere application of a known technique to a piece of prior art ready for the improvement" is consistent with obviousness). (INTEL-1004, 4:31-34, 3:3-9; INTEL-1009, 214; INTEL-1003, ¶ 94.) Thus, the combination of Post and Kao disclosed an embedded system

comprising "one or more DMA controllers," and a POSITA would have found this combination obvious.

A random access memory (RAM) and a memory controller. Post disclosed that SoC 110 included memory 114 (INTEL-1005, 3:33-34), which was "a random access memory (RAM)." "Memory 114 can include any suitable type of volatile memory, such as random access memory (RAM)." (INTEL-1005, 3:44-45; *id*., Fig. 2.)) Post disclosed that memory 114 acted as "the main memory for any processors implemented as part of control circuitry 112" and that control circuitry 112 wrote data to, and read data from, memory 114. (*Id*., 3:50-51; *id*., 3:55-57, 4:5-8; INTEL-1003, ¶ 95.) Given this disclosed interoperability between control circuitry 112 and memory 114, a POSITA would have understood that SoC 110 included a memory controller, and at a minimum, Post suggested including a memory controller in SoC 110, to enable control circuitry 112 to interface with memory 114 in the manner described. (INTEL-1009, 110 ("A memory controller (MEMC) is used to implement and provide glueless interfaces to the different types of memory in the system, such as cache, SRAM, and DRAM, synchronizing access to memory and verifying the integrity of the data being transferred."); INTEL-1003, ¶ 96.)

A nonvolatile memory and a nonvolatile memory controller. Post disclosed nonvolatile memory 120. (INTEL-1005, 3:1-2 ("Electronic device 100 can include … non-volatile memory ("NVM) 120."); *id*., Fig. 1.) Post further disclosed that

nonvolatile memory 120 included a "memory controller" such as a "flash controller." (*Id*., 4:19-23 ("some of the components and functions of NVM interface 118, as well as some of the functionality of any NVM drivers, may be implemented and performed by a separate memory controller (e.g., flash controller) included, for example, in NVM 120.").) A POSITA would have understood Post's disclosure of a flash controller to be a nonvolatile memory controller, and a POSITA would have understood Post's disclosure of a flash controller that performed the functionality of a nonvolatile memory driver and was included in nonvolatile memory 120 (*id*.) to be an example of a "nonvolatile memory controller." (INTEL-1003, ¶ 97; INTEL-1009, 207 ("The two main types of nonvolatile Flash memory widely used today are NOR-based and NAND-based").)

A debug interface. Post disclosed that control circuitry 112, within SoC 110, performed a "firmware update as part of a restore process" that caused boot data in NVM to be "changed or updated." (INTEL-1005, 10:60-11:3; *id*., Fig. 6 & 2:54-56 (showing "process for storing boot data in a non-volatile memory"); *id*., 8:7-12 (process of FIG. 6 "performed by … control circuitry 112 of FIG. 1"); INTEL-1003, ¶¶ 98-99.) Control circuitry 112 first identified a particular nonvolatile memory out of "a variety of potential NVMs." (*Id*., 11:4-10; INTEL-1003, ¶ 100.) Control circuitry 112 then identified "a boot data storage area in the non-volatile memory" and a subset of pages within that boot data storage area. (*Id*., 11:21-30.) Then,

control circuitry 112 "store[d] (or [did] not store) boot data in each page of the boot data storage area" by determining if a current page was within the selected subset of pages; "prepar[ing] some of the boot data for programming into the page" by "creat[ing] a data packet that includes boot data"; and "program[ming] the current page with the data packet." (*Id*., 11:41-56.) Following these steps, "all of the boot data may be stored in the non-volatile memory." (*Id*., 11:62-63; INTEL-1003, ¶ 101.) The '694 patent describes a "debug interface" as including, for example, a component "used to preprogram the nonvolatile memory and storage device with data and code." (INTEL-1001, 2:43-45.) Thus, SoC 110 included a "debug interface." (INTEL-1003, ¶ 102.) At the very least, a POSITA would have understood Post's disclosure of "changing or updating" stored boot data to suggest the inclusion of a debug interface (like a JTAG port) in Post's SoC 110 because these components were commonly included in processors for enabling "real-time debugging of hardware" (INTEL-1008, 157-59) and because a POSITA would have understood this to provide a means of implementing the "changing or updating" boot data functionality. (INTEL-1003, ¶ 102.)

3. Claim 1[b]. a power-on reset (POR) sequencer configured to detect, in the nonvolatile memory, a descriptor used in a boot process and configured to copy the descriptor from the nonvolatile memory to the RAM,

A power-on reset (POR) sequencer. Post's control circuitry 112, as modified to include Wilson's register file and circuitry to perform the register loading functionality of Wilson's RLA engine, facilitated a boot process for electronic device 100 after power-on and was a "POR sequencer." (INTEL-1003, ¶¶ 103-106.) Post disclosed that when electronic device 100 was powered on, control circuitry 112 loaded "firmware, such as a first stage bootloader" that was used to "load a second stage bootloader or the kernel and operating system of electronic device 100." (INTEL-1005, 3:55-65; 1:16-19 ("When an electronic device is being powered up or rebooted, an operating system can be booted up from a NVM of the electronic device."); INTEL-1003, ¶ 107.) Once the operating system was loaded, "control of circuitry 112 [w]as transferred to the operating system." (*Id*., 4:5-7; *id*., 1:19-26; INTEL-1003, ¶ 108.) Post's Figures 4-6 illustrated boot processes performed by control circuitry 112. (*Id*., 8:9-12, 8:15-16, 9:14-15, 10:60-62.)

A POSITA would have found it obvious to include, in control circuitry 112, Wilson's register file and circuitry to perform the register loading functionality of Wilson's RLA engine. Wilson disclosed a microcontroller that included a processor

101, volatile memory 103, and a "[r]egister file 102 …. [with] various registers which are used by the processor 101 in the ordinary operation of the microcontroller." (INTEL-1013, 2:53-56, Fig. 1; INTEL-1003, ¶ 109.) The microcontroller could be "custom designed to perform specific predefined functions" for, for example, a phone with a multifunction touch screen, and it included an "unorthodox register file" with registers for configuring the touch screen display and screen pixel colors. (*Id.*, 3:35-62; INTEL-1003, ¶ 110.) Wilson's microcontroller 100 was shown in Figure 1:

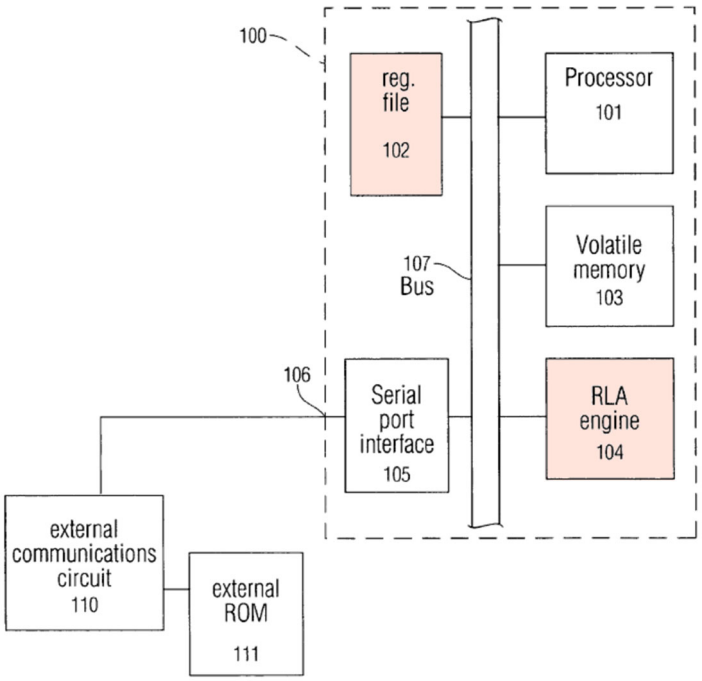


Fig. 1

(*Id.*, Fig. 1 (annotated); INTEL-1003, ¶ 109.)

Wilson taught that predefined initial values had to be stored in the registers of the register file before microcontroller 100 could assume ordinary operation. (*Id*., 2:56-59.) These initial register values were therefore included as "part of boot data that is loaded in the microcontroller during a boot-up procedure." (*Id*., 2:53-63; *id*., 5:1-4 ("The predefined [register] values can be sent from an element external to the microcontroller … to the microcontroller at time of boot up."); *id*., 5:30-45, 3:26-34; INTEL-1003, ¶ 111). Wilson's microcontroller included a Register Load Assist ("RLA") engine 104 that "decompress[ed] [compressed boot data] into various register initialization values" and "load[ed] these values into their respective registers" during the boot process. (*Id*., 6:4-7, 5:47-51, Fig. 1, Fig. 3; INTEL-1003, ¶ 112.)

A POSITA would have found it obvious to modify Post in view of Wilson's teachings, such that Post's control circuitry 112 included Wilson's register file and circuitry to perform the Register Load Assist functionality of the RLA engine. Wilson described a microcontroller "on a chip" similar to Post's SoC 110. (INTEL-1013, 1:11-17; INTEL-1005, 3:33-41, Fig. 1; INTEL-1003, ¶ 113.) Wilson taught that the boot-up process for this microcontroller was understood to involve "load[ing] initialization values in [the microcontroller's] registers" and that this was needed to "prepare the microcontroller for ordinary operation." (*Id*., 1:24-28.) As Post is silent about techniques for loading initialization values into its registers

during boot, a POSITA would have looked to Wilson for implementation details on how to do so. (INTEL-1003, ¶ 114.)

Additionally, both Post and Wilson described similar embedded systems: a SoC or microcontroller used to control a mobile phone. (INTEL-1005, 2:60-67, 3:33-36; INTEL-1013, 1:11-17, 3:35-37; INTEL-1003, ¶ 113.) Wilson taught the benefit of "custom design[ing]" a microcontroller to "perform specific predefined functions" like configuring screen colors or configuring a multi-touch panel processor and, in doing so, using "registers of unorthodox sizes [that] may be chosen as best suited" for those functions. (INTEL-1013, 3:35-62, 11:11-39.) Wilson further taught that initial values for those registers would be included as part of the boot data. (*Id*., 6:49-7:3.) A POSITA would have been motivated to similarly custom design Post's SoC 110, including control circuitry 112, to realize the same benefits of performing specific predefined functions for configuring a mobile phone, which required initialization of configuration registers as taught by Wilson. (INTEL-1003, ¶ 115.) Post taught that its control circuitry "can include any suitable components, circuitry, or logic" (INTEL-1005, 3:36-38), which a POSITA would have viewed as enabling and encouraging the incorporation of circuitry such as that described by Wilson. (INTEL-1003, ¶ 115.)

Further, a POSITA would have viewed Wilson as providing a teaching, suggestion, or motivation to include registers and Register Load Assist functionality

in Post's control circuitry 112 because Wilson taught that register values *must* be loaded before a microcontroller can begin normal operation. (INTEL-1013, 4:59-61 ("for microcontroller 100 to commence ordinary operation, it can be required that certain predefined values be loaded in the registers of register file 102"); INTEL-1003, ¶ 116.) Wilson also taught that including initial register information along with other boot data, such as compressed initial memory values and instructions, could provide the benefit of improving the speed of a boot process, which a POSITA would have viewed as a further teaching, suggestion, or motivation for this combination. (*Id*., 5:10-11, 5:25-26, 5:46-47, 6:42-44; INTEL-1003, ¶ 117.)

A POSITA would also have had a reasonable expectation of success in making this combination, because POSITAs were familiar with processor registers and techniques for initializing registers at system bootup. (INTEL-1016, 137, 156, 555; INTEL-1013, 1:24-28 ("Many microcontrollers require a boot-up process when initially powered up. The boot-up process allows a microcontroller to perform various actions to prepare the microcontroller for ordinary operation, such as load initialization values in its registers …"); INTEL-1003, ¶ 118.) For example, Wilson's teachings indicate that a POSITA would have known how to custom design a microcontroller, similar to Post's SoC, with configuration registers and circuitry for loading these registers with initial values included in the boot data, as required prior to normal operation. (INTEL-1003, ¶ 118.) Further, Wilson taught that its

RLA engine could be implemented by either a "generally programmable processor" or "an application specific circuit … within a larger application specific integrated circuit (ASIC)." (*Id*., 6:14-24.) Therefore, a POSITA would have been motivated in view of Wilson's disclosure to implement the circuitry to perform functionality of Wilson's RLA engine as an application specific circuit within control circuitry 112, and would have had a reasonable expectation of success in doing so. INTEL-1003, ¶ 118.)

Further, a POSITA would have been motivated to make this combination because Wilson's teaching of including compressed initial register values in boot data, and using the Register Load Assist circuitry to decompress those values and load the microcontroller's registers with those values, was a known technique for solving the known problem of how to initialize processor register information at bootup (INTEL-1013, 1:25-28, 2:54-63, 6:27-41, Figs. 1, 3; INTEL-1016, 555); and including register initialization information in Post's boot data would have been within the skill of an ordinary artisan, because they would be generally familiar with processor registers and processor boot techniques (INTEL-1009, 137; INTEL-1013, 1:25-28; INTEL-1005, 1:14-26), and because Wilson showed that a POSITA knew how to do so for a microcontroller similar to Post's SoC. (INTEL-1003, ¶ 119, 121.) The Federal Circuit has recognized that "nothing more is required to show a motivation to combine." *Intel*, 2023 WL 2469631, *5 ("It's enough … to show that

there was a known problem … in the art, that [the cited prior art teaching] helped address that issue, and that combining the teachings of [the cited art] wasn't beyond the skill of an ordinary artisan"); *id*. ("That [the cited prior art references] address the same problem and that [the cited prior art teaching] was a known way to address that problem is precisely the reason that there's a motivation to combine under *KSR* and our precedent").

Further, a POSITA would have recognized the systems of Post and Wilson to have various similarities, which would have prompted them to look to Wilson as related teachings. *KSR*, 550 U.S. at 420 ("in many cases a person of ordinary skill will be able to fit the teachings of multiple patents together like pieces of a puzzle"). Both Post and Wilson taught storing boot data of an embedded system in an external nonvolatile memory. (INTEL 1005, 1:32-34; INTEL-1013, 1:57-60; INTEL-1003, ¶ 120.) In addition, Wilson, like Post, was directed to improving the boot process in an embedded system. (INTEL-1013, 2:4-8; INTEL-1005, 1:35-38; INTEL-1003, ¶ 120.) Further, Wilson, like Post, taught booting an embedded system within a mobile phone featuring a multifunction touch screen. (INTEL-1013, 3:35-37; INTEL-1005, 2:60-67; INTEL-1003, ¶ 120.)

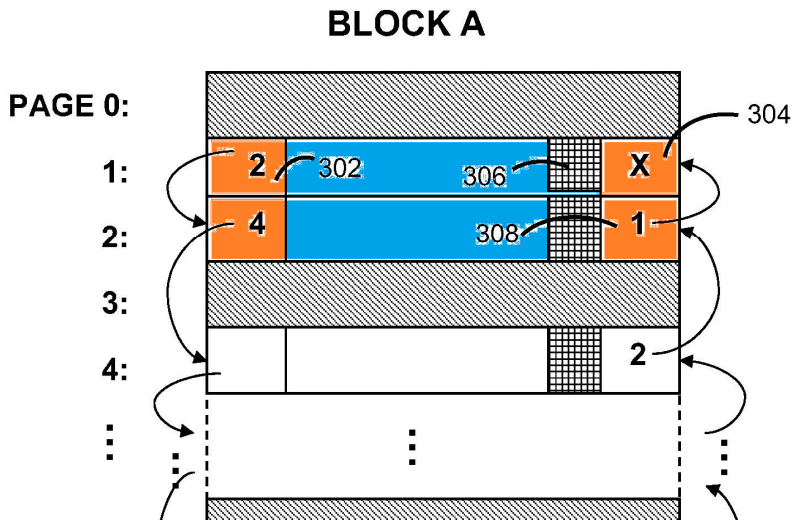
A POSITA would thus have found it obvious to combine Post with Wilson's teachings, with the resulting combination including Post's control circuitry 112 modified to include Wilson's register file and circuitry to perform the register

loading functionality of Wilson's RLA engine. A POSITA would have understood control circuitry 112, as modified, to be a "Power-on Reset Sequencer." (INTEL-1003, ¶ 121.)

Configured to detect, in the nonvolatile memory, a descriptor used in a boot process. Control circuitry 112 of the asserted combinations (*i.e.*, modified to include Wilson's register file and circuitry to perform the register loading functionality of Wilson's RLA engine control circuitry 120), detected boot data and associated pointer metadata (collectively, "*a descriptor used in a boot process*") in NVM 120 ("*in the nonvolatile memory*"). (INTEL-1003, ¶ 122.) At bootup of the electronic device, control circuitry 112 "load[ed] additional information from NVM 120 into memory 114." (INTEL-1005, 3:41-43, 3:53-57.) This "additional information" included "firmware, such as a first stage bootloader, that, once loaded into memory 114, is used by electronic device 100 to load a second stage bootloader or the kernel and operating system of electronic device 100." (*Id*., 3:57-63.) The "firmware (or other data) retrieved from NVM 120" by the control circuitry 112 was "referred to as 'boot data.'" (*Id*., 3:63-65; INTEL-1003, ¶ 123.)

The boot data was stored in a linked-list format in pages of NVM 120, along with metadata pointers that provided the address of next and prior boot data pages and were used to verify and retrieve additional boot data pages. (*Id*., 6:19-53; *id*., 6:22-24; *id*., 7:6-33, 8:9-12, 9:14-62.) Post's Figure 3 (excerpted below) illustrated

pages of nonvolatile memory 120, where boot pages 1 and 2 included both boot data (highlighted in blue) and next-page and prior-page pointer metadata (highlighted in orange):



(*Id*., Fig. 3; INTEL-1003, ¶ 124.)

In the asserted combinations involving Post and Wilson, a POSITA would have understood the boot data to include initial register values, as disclosed by Wilson. (INTEL-1013, 2:56-63 ("The microcontroller can require that certain specific predefined initial values are stored in one or more of the registers of the register file before the microcontroller assumes ordinary operation. … [T]hese initial values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure."), 6:50-54 ("Compressed boot data 500 is shown. … The compressed data may include a plurality of initial register values intended for the register file."); INTEL-1003, ¶¶ 125-26.) Because the combined system includes Wilson's register

file, which must be initialized with values before the microcontroller assumes ordinary operation, as well as functionality for *loading* register values into registers, a POSITA would have understood the boot data of the combined system to include these initial register values disclosed by Wilson. At a minimum, a POSITA would have found this obvious for the same reasons described in connection with claim 1[b]. (INTEL-1003, ¶¶ 125-26.)

Post disclosed searching for and identifying boot data and metadata ("*detecting, in the nonvolatile memory, a descriptor …*") in the nonvolatile memory. (INTEL-1005, 8:34-35, 7:34-50, 8:7-9:2; *id*., 6:22-24.) Control circuitry 112 "read an initial page from the non-volatile memory" and used various techniques to "determine whether the initial page contains boot data." (*Id*., 8:29-31, 8:34-42.) If boot data was not found on a particular page, "the electronic device may read the next page … so that the electronic device can determine whether storage of the boot data begins at a subsequent page." (*Id*., 8:43-49.) Control circuitry 112 then "load[ed] the rest of the boot data into memory at step 422. For example, the rest of the boot data may be loaded using the steps of process 500 of FIG. 5." (*Id*., 9:7-10.) Further, control circuitry 112 detected scenarios in which the first page of boot data could not be located (*id*., 7:34-50, 8:52-9:2) and, in response, attempted to detect another copy of the boot data elsewhere in the non-volatile memory (*id*., 4:36-44, 7:44-50). (*Compare* INTEL-1001, 6:64-7:12 ('694 patent describing that "[i]f the

first copy [of the POR sequencer descriptor] is unsuccessfully detected in step 604, in block 609, the POR sequencer will try to detect a second copy of the POR sequencer descriptor … [in] the nonvolatile memory."); INTEL-1003, ¶¶ 127-28.)

The boot data and metadata was also *used in a boot process*. As discussed, control circuitry 112 used pointer metadata to load and verify boot data and used boot data to load a second stage boot loader, kernel and operating system. (INTEL-1013., 3:55-61, 9:47-62.) After control circuitry 112 loaded the operating system, "control of processor 112 [w]as transferred to the operating system." (*Id*., 4:5-7; *id*., 1:19-26.) Post taught that "successful completion of this bootup procedure is typically critical for the electronic device to function properly." (*Id*., 1:26-28, 1:34-37; INTEL-1013, 2:56-63 (initial register values in boot data were "required" to be stored in registers of the register file before the microcontroller assumed ordinary operation). The '694 patent describes using the POR sequencer descriptor in a boot-up sequence in a similar manner. (INTEL-1003, ¶ 129.)

And configured to copy the descriptor from the nonvolatile memory to the RAM. Post disclosed that control circuitry 112 was configured to copy the boot data and associated pointer metadata (collectively, "*the descriptor*") *from nonvolatile memory* (NVM 112) *to the RAM* (memory 114). (INTEL-1003, ¶ 130.) Post disclosed that control circuitry 112 copied boot data from nonvolatile memory 120 into memory 114. (INTEL-1005, 3:53-65 (control circuitry copied boot data

from NVM 120 to memory 114), 5:13-23 (boot data was copied from NVM 120 to memory 114); *id*., 6:28-29, 5:43-47, Fig. 2.) Memory 114 was random access memory. (*Id*., 3:44-49 ("Memory 114 can include any suitable type of volatile memory, such as random access memory (RAM)").)

Post also disclosed that pointer metadata from each boot page was saved during the process of loading boot data into memory: "If the current page [of boot data] is not the last page, process 500 may instead continue to step 516, where the electronic device can ***save the current page's next page pointer for use in processing subsequent pages***." (*Id*., 9:15-18, 9:56-60; *id*., 8:9-12, 9:7-11.) The saved pointer data was used to determine whether later pages of the nonvolatile memory included boot data and should therefore be copied:

> [A]t step 517, the electronic device can refer to ***a next page pointer, which was saved from a previous boot data page*** (e.g., from a previous iteration of step 516). The electronic device can determine whether this saved next page pointer references a subsequent page (and not the current page). If the electronic device determines that the saved next page pointer references a different page, and therefore the current page should not contain boot data, process 500 can return to step 504 without maintaining information from the current non-data page.

(*Id*., 9:65-10:7; *id*., 5:59-64 ("A stored map may indicate which memory locations are currently storing boot data … the mask ROM code from a boot ROM (e.g., boot ROM 116 [within electronic device 110]) can be designed to use the map when locating the boot data").) A POSITA would have understood Post's references to

"saving" the pointer metadata to mean storing the metadata in on-chip volatile memory such as RAM 114, or at least was suggestive of doing so, because the other boot data was stored in RAM and because storage in on-chip RAM would have allowed the pointer data to be accessed quickly during the boot loading process. (INTEL-1003 ¶¶ 131-32.) Indeed, Post described embodiments in which *all* data from the boot storage area was stored together in memory 114. (INTEL-1005, 5:53-57 ("[T]he electronic device can first ***load the contents of all of the pages in the boot data storage area*** (including the non-data pages) ***into memory 220***. The loaded content may then be post-processed to discard the unneeded information obtained from the non-data pages."), 5:19-20 ("memory 220 may illustrate the state of memory 114 (FIG. 1)").)

4. Claim 1[c]. wherein the POR sequencer facilitates a boot process and verifies an integrity of loaded data and code in the RAM,

Wherein the POR sequencer facilitates a boot process. Post disclosed that control circuitry 112 facilitated a boot process by executing processes for "initiating boot up of an electronic device," "loading boot data into memory," and "storing boot data in a reprogrammable non-volatile memory." (INTEL-1005, 8:15-16, 9:14-15, 10:60-62, Figs. 4-6; *id*., 8:7-12; INTEL-1003, ¶¶ 133-37.)

Control circuitry 112 used pointers to fetch boot data and associated metadata stored in NVM 120 and copy it to system RAM. Figure 5 showed the process "for

loading boot data into memory." (INTEL-1005, 9:14-15.) This began with "the electronic device [reading] a page from the nonvolatile memory" and "determin[ing] whether the current page contains boot data." (*Id*., 9:19-26; *id*., Fig. 5 (step 504).)

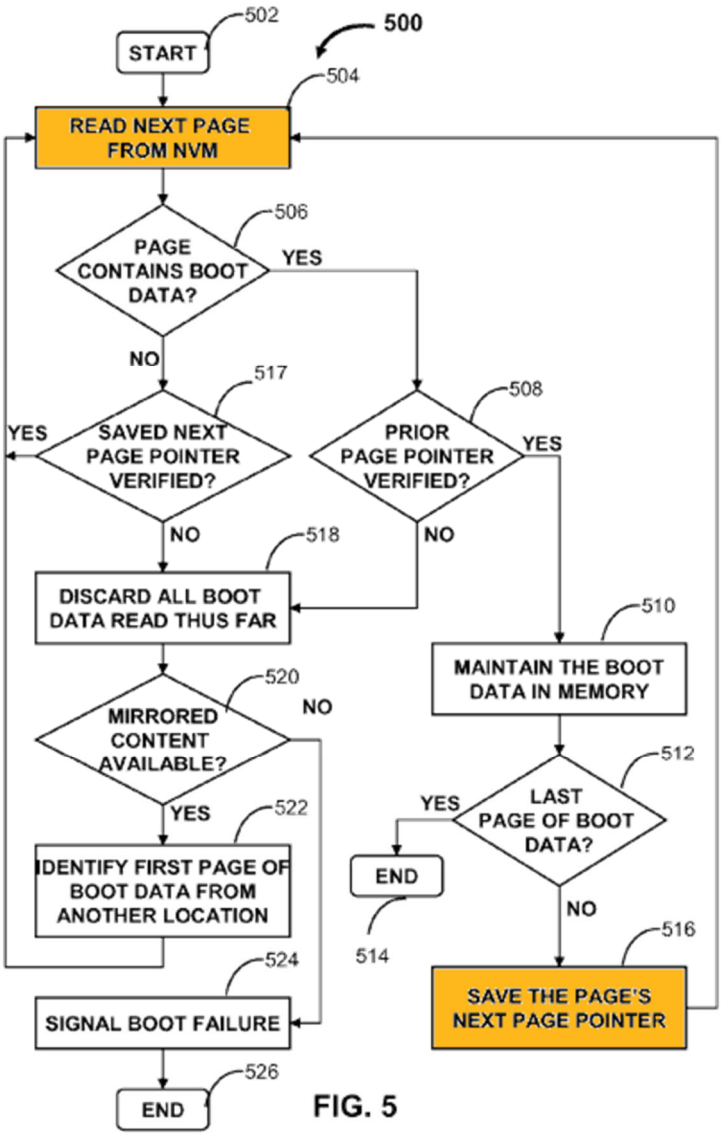


FIG. 5

Post's device "examine[d] the next page pointer" of the current boot data page "to determine whether the pointer reference[d] a subsequent page of the non-voltage memory." (*Id*., 9:47-52.) If the pointer referenced another boot data page, the device

"save[d] the current page's next page pointer for use in processing subsequent pages." (*Id*., 9:56-60.) The device then "return[ed] to step 504 [i.e., reading a page from the non-volatile memory] to obtain the rest of the boot data." (*Id*. 9:60-62.) Post described iterating through this process until "the pointer indicates that there are no more pages of boot data," at which point "all of the boot data [was] stored in memory." (*Id*., 9:49-55.)

And verifies an integrity of loaded data and code in the RAM. Control circuitry 112 *verified an integrity* of the boot data that was loaded into memory 114 ("*loaded data and code in the RAM*") by using one or more of next page pointers 302, prior page pointers 304, and boot data page indicators 306. "[T]he electronic device can use the next page pointers to verify the integrity of boot data storage." (INTEL-1005, 6:54-55; *id*., 7:21-23 ("As with the next page pointers, the prior page pointers may allow an electronic device to verify and account for all of the boot data stored in the NVM."), 7:58-64 (boot data page indicators 306 "provide another way in which the electronic device can verify that a page is a boot data page"), 1:55-2:3 (device used "pointers … [to] load the boot data and verify that pages expected to have boot data actually contain boot data (and vice versa)"); *id*., 6:22-24, 8:52-66, 9:33-46, 10:52-54; INTEL-1003 ¶¶ 138-39.)

Post also disclosed that control circuitry 112 used error correcting codes to verify an integrity of the boot data that was loaded in memory 114. (*Id*., 6:57-59

("The electronic device may confirm that page 2 provides boot data by, for instance, employing an error correcting code on the contents of page 2.").) A POSITA would have understood that these functions were performed by control circuitry 112, and would also found this obvious, because Post described control circuitry 112 as verifying the integrity of boot data that was loaded from nonvolatile memory using pointers and error correcting codes. (*Id*., 9:25-46, 8:9-12, 3:34-36; INTEL-1003 ¶¶ 140-41.)

5. Claim 1[d]. wherein the descriptor comprises a POR sequencer descriptor, and wherein the POR sequencer uses the POR sequencer descriptor which is a preassembled descriptor that is stored in the nonvolatile memory,

Wherein the descriptor comprises a POR sequencer descriptor. The boot data (including initial register values) and associated metadata of the asserted combinations facilitated a boot process and were thus a *POR sequencer descriptor*. Post taught that at system bootup, control circuitry 112 "load[ed] additional information from NVM 120 into memory 114" that included "firmware, such as a first stage bootloader," which was used to "load a second stage bootloader or the kernel and operating system of electronic device 100." (INTEL-1005, 3:55-65; *id*., 8:15-24 & Fig. 4 (showing process for "initiating bootup of an electronic device" by "identify[ing] which memory location of a non-volatile memory contains the first page of boot data"); *id*., 9:14-18 & Fig. 5 (showing process for "loading boot data

into memory" after first boot data page identified and loaded).) Wilson further taught that initial register values included in the boot data were loaded into registers during the boot process. (INTEL-1013, 2:56-63 ("[T]hese initial [register] values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure."); *supra* Sec. VI. B.3 (describing asserted combinations' usage of boot data to facilitate boot process in connection with claim 1[b]).)

A POSITA would thus have understood the boot data and metadata of the asserted combinations to be a "*POR [power-on reset] sequencer descriptor*." (INTEL-1003 ¶ 142.) The '694 similarly describes the POR sequencer descriptor as facilitating a boot process. (INTEL-1001, 2:47-50 ("The POR Sequencer makes use of a POR Sequencer Descriptor which is a preassembled descriptor that is stored in a nonvolatile memory of the system."), Abstract ("The POR sequencer controls the overall boot process of the embedded system. Said sequencer uses descriptors (POR Sequencer descriptors)[.]").)

And wherein the POR sequencer uses the POR sequencer descriptor. In the asserted combinations, the *POR sequencer used* boot data (part of "*the POR sequencer descriptor*") to load a second stage bootloader and device kernel and operating system (INTEL-1005, 3:57-61) and load values into registers (INTEL-1013, 6:4-7, 7:6-18). (INTEL-1003 ¶ 142.) The POR sequencer also *used* metadata pointers (also part of "*the POR sequencer descriptor*") to fetch each fragment of

boot data stored in NVM 120 and copy it to memory 114. Figure 5 showed the process "for loading boot data into memory." (INTEL-1005, 9:14-15; *id*., 8:7-12 (Fig. 5 process performed by control circuitry 112).) Control circuitry 112 first "read a page from the nonvolatile memory" and "determine[d] whether the current page contains boot data." (*Id*., 9:19-26; *id*., Fig. 5 (step 504).)

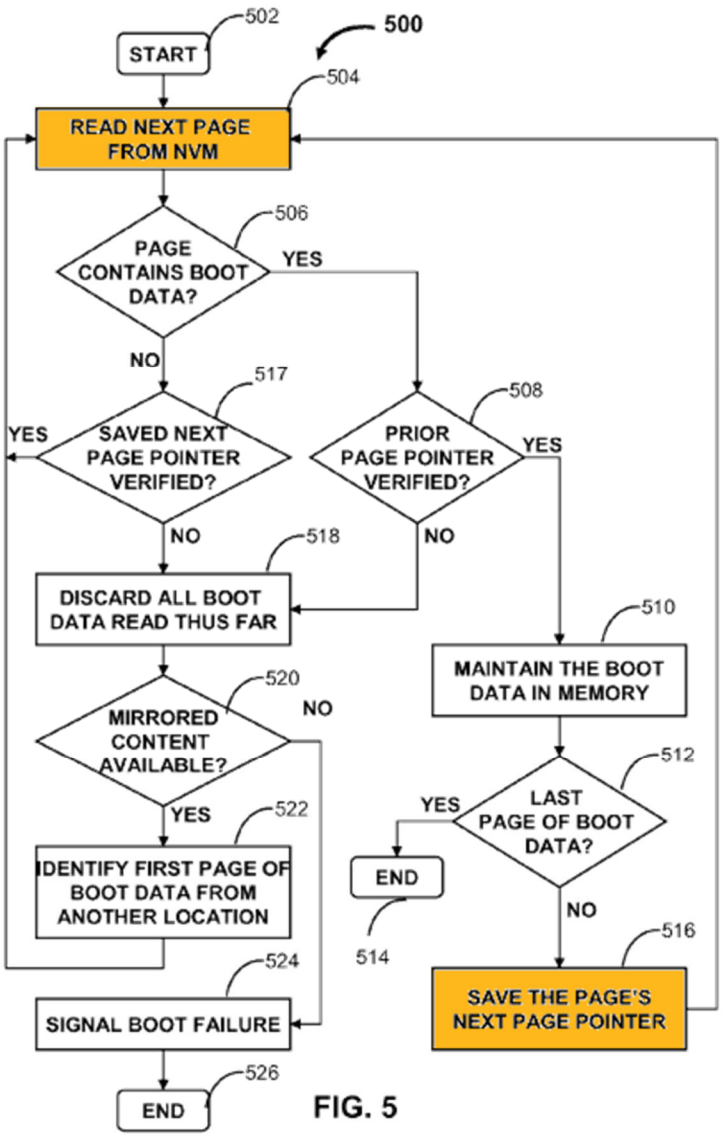


FIG. 5

Control circuitry 112 "examine[d] the next page pointer" of the current boot data page "to determine whether the pointer reference[d] a subsequent page of the non-volatile memory." (*Id*., 9:47-52.) If the pointer referenced another boot data page, circuitry 112 "save[d] the current page's next page pointer for use in processing subsequent pages." (*Id*., 9:56-60.) It then "return[ed] to step 504 [i.e., reading a page from the non-volatile memory] to obtain the rest of the boot data." (*Id*. 9:60-62.) Control circuitry 112 iterated through this process until "the pointer indicates that there are no more pages of boot data," at which point "all of the boot data [was] stored in memory." (*Id*., 9:49-55.)

… Which is a preassembled descriptor that is stored in the nonvolatile memory. Post taught storing boot data and metadata in a linked-list format in nonvolatile memory 120 ("*a preassembled descriptor that is stored in the nonvolatile memory*"). (INTEL-1005, 6:19-24, 4:46-47, 5:39-40; *id*., 1:57-64; (INTEL-1003 ¶¶ 143-44.) In this "linked-list format," each boot data page "store[d] both data … and metadata … includ[ing] a next page pointer 302 and a prior page pointer 304." (*Id*., 6:19-24, 6:28-31.) These pointers "reference[d] the next boot page in the sequence" by "provid[ing] the address" of the next page of boot data. (*Id*., 6:38-42.)

Post's Figure 3 (shown below, with boot data pages annotated in orange) illustrated this linked-list format of boot data and associated pointer metadata:

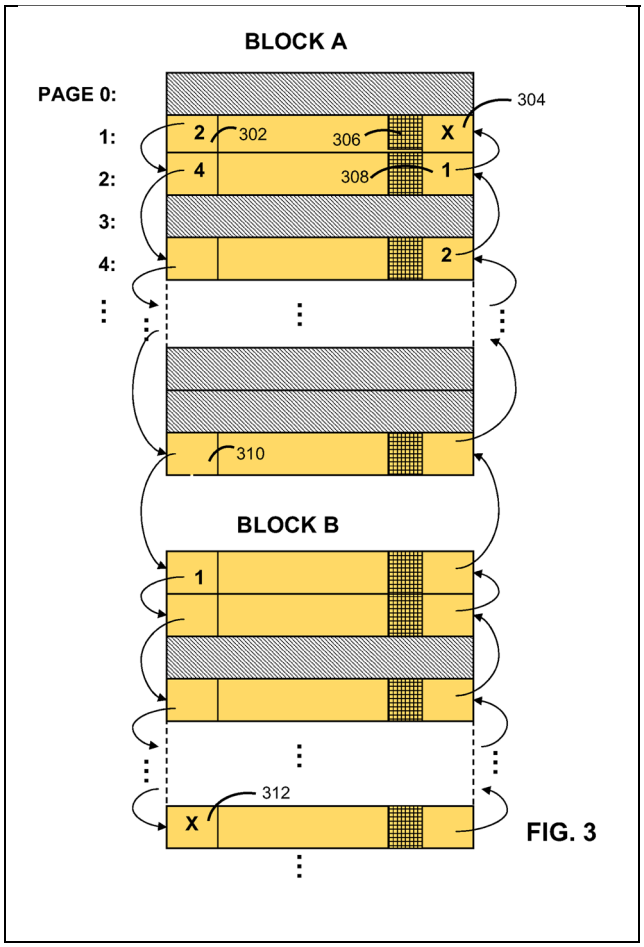


FIG. 3

(INTEL-1005, Fig. 3; *compare* INTEL-1001, Fig. 5 ('694 patent disclosing a similar linked-list storage of POR sequencer descriptor).) Wilson similarly disclosed storing boot data in nonvolatile memory. (INTEL-1013, 3:29-34.)

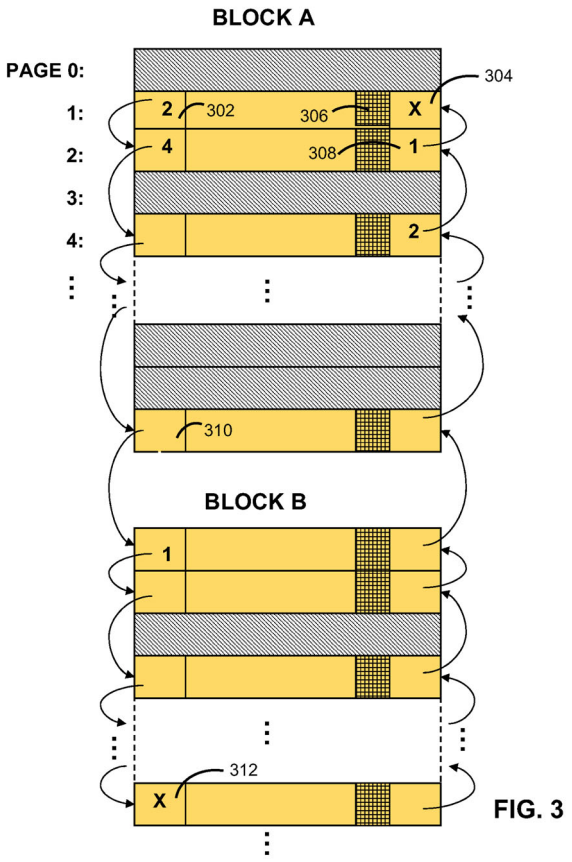
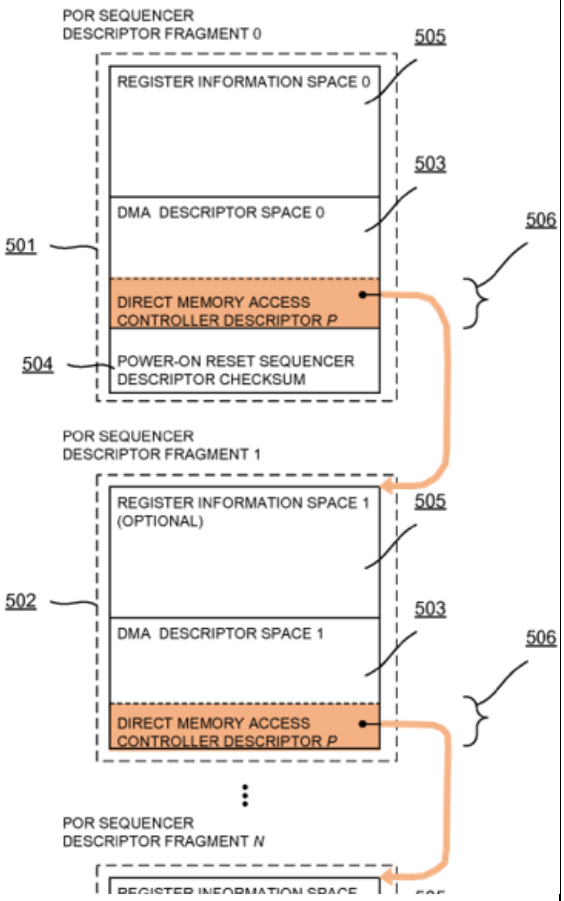
6. Claim 1[e]. wherein the POR sequencer descriptor includes register information and DMA controller descriptors, and

Wherein the POR sequencer descriptor includes register information. Wilson disclosed initial register values ("register information") included in the boot data. (INTEL-1013, 2:56-63 ("The microcontroller can require that certain specific

predefined initial values are stored in one or more of the registers of the register file before the microcontroller assumes ordinary operation. … [T]hese initial values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure."), 6:50-54 ("Compressed boot data 500 is shown. … The compressed data may include a plurality of initial register values intended for the register file.").) As discussed in connection with claims 1[b] and 1[d], a POSITA would have understood the boot data of the asserted combinations to include initial register values, as disclosed by Wilson, and it would also have been obvious for the reasons described. (*Supra* Sec. VI.B.3, .5; INTEL-1003, ¶ 145.)

… And DMA controller descriptors. A POSITA would further have understood, and found it obvious, that Post's next-page pointers (also part of the *POR sequencer descriptor*) would be *DMA controller descriptors* in the asserted combinations. (INTEL-1003 ¶¶ 146-50.) As discussed for claim 1[a], a POSITA would have found it obvious for Post's control circuitry 112 to include a *DMA controller* (*e.g.*, Kao's DMA engine 120) that performed Post's described functions of, at least, transferring pages of boot data subsequent to the first page from nonvolatile memory 120 to memory 114. (*Supra* Sec. VI.B.2.) The DMA controller of the asserted combinations would have used Post's next-page pointer metadata as the "descriptors" to enable these transfer operations. (INTEL-1003 ¶ 149.)

Post disclosed that the pointer metadata "reference[d] the next boot page in the sequence" by "provid[ing] the address" of the next page of boot data. (INTEL-1005, 6:38-42; *id*., 6:51-53 (a "next page pointer can be an address referencing another page in the same block" or "another page in a different block of the same or a different integrated circuit"); *id*., 6:19-21, 6:28-31, 6:22-24.) Post further taught that pointers were used to "selectively read" pages of NVM 120, "such that some or all of the non-data pages are skipped over." (*Id*., 10:48-49; INTEL-1003, ¶ 146.) Post's Figure 3 (shown below left, with boot data pages annotated in orange) illustrated the linked-list format of Post's boot data pages, with pointer descriptors 302 and 304 identifying additional pages of boot data. The '694 patent describes DMA controller descriptors in this same way, as pointers pointing to additional locations in nonvolatile memory (*id*. ¶ 147):

Post (INTEL-1005)	'694 Patent (INTEL-1001)
	

(INTEL-1005, Fig. 3; INTEL-1001, Fig. 5.)

In the asserted combinations, the DMA controller within control circuitry 112 performed the functions of transferring pages of boot data subsequent to the first identified boot page from NVM 120 to memory 114. *Supra* Sec. VI.B.2. Kao disclosed a DMA controller that automatically loaded boot codes from nonvolatile memory into SDRAM in an embedded system at power-up. (INTEL-1004, 9:10-13 ("when the PowerUp_Trigger signal is received, the DMA engine 120 may

automatically load the boot and controller codes 132, 134"); *id.*, 10:23-26, 11:16-19.) The DMA controller identified the address in nonvolatile memory where the boot codes were stored, and then "loaded [the boot codes] from boot code image 232 in the NAND flash memory 130 [*i.e.*, nonvolatile memory] to the SDRAM array 108 via a DMA transfer[.]" (*Id.*, 10:21-26; *id.*, 9:16-21, 8:61-67.) Kao further taught that the DMA engine's automatic transfer of stored boot codes occurred "without interaction with the host." (*Id.*, 3:19-21.) A POSITA would thus have understood that Kao's DMA controller, within control circuitry 112, would have used Post's pointer metadata to provide the address of the boot data to be transferred from NVM 120 to memory 114—in line with the normal operation of the DMA controller. (INTEL-1003, ¶¶ 148-49.)

Additionally, a POSITA would have found it obvious for Kao's DMA controller to use Post's next-page pointer metadata as a "DMA controller descriptor." As discussed in connection with claim 1[a] (*supra* Sec. VI.B.2), a POSITA would have been motivated to incorporate a DMA controller that performed DMA transfers of boot data from NVM 120 to memory 114 into Post. Kao supplied a teaching, suggestion, or motivation for a DMA controller to use the address of an item in NVM—*e.g.*, the information in Post's next-page pointer metadata (INTEL-1005, 6:40-50)—to facilitate a transfer. (INTEL-1004, 5:7-16, 7:1-4, 7:30-33, 10:4-7, 10:42-46.) A POSITA would thus have been motivated to

use the address information of Post's next-page pointer metadata to allow Kao's DMA controller to transfer pages of boot data from NVM 120 to memory 114. (INTEL-1003, ¶ 149.)

A POSITA would also have understood this to be the combination of known elements (Kao's DMA controller and Post's next-page pointer information) according to known methods (Kao's DMA techniques of identifying data or instructions located at a particular address in NVM), to yield predictable results (identification of pages of boot data in Post's NVM 120 and transfer of the boot data to RAM 114). (INTEL-1003, ¶ 150.) *KSR*, 550 U.S. at 416 ("The combination of familiar elements according to known methods is likely to be obvious when it does no more than yield predictable results."); *id*. at 417 ("when a patent simply arranges old elements with each performing the same function it had been known to perform and yields no more than one would expect from such an arrangement, the combination is obvious") (quotation omitted).

7. Claim 1[f]. wherein the POR sequencer reads and uses the register information to update configuration registers of the embedded system.

The control circuitry of the asserted combinations *read and used the register information to update configuration registers of the embedded system*. Wilson taught loading initial register values into a microcontroller's registers. (INTEL-1013, 2:59-63 ("the initial values [of boot data] can be stored in the register file as

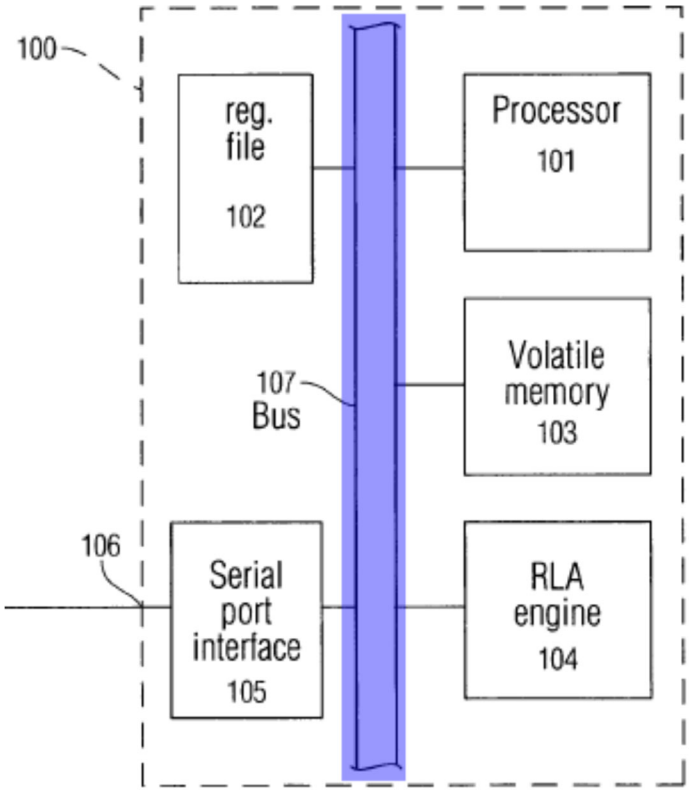
part of an initialization … operation"), 1:24-28 ("The boot-up process allows a microcontroller to perform various actions to prepare the microcontroller for ordinary operation, such as load initialization values in its registers, load instructions in its instruction memory, etc."), 2:1-3 (describing "stor[ing] various portions of [the boot data] in their final destinations (such as, for example, in respective registers)"), 6:27-29 (boot data was "use[ed] … to load the various registers in register file 102"), Figure 3; INTEL-1003, ¶¶ 152-54.) Wilson's registers included registers that were "custom designed to perform specific predefined functions," such as configuring the touch screen display and pixel color values ("*configuration registers*"). (*Id*., 3:35-62, 11:11-39; INTEL-1003 ¶ 155.) The '694 patent similarly describes a "register initialization phase" that entails loading registers with initial values. (INTEL-1001, 7:17-44, Figure 7; INTEL-1003 ¶ 156.)

For the same reasons described in connection with claim 1[b] (*supra* Sec. VI.B.3), a POSITA would have found it obvious for control circuitry 112, as modified to include Wilson's register file and circuitry to perform functionality of Wilson's RLA engine ("POR sequencer") to read and use register information to update configuration registers of Post's SoC 110 ("embedded system") as part of the boot-up process. (INTEL-1003, ¶¶ 151-57.)

C. Claim 5. The apparatus of claim 1, further comprising: a CPU bus in the embedded system.

A POSITA would have understood that Post's SoC 110 included a CPU bus, and this would have been obvious, because Post disclosed that processor 112 communicated with the other components of SoC 110. INTEL-1005, 4:3-5, 3:53-57 (processor 112 worked with NVM interface 118 to load boot data, stored boot data in memory 114). Post's disclosure that these communications happened would have suggested to a POSITA that SoC 110 included a CPU bus that interconnected processor 112 with the other components. (INTEL-1003, ¶ 158.) This would also have been obvious, because CPU buses were common in embedded systems and a known way to facilitate the interconnection needed by Post's processor 112. (*Id*.)

Additionally, Wilson disclosed a bus 107 that interconnected processor 101 with the other components within microcontroller 100. "Bus 107 is used for communications between the various modules of the microcontroller. The bus may allow each module to communicate with each other module according to known protocols." (INTEL-1013, 3:11-14; *id*., Fig. 2 (excerpted).)



A POSITA would have found it obvious to incorporate Wilson's teaching of bus 107 (a "*CPU bus*") into Post's SoC 110. (INTEL-1003, ¶¶ 159-60.) A POSITA would have viewed Wilson as providing a teaching, suggestion, or motivation to include bus 107 pursuant to its teaching that the bus allowed modules within a microcontroller to communicate with each other. (INTEL-1013, 3:11-14.) Like Wilson's microcontroller, Post's SoC 110 had a number of modules that needed to communicate with, and even controlled, each other. (INTEL-1005, Fig. 1, 3:33-65; INTEL-1003, ¶ 160.) Further, as Post is silent about the specific structure within SoC 110 that allows communication between components, a POSITA would have looked to Wilson for potential implementation details. (*Id.*)

D. Claims 6 and 9

The limitations of claims 6 and 9 recite limitations that are analogous to those of claim 1 and are obvious for the reasons discussed above. (INTEL-1003, ¶¶ 161-67.) To summarize:

Claim Limitation[1]	Basis for Obviousness
Claim 6[pre]	*See* Claim 1[pre]
Claim 6[a]	*See* Claim 1[a]
Claim 6[b]	*See* Claim 1[b] and 1[d]
Claim 6[c]	*See* Claim 1[e]
Claim 6[d]	*See* Claim 1[f]
Claim 9	*See* Claim 1[c]

E. Claim 12

1. Claim 12[pre]. An article of manufacture, comprising:

Intel does not concede that the preamble is limiting. Post described an electronic device 100 that was any of a portable media player, an iPhone or other cellular telephone, "a pocket-sized personal computer, a personal digital assistance ("PDA"), a desktop computer, a laptop computer, and any other suitable type of

[1] Appendix B to INTEL-1003 lists the challenged claims in full.

electronic device or system." (INTEL-1005, 2:61-67.) Each of these was an example of an "article of manufacture." (INTEL-1003, ¶¶ 169-70.)

2. Claim 12[a]. a non-transitory computer-readable medium having stored thereon instructions to permit an apparatus to perform a method comprising:

Post disclosed boot ROM 116 (a "*non-transitory computer-readable medium*") that stored program code ("*instructions*") that permitted control circuitry 112 ("*apparatus*") to load information from NVM 120:

> Boot ROM 116 can include any suitable read-only memory ("ROM), such as a mask ROM. Boot ROM 116 may be configured to store program code that is executed by control circuitry 112 on bootup of electronic device 100. The program code may be executed to begin loading additional information from NVM 120 into memory 114.

(INTEL-1005, 3:53-57; *id*., 3:41-43.) Loading the "additional information from NVM 120," which included boot data, permitted control circuitry 112 (as modified by the teachings of Kao and Wilson) to perform the operations of Post's boot data retrieval method, illustrated in, *e.g*., processes 400 and 500. (*Id*., 3:52-4:3, 3:41-43, 8:8-12, 8:16-10:59; INTEL-1003, ¶¶ 171-73.) These teachings render obvious claims 1[b]-1[f], as discussed (*supra* Secs. VI.B.3-7), which are substantively identical to the limitations of claims 12[b]-1[f] (the method of claim 12).

3. Claim 12[b]-12[f]

The remaining limitations of claim 12 recite limitations that are generally analogous to limitations of claim 1 and are obvious for the reasons discussed above. (INTEL-1003, ¶¶ 168, 174-78.) To summarize:

Claim Limitation	Basis for Obviousness
Claim 12[b]	*See* Claim 1[b]
Claim 12[c]	*See* Claim 1[c]
Claim 12[d]	*See* Claim 1[d]
Claim 12[e]	*See* Claim 1[e]
Claim 12[f]	*See* Claim 1[f]

VII. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(B)

A. Real Party in Interest

Pursuant to 37 C.F.R. § 42.8(b)(1), Intel Corp. certifies that it is the real party-in-interest, and further certifies that no other party exercised control or could exercise control over Intel's participation in this proceeding, the filing of this petition, or the conduct of any ensuing trial.

B. Related Matters

The '694 patent is asserted in the following litigation:

BiTMICRO LLC v. Intel Corporation, 5:23-cv-625 (NDCA).

The ’694 patent is also related to U.S. Pat. No. 9,858,084, which is being challenged in a concurrently-filed petition. *See* IPR2023-00785.

C. Lead and Back-Up Counsel

Pursuant to 37 C.F.R. §§ 42.8(b)(3), 42.8(b)(4) and 42.10(a), Petitioner appoints Christina McCullough (Reg. No. 58,720), at the address of Perkins Coie LLP, 1201 Third Avenue, Suite 4900, Seattle, Washington 98101-3099, telephone number (206) 359-6897, as lead counsel; and Lori A. Gordon (Reg. No. 50,633) at the address of Perkins Coie LLP, 700 Thirteenth Street, N.W., Suite 800, Washington, DC 2005-3960, telephone number (202) 654-6200, Christopher Kelley (Reg. No. 42,714) at the address of 3150 Porter Drive, Palo Alto, CA 94304-1212, telephone number (650) 838-4300, and Chad S. Campbell (pro hac vice to be requested), at the address of Perkins Coie LLP, 2901 N. Central Avenue, Suite 2000, Phoenix, AZ 85012, as back-up counsel.

Petitioner consents to electronic service by email at the email addresses: Intel-BitMicro-084IPRService@perkinscoie.com; mccullough-ptab@perkinscoie.com; gordon-ptab@perkinscoie.com; kelley-ptab@perkinscoie.com; and campbell-ptab@perkinscoie.com.

VIII. CONCLUSION

For the grounds specified above, *Inter Partes* Review of the challenged claims is respectfully requested.

Respectfully submitted,

/s/ Christina J. McCullough

Christina J. McCullough
Registration No. 58,720

Perkins Coie LLP
1201 Third Avenue, Suite 4900

Seattle, WA 98101-3099

Dated: March 31, 2023

CERTIFICATE OF COMPLIANCE

The undersigned certifies that this brief complies with the type-volume limitations of 37 CFR §42.24. This brief contains 13,651 words, excluding the cover page and signature block, and the parts exempted by 37 C.F.R. § 42.24(a), as calculated by the "Word Count" feature of Microsoft Word for Office 365, the word processing program used to create it.

Respectfully submitted,

Perkins Coie LLP

/s/ Christina J. McCullough

Christina J. McCullough
Registration No. 58,720

Attorney for Petitioner

Dated: March 31, 2023

Perkins Coie LLP
1201 Third Avenue, Suite 4900
Seattle, WA 98101-3099

CERTIFICATE OF SERVICE

The undersigned hereby certifies that true copies of the foregoing Petition for *Inter Partes* Review of U.S. Patent 10,120,694 and supporting materials (Exhibits 1001-1021 and Power of Attorney) have been served via overnight mail on March 31, 2023, in their entirety on the following:

Jackson Walker LLP
100 CONGRESS AV, STE 1100
AUSTIN, TX 78701

Respectfully submitted,

Perkins Coie LLP

/s/ Christina J. McCullough

Christina J. McCullough
Registration No. 58,720

Dated: March 31, 2023

Attorney for Petitioner

Perkins Coie LLP
1201 Third Avenue, Suite 4900
Seattle, WA 98101-3099