

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,
Petitioner

v.

BITMICRO LLC,
Patent Owner

Case No.: IPR2023-00785

U.S. Patent No. 9,858,084

**PETITION FOR *INTER PARTES* REVIEW
OF U.S. PATENT NO. 9,858,084**

TABLE OF CONTENTS

	Page
I. INTRODUCTION	1
II. GROUNDS FOR STANDING.....	2
III. IDENTIFICATION OF CHALLENGE	2
A. Identification of Prior Art.....	2
B. § 325(d): The Art and Arguments Were Not Previously Presented	3
C. Ground for Challenge.....	5
IV. TECHNICAL BACKGROUND	6
A. Power-On Reset and Boot	6
B. System Registers	8
C. Direct Memory Access (“DMA”)	9
V. CHALLENGED U.S. PATENT NO. 9,858,084.....	12
A. Overview of the ’084 Patent.....	12
B. Prosecution History	15
C. Claim Construction	17
D. Level of Ordinary Skill in the Art.....	17
VI. BASIS OF THE CHALLENGE TO PATENTABILITY	17
A. Overview of Prior Art	18
1. Post Disclosed a Process for Copying Boot Data from Nonvolatile Memory to RAM and Verifying Integrity of the Boot Data	18
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. Overview of Combination.....	25
B. Claim 1: The combinations of Post and Kao, and Post and Kao in view of Wilson, rendered claim 1 obvious	26
1. Claim 1[pre]. A computer-implemented method, comprising:.....	26
2. Claim 1[a]. releasing components of an embedded system from reset;	26
3. Claim 1[b]. detecting a power-on reset (POR) sequencer descriptor in a nonvolatile memory; wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system, Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments, and a system firmware;	30
4. Claim 1[c]. copying the POR sequencer descriptor from the nonvolatile memory to a random access memory (RAM);	48
5. Claim 1[d]. verifying an integrity of the POR sequencer descriptor; and.....	50
6. Claim 1[e]. detecting any error in the POR sequencer descriptor;.....	51
7. Claim 1[f]. wherein the computer-implemented method allows the embedded system to verify an integrity of user data and program code used in a boot process in order to provide a resilient boot up sequence and wherein the computer-implemented method reduces an amount of the nonvolatile memory used in the boot process.....	51
C. Claim 5. The method of claim 1, further comprising activating a POR sequencer error handler if an error is detected in the POR sequencer descriptor.	56

TABLE OF CONTENTS
(continued)

	Page
D. Claim 8. The method of claim 1, further comprising: performing a register initialization phase in the embedded system.	57
E. Claim 9. The method of claim 1, further comprising: copying system firmware from a storage device to the RAM.	57
F. Claims 19 and 20.....	59
1. Claim 19[pre]. An article of manufacture, comprising:	59
2. Claim 19[a]. a non-transitory computer-readable medium having stored thereon instructions operable to permit an apparatus to perform a method comprising:	59
3. Remaining limitations of claims 19 and 20	60
VII. MANDATORY NOTICES UNDER 37 C.F.R. § 42.8(B)	61
A. Real Party in Interest	61
B. Related Matters.....	61
C. Lead and Back-Up Counsel.....	62
VIII. CONCLUSION.....	62

EXHIBIT LIST

Exhibit	Description
INTEL-1001	U.S. Patent 9,858,084 to Manlapat (“the ’084 patent”)
INTEL-1002	File History of U.S. Patent 9,858,084
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	<u>Microsoft Computer Dictionary</u> (5th Ed.)
INTEL-1008	J. Catsoulis, <u>Designing Embedded Hardware</u> (2005)
INTEL-1009	J. Ganssle et al., <u>Embedded Hardware</u> (2008) (“Ganssle”)
INTEL-1010	Q. Li, <u>Real-Time Concepts for Embedded Systems</u> (2003)
INTEL-1011	Federal Judiciary Statistics prepared by Administrative Office of the U.S. Courts
INTEL-1012	P. Barry et al., <u>Modern Embedded Computing</u> (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., <u>Hardware Verification with SystemVerilog</u> (2007)
INTEL-1016	T. Noergaard, <u>Embedded Systems Architecture</u> (2005)
INTEL-1017	AHB Example AMBA System Technical Reference Manual, ARM, 1999
INTEL-1018	J. Lenk, <u>Simplified Design of Microprocessor Supervisory Circuits</u> (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, 8-9 and 19-20 of U.S. Patent No. 9,858,084 (INTEL-1001). The ’084 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, 2:48-54, claims 1-20.)

The ’084 patent specification and challenged claims identify specific contents of the POR sequencer descriptor: “The POR Sequencer descriptor contains register information and direct memory access (DMA) controller descriptors.” (*Id.*, 2:52-54.) Claim 1 recites that the POR sequencer descriptor also comprises “a system firmware.” (*Id.*, 9:22.) At system power-on, the “POR sequencer descriptor” is copied from non-volatile memory to RAM. (*Id.*, 3:14-25.) The embedded system uses register information in the POR sequencer descriptor to program configuration registers (*id.*, 3:28-34) and then processes the DMA descriptor to load additional information or firmware (*id.*, 3:34-45).

The phrase “POR sequencer descriptor”—which was added to the claims during prosecution to overcome prior art—was coined by the applicants. But there is nothing in the ’084 patent’s “POR sequencer descriptor” elements, or in the patent’s disclosed boot processes, that was not already known in the art. Instead, the claims recite conventional aspects of embedded system boot processes and Direct

Memory Access techniques that were known years before the '084 patent's priority date. This Petition, supported by the declaration of Dr. Donald Alpert, demonstrates that the challenged claims are obvious in view of the cited prior art and are unpatentable.

II. GROUNDS FOR STANDING

Petitioner certifies the '084 patent is available for IPR and it is not barred or estopped from requesting IPR on the grounds in this Petition. (INTEL-1006.)

III. IDENTIFICATION OF CHALLENGE

A. Identification of Prior Art

The '084 patent issued from application 14/217,399 filed on March 17, 2014. That application claimed priority from provisional application 61/801,952 filed on March 15, 2013. Petitioner does not concede that the '084 patent is entitled to benefit of the provisional application. Nevertheless, each applied reference was issued, published, or filed before the earliest alleged priority date.

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 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 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 pre-AIA 35 U.S.C §§ 102(a), (b) and (e).

The Examiner indicated that he considered Kao during examination of the '084 patent. (*See* INTEL-1002, Notice of Allowance of 20 July 2016, IDS sheet 14 of 37.) The Examiner, however, does not appear to have considered Kao in combination with other references, and Kao was not substantively discussed in any office action or response. The Post and Wilson references were not presented and could not, therefore, 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, Paper No. 6, 7 (PTAB Feb. 13, 2020) (precedential). 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.*, 8.

The applicants identified hundreds of patents and publications during prosecution of the application that led to the '084 patent. While Kao was one of these references, neither the Examiner nor the applicants referred to Kao during prosecution. Post and Wilson were not presented at all. Thus, the Office has not previously decided patentability of the challenged claims over the combination presented in this Petition.

Further, the prior art presented during prosecution is not substantially the same as that presented in this Petition. The Examiner initially rejected the applicants' pending claims over U.S. Patent No. 7,921,237 (INTEL-1014, "Holland"). (*See* INTEL-1002, Office Action of 11 Jan. 2016.) In response, the applicants amended the independent claims to recite specific components of a "POR sequencer descriptor." (*Id.*, [Amendment of 10 April 2016, at 3, 7].) The applicants argued that Holland did not disclose these newly-recited "POR sequencer descriptor" elements. (*Id.*, [at 9].) The Examiner then allowed the claims as amended.¹ (INTEL-1002, [Notice of Allowance of 20 July 2016].) For the reasons

¹ Prosecution continued after the July 20, 2016, Notice of Allowance, as applicant filed two successive Requests for Continued Examination (RCE), but the Examiner did not enter any further rejections based on 35 U.S.C. §§ 102 or 103.

presented in this Petition, the combinations of Kao, Wilson, and Post render obvious the “POR sequencer descriptor” elements that the applicants added.

The first part of the 35 U.S.C. § 325(d) framework therefore fails because neither the same or substantially the same art nor the same or substantially the same arguments were previously presented to the Office. Accordingly, the second part of the framework also fails because neither condition of the first part is satisfied. *See, e.g., Caterpillar Inc. v. Wirtgen Am. Inc.*, IPR2022-01278, Paper 12, 43 (PTAB Feb. 7, 2023). *See also Innovations4Flooring NV v. Välinge Innovation AB*, IPR2017-02130, Paper No. 13, 7-8 (P.T.A.B. March 26, 2018) (declining to exercise discretion where Examiner initialed but did not make reference to applicant-disclosed art). This ends the §325(d) inquiry.

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 '084 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. Ground for Challenge

Ground	35 U.S.C. §	Claims	Combination
1	103	1, 5, 8-9, 19, 20	Post, Wilson, and Kao

IV. TECHNICAL BACKGROUND

The '084 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, 1:16-20.) At the time of the '084 patent’s earliest alleged 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.”).)

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”).)

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.”).)

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.”.)

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.”).)

C. Direct Memory Access (“DMA”)

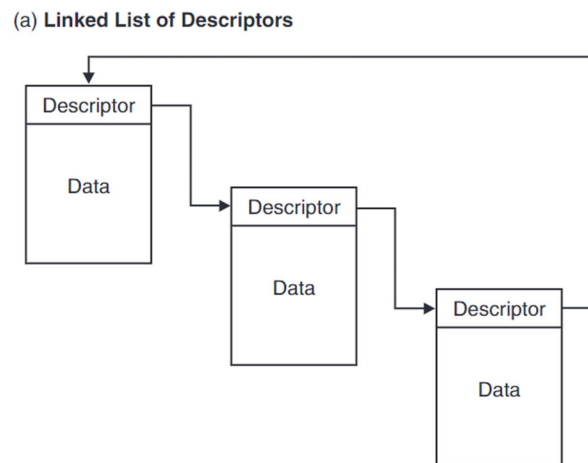
The core processor itself may be responsible for performing “data transfer between internal and external memory spaces.” (INTEL-1009, 214; INTEL-1008, 20.) These memory transfer operations require significant processing workload and

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”).)

Direct Memory Access (“DMA”) techniques were one way to offload this work from the core system processor. 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.”).) 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.”.)

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.)

V. CHALLENGED U.S. PATENT NO. 9,858,084

A. Overview of the ’084 Patent

The ’084 patent describes a system “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, 1:17-20.) 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.*, 1:28-33.) The patent notes that reducing the size of this firmware would reduce the size of the required nonvolatile memory. (*Id.*, 2:12-16.) It also recognizes, however, that the boot process should be able to verify the integrity of data and initialize configuration registers of the embedded system. (*Id.*, 2:26-21.)

The patent aims to address these issues through the use of a “Power-On Reset (“POR”) sequencer descriptor” that can be distributed across multiple storage devices. Figure 2 illustrates components of the ’084 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), POR sequencer (in blue) and various POR sequencer descriptor fragments (in orange):

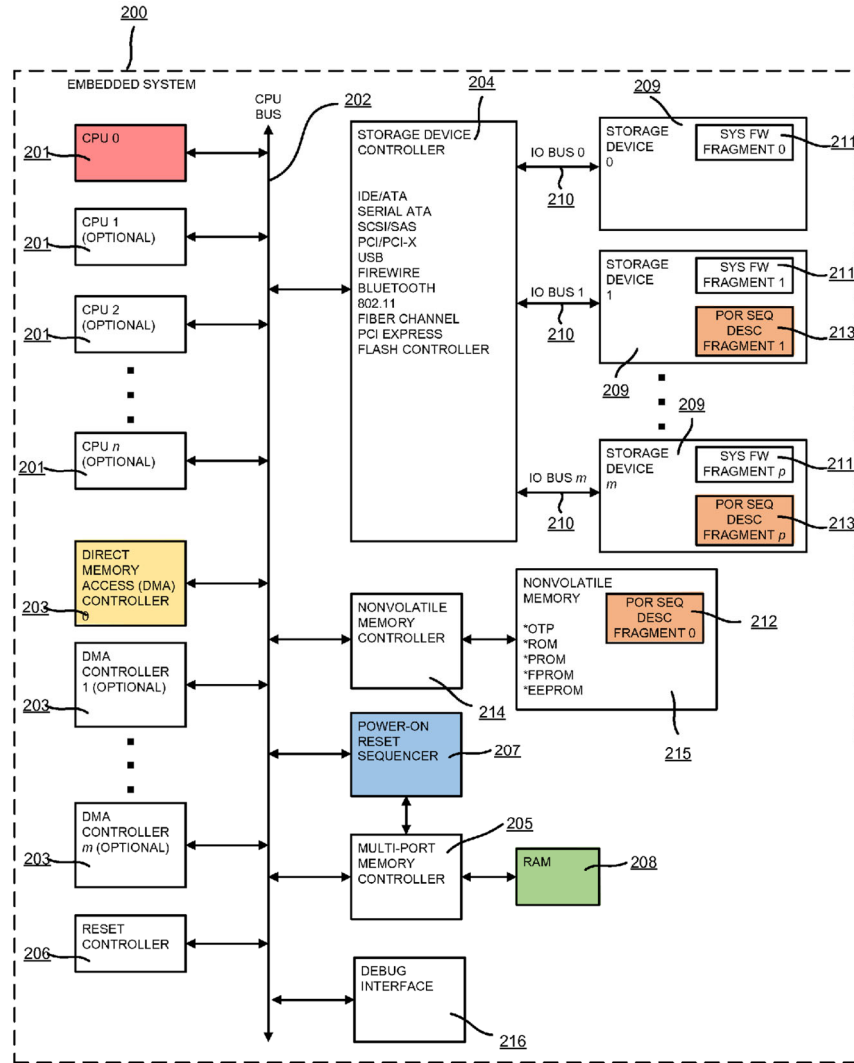
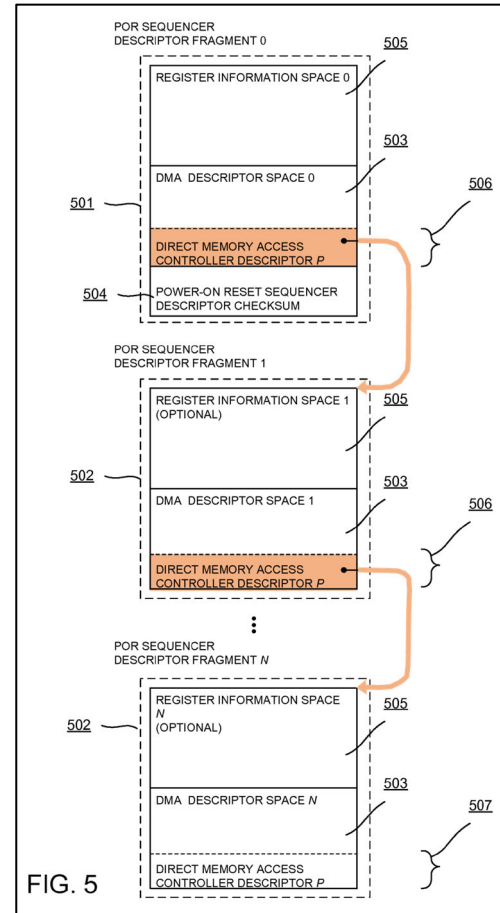


FIG. 2

(*Id.*, Fig. 2, 2:31-41; 4:49-60; Fig. 2.)

The '084 patent explains that the “POR Sequencer Descriptor” is “a preassembled descriptor that is stored in a nonvolatile memory of the system.” (*Id.*, 2:49-52.) It 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.*, 2:52-58.) 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).) These fragments can be stored as a “linked-list” format, where each DMA descriptor “points to the next POR sequencer



descriptor fragment.” (*Id.*, 4:30-31, 2:61-65, 6:44-46.) Figure 5, shown at right, illustrates this “linked-list” POR sequencer descriptor format. (*Id.*, Fig. 5, 6:42-55.)

The “Power-On Reset Sequencer” component “controls the overall boot process of the embedded system.” (*Id.*, 2:37-38.) At system power-on, the POR sequencer and various other system components are brought out of reset. (*Id.*, 6:58-63.) The POR sequencer “detects the presence of POR sequencer descriptor 212 [i.e., fragment 0] in the nonvolatile memory [215].” (*Id.*, 6:64-65; *id.*, Fig. 2.) It then copies the first fragment from nonvolatile memory 215 to RAM and checks the

integrity of that fragment. (*Id.*, 6:66-7:5.) “If an error is found, a POR sequencer error handler is activated[.]” (*Id.*, 7:7-9.)

The POR sequencer then processes the POR sequencer descriptor fragment that was loaded into RAM. If the fragment contains register information, the POR sequencer “processes the register information and then updates the corresponding configuration register[s] based on the contents of the register information.” (*Id.*, 7:23-29.) The POR sequencer then writes the DMA controller descriptor to the DMA controller’s register space, and the DMA controller transfers the next fragment of the POR sequencer descriptor to RAM. (*Id.*, 7:29-37.) If the fragment contains a new set of register information, the POR sequencer processes and loads that additional information into system registers. (*Id.*, 7:39-45.) The DMA controller descriptor may also point to system firmware, in which case the DMA controller copies the firmware into RAM, and the POR sequencer verifies its integrity. (*Id.*, 7:47-8:1.)

B. Prosecution History

During prosecution, the Examiner rejected the preliminarily amended claims (1-20) under pre-AIA 35 U.S.C. § 102 over Holland (INTEL-1013). (*See* INTEL-1002, Non-Final Rejection sent Jan. 11, 2016.) In response, applicant amended independent claims 1, 10 and 19 to insert the following additional language specifying components of the “POR sequencer descriptor”:

wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system, Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments, and a system firmware.

(*Id.*, Amendment of 10 April 2016.) The applicants argued that Holland did not disclose these new elements of the “POR sequencer descriptor.” (*Id.*, 9.)

In an office action sent on November 1, 2016, the Examiner rejected claims 1 through 18 for patentability under 35 U.S.C. § 101. (*Id.*, Office Action sent Nov. 1, 2016 at 3-4.) Claims 19 and 20 were allowed. The applicant responded by adding the following limitation to claim 1:

wherein the computer-implemented method allows the embedded system to verify an integrity of user data and program code used in a boot process in order to provide a resilient boot up sequence and wherein the computer-implemented method reduces an amount of the nonvolatile memory used in the boot process.

(*Id.*, Amendment dated Feb. 1, 2017.) Other modifications were made to claim 10, which is not challenged here.

The Examiner issued a notice of allowance on May 12, 2017. The applicant then filed a second RCE with minor amendments to claim 19 on August 12, 2017. The Examiner issued a new notice of allowability on August 21, 2017.

As Petitioner demonstrates below, none of the limitations recited by the challenged claims was new when the '084 patent was filed. In particular, each of the added “POR sequencer descriptor” components—information to initialize

configuration registers, DMA descriptors, and firmware—was already known. Computer-implemented methods to verify an integrity of data and program code used in a boot process and to reduce an amount of nonvolatile memory were also known.

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). Intel does not believe that any term in the challenged claims requires an express construction. *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 CHALLENGE TO PATENTABILITY

As described below, the challenged claims are obvious in view of Post combined with teachings from Wilson and Kao.

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 a chip (SoC) 110 and non-volatile memory (“NVM”) 120:

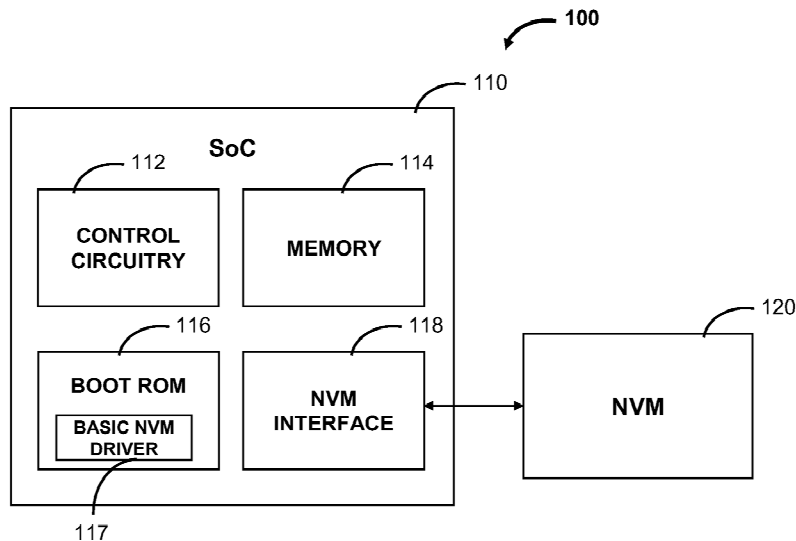
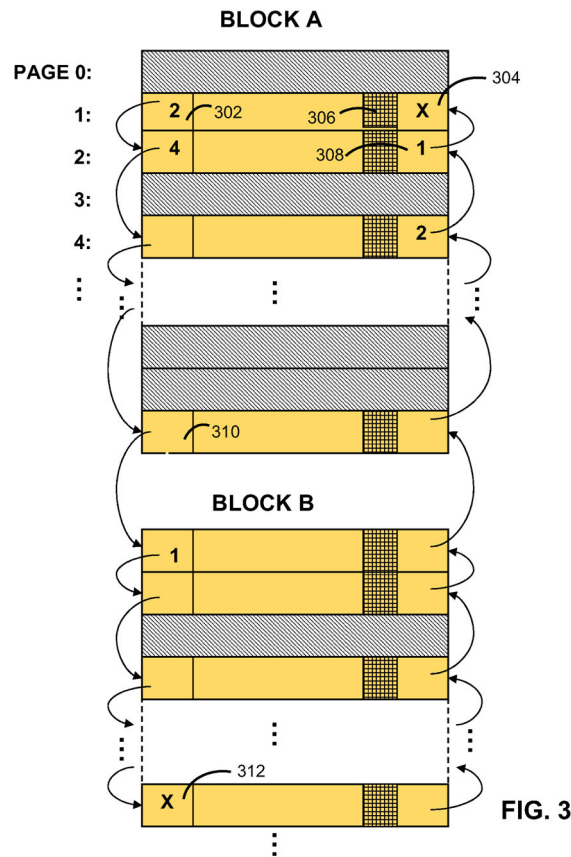


FIG. 1

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 100 further included memory 114 that was “any suitable type of volatile memory,” such as RAM. (*Id.*, 3:33-34, 3:44-49.)

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.*, 2:39-42.)

The stored boot data pages had “both data (e.g., boot data) and metadata.” (*Id.*, 6:28-29.) 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:



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.) 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.) 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). 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.)

Post’s control circuitry incorporated integrity checking and error detection 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. (INTEL-1005, 8:34-38, 6:60:65.) Through these techniques, Post’s process provided a resilient boot up sequence that could determine if critical boot data was

missing and abort the boot sequence as necessary. (INTEL-1005, 6:64-65; *id.* 8:62-66.)

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.)

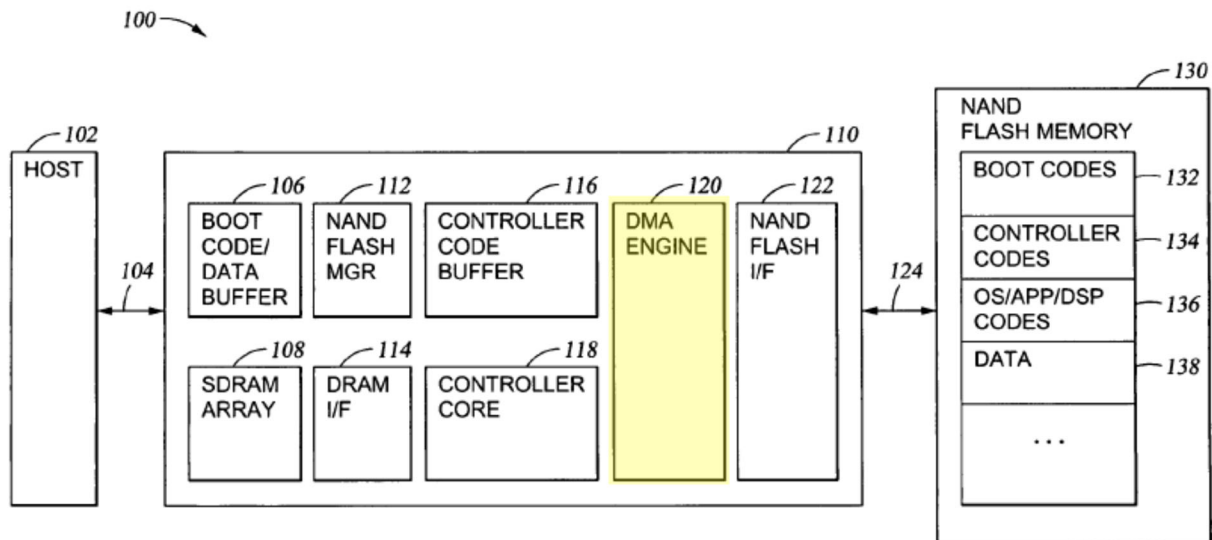


Fig. 1

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.)

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.)

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.)

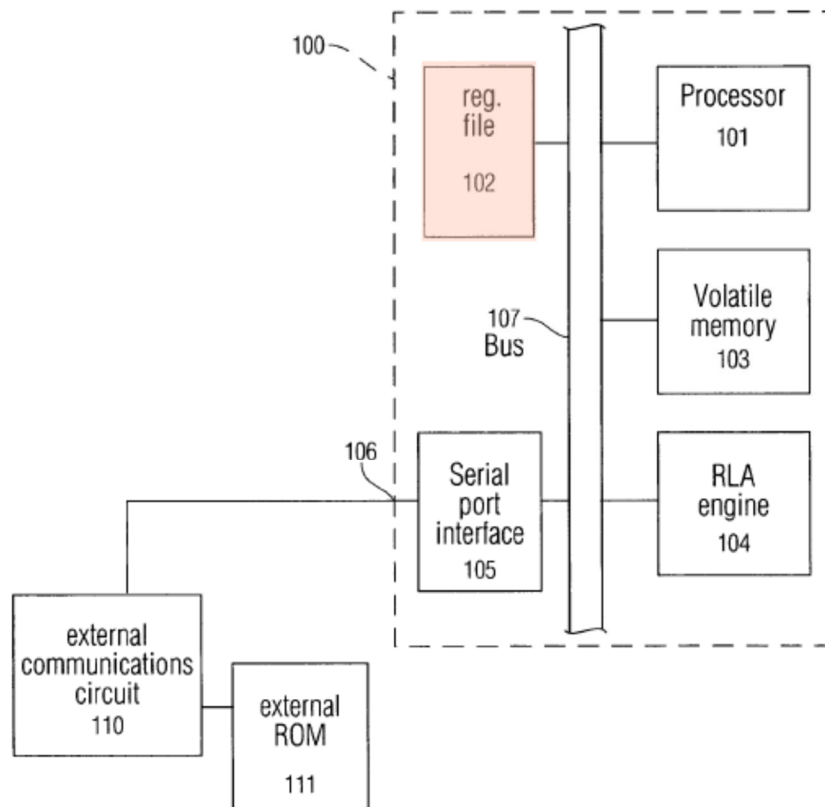


Fig. 1

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.)

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.)

The microcontroller received the data block and “stor[ed] data field 402 of the data block in volatile memory.” (*Id.*, 5:56-61.) A generally programmable processor implemented as a Register Load Assist (“RLA”) engine decompressed the boot data “into various register initialization values.” (*Id.*, 6:3-6, 6:13-15.) 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.)

4. Overview of Combination

As and for the reasons described in detail below, the combination of Post, Kao, and Wilson disclosed all limitations of the challenged claims. In the resulting combination, Post’s control circuitry 112 would be modified to include circuitry that performed the register loading functionality of Wilson’s RLA engine as well as circuitry to perform the boot data transfer functionality of Kao’s DMA engine; and Post’s boot data included initial register values, as taught by Wilson.

B. Claim 1: The combinations of Post and Kao, and Post and Kao in view of Wilson, rendered claim 1 obvious

1. Claim 1[pre]. A computer-implemented method, comprising:

Intel does not concede that the preamble is limiting. To the extent it is, Post disclosed *a computer-implemented method*. Post disclosed an “electronic device 100” 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.) Figures 4-6 of Post showed “flowcharts of illustrative processes” that were “performed by ... control circuitry 112 ... under the control of mask ROM code (e.g., from boot ROM 116) or from an operating system and NVM driver.” (*Id.*, 8:7-15.) The techniques discussed below were therefore performed by components of Post’s electronic device 100, and Post taught implementing this device as a computer. (INTEL-1003, ¶¶ 68-70.)

2. Claim 1[a]. releasing components of an embedded system from reset;

Post’s system-on-chip (“SOC”) 110 and NVM 120 were collectively *an embedded system*. (INTEL-1003, ¶ 71.) The ’084 patent describes an “embedded

system” as including a system with one or more processors. (INTEL-1001, 1:28-30 (“embedded system will be used to refer to a single CPU or a multiprocessor system in this paper”).) Post’s electronic device 100 “include[d] system-on-a-chip (‘SoC’) 110 and non-volatile memory (‘NVM’) 120.” (INTEL-1005, 3:1-2.)

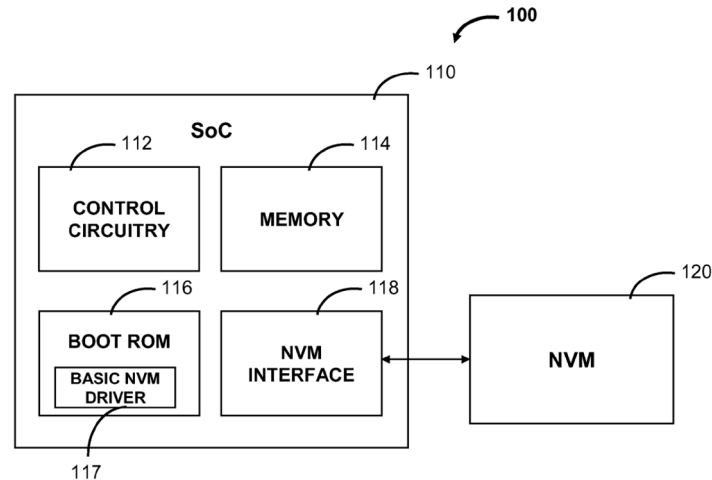


FIG. 1

(*Id.*, Fig. 1.) System-on-chip 110 included various components, such as “control circuitry 112, memory 114, boot ROM [read-only memory] 116, and NVM interface 118,” and control circuitry 112 “include[d] any suitable components, circuitry, or logic, such as one or more processors.” (*Id.*, 3:33-34, 3:36-38, 3:52-53.) Because Post’s SoC 110 and NVM 120 had at least one processor (control circuitry 112, implemented as “one or more processors”), a POSITA would have understood SOC 110 and NVM 120 to be an example of an “embedded system” as the ’084 patent used that term. (INTEL-1003, ¶¶ 71-72.) A POSITA would also have understood that a cellular phone, such as Post’s device 100 that included SoC 110 and NVM

120 (INTEL-1005, 2:61-64, Fig. 1), was another example of an “embedded system.” (INTEL-1003, ¶ 73; INTEL-1016, 6 (cell phone one example of an “embedded system”).) Post referred to control circuitry 112 interchangeably as “processor 112” (*id.*, 4:5-6), and this Petition will do the same.

Post further disclosed *releasing components (control circuitry 112 and NVM interface 118) of an embedded system from reset*. When a system is powered on, components of the system are initially held in a reset state before they can be booted up. (INTEL-1004, 6:65-67 (“When the system 100 is initially powered up, each of the system components may enter a reset state 402.”); INTEL-1018, 2 (“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 ... reset[s] all internal circuits at a predictable starting point.”); INTEL-1003, ¶ 78.) For example, the ’084 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, 8:25-40, 3:62-67.)

Post’s control circuitry 112 and NVM interface 118 were components of SoC 110 (“*components of [the] embedded system*”). (INTEL-1001, 3:33-34.) Post disclosed that after the device including SoC 110 was powered on, control circuitry 112 was booted by executing program code from boot ROM: “at bootup of electronic device 100, control circuitry 112 may operate under the control of program code

stored in boot ROM 116.” (*Id.*, 3:41-43.) Post also disclosed that NVM interface 118 helped control circuitry 112 load additional boot data from NVM 120. (*Id.*, 3:57-4:5 (under control of the program code, control circuitry 112 “beg[a]n loading additional information from NVM 120 into memory 114,” including, “with the aid of NVM interface 118 ... boot data from NVM 120”).) A POSITA would have understood Post’s disclosure of control circuitry 112 and NVM interface 118 performing the function of copying data from NVM 120 after “bootup of electronic device 100” to indicate that these components had been released from reset. (INTEL-1018, 2; INTEL-1003, ¶¶ 79-80.) *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 primary reference disclosed).)

Further, Post taught that process 400 for “initiating bootup” (INTEL-1005, 8:15-16) began with “a startup or reboot command”:

Process 400 may begin at step 402 in response to, for example, a startup or reboot command of the electronic device. At step 404, the electronic device may begin executing mask ROM code ... At step 406, the boot ROM data may instruct the electronic device to read an initial page from the non-volatile memory.

(*Id.*, 8:25-31.) A POSITA would have understood this to further indicate that the control circuitry and NVM interface—the components that performed the process of

transferring boot data from nonvolatile memory (*id.*, 3:52-4:5, 8:9-12)—had been released from reset before performing the above operations. (INTEL-1003, ¶¶ 81.)

3. **Claim 1[b]. detecting a power-on reset (POR) sequencer descriptor in a nonvolatile memory; wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system, Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments, and a system firmware;**

The combination of Post with teachings from Wilson and Kao rendered limitation 1[b] obvious. Section VI.B.3.i addresses the POR sequencer descriptor element of claim 1[b], and section VI.B.3.ii addresses the element of detecting the POR sequencer descriptor in a nonvolatile memory.

- i. **A POR [power-on reset] sequencer descriptor [that] comprises information to initialize configuration registers of the embedded system, Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments, and a system firmware;**

Post described boot data and associated pointer metadata stored in NVM 120. This boot data and associated metadata (as modified in view of Wilson’s teachings to include initial register values) facilitated a boot sequence after device 100 was powered on and was thus a “power-on reset sequencer descriptor.” (INTEL-1003, ¶¶ 83-84.)

“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-1005, 1:16-19.) Post described this boot sequence as facilitated by boot data and associated

pointer metadata stored in NVM 120. At system bootup, control circuitry 112 “beg[an] loading additional information from NVM 120 into memory 114” that included “firmware, such as a first stage bootloader.” (*Id.*, 3:55-59.) Post referred to this “firmware [or] other data” as “boot data.” (*Id.*, 3:63-65.) 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 identify additional boot data pages to retrieve. (*Id.*, 6:19-53, 9:47-62; *id.*, 6:22-24, 8:9-12.) Control circuitry 112 used the boot data to “load a second stage bootloader or the kernel and operating system of electronic device 100.” (*Id.*, 3:59-63.) Once the operating system was loaded, “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.)

Post’s stored boot data and associated metadata were thus a “*POR [power-on reset] sequencer descriptor*” that defined Post’s boot up sequence and enabled the retrieval of other information in the same way that the ’084 patent’s POR sequencer descriptor was described as being processed and used to control the boot-up sequence. (INTEL-1003, ¶¶ 83-84.) As discussed below, a POSITA would have found it obvious to combine Post’s boot data and associated metadata with the

identified teachings of Wilson and Kao, and this combination disclosed the remaining elements of claim 1[b].

Wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system. Wilson taught that boot data stored in nonvolatile memory included *information to initialize configuration registers of the embedded system*. Wilson disclosed a microcontroller 100 with a processor 101 and a register file 102 that included registers used by the processor. “Microcontroller 100 can include a programmable processor 101. Register file 102 can include various registers which are used by the processor 101 in the ordinary operation of the microcontroller.” (INTEL-1013, 2:51-56; *id.*, Fig. 1.)

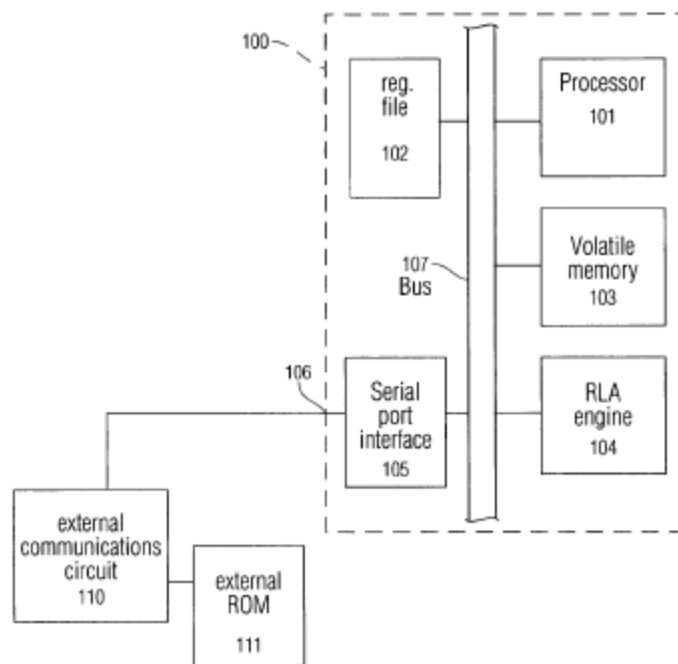


Fig. 1

Wilson disclosed an embodiment in which the “microcontroller [wa]s used to control ... a mobile phone featuring a multifunction touch screen.” (*Id.*, 3:35-37.) The touch screen microcontroller was “custom designed to perform specific predefined functions” and therefore included an “unorthodox register file” with registers responsible for configuring the touch screen display—for example, registers that defined the color values for screen pixels. (*Id.*, 3:37-62.)

Wilson taught that many microcontrollers required a boot-up process that allowed the microcontroller to “perform various actions to prepare [it] for ordinary operation, such as load initialization values in its registers.” (*Id.*, 1:24-28.) The microcontroller’s boot data, stored in external nonvolatile memory ROM 111, thus included “predefined initial values” to be loaded into the microcontroller’s registers during the boot process: “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.” (*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 (boot data obtained from external nonvolatile ROM 111), 3:26-34.)

A POSITA would have understood that these “initial values” stored in registers during a boot process were “information to initialize configuration registers of the embedded system”—for example, information to initialize the registers that configured display colors of the mobile phone touchscreen (“*configuration registers of the embedded system*”). (INTEL-1003, ¶ 89.) These initial register values were included in boot data stored in external nonvolatile ROM and copied to volatile memory on the microcontroller. (INTEL-1013, 5:57-61 (microprocessor receives block of boot data 400 and stores boot data portion in volatile memory 103), 5:46-51 (“The boot data can be sent in a single block. ...”), 6:42-44 (boot data includes initial register values, optionally with other data/code), 1:61-67 (boot data sent from external nonvolatile memory to internal microcontroller volatile memory).).

A POSITA would have found it obvious to modify Post in view of Wilson’s teachings, such that Post’s boot data in NVM 120 included information to initialize configuration registers of control circuitry 112 (including its processors). 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.) Wilson taught that the boot-up process for this type of microcontroller was understood to involve “load[ing] initialization values in [the microcontroller’s] registers” and that this was required 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 for a register initialization technique. (INTEL-1003, ¶¶ 92-93.)

Additionally, both Post and Wilson described use of a SoC or microcontroller to control a mobile phone. (INTEL-1005, 2:60-67, 3:33-36; INTEL-1013, 1:11-17, 3:35-37.) Wilson taught the benefit of “custom design[ing]” the 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 benefits of performing specific predefined functions like configuring screen color and touchscreen functionality, which required initialization of configuration registers, as taught by Wilson. (INTEL-1003, ¶ 94.)

Further, a POSITA would have viewed Wilson as providing a teaching, suggestion, or motivation to include initial register values in Post’s boot data because Wilson taught that register values *must* be loaded before a microcontroller can begin its 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”).) Wilson also taught that including initial register

information along with other boot data, such as initial memory values and instructions, could improve the speed of the 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, ¶ 95.) 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-1009, 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, ¶ 95.) For example, Wilson’s teachings indicate that a POSITA would have known how to custom design a microcontroller similar to Post’s SoC that included configuration registers loaded with initial values included in the boot data, as required prior to its normal operation. (INTEL-1003, ¶ 94.)

Further, a POSITA would have found this combination obvious because Wilson’s teaching of including initial register values in boot data 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; INTEL-1016, 555); and including register initialization information in Post’s boot data would have been within the skill of an ordinary artisan, who would be generally familiar with

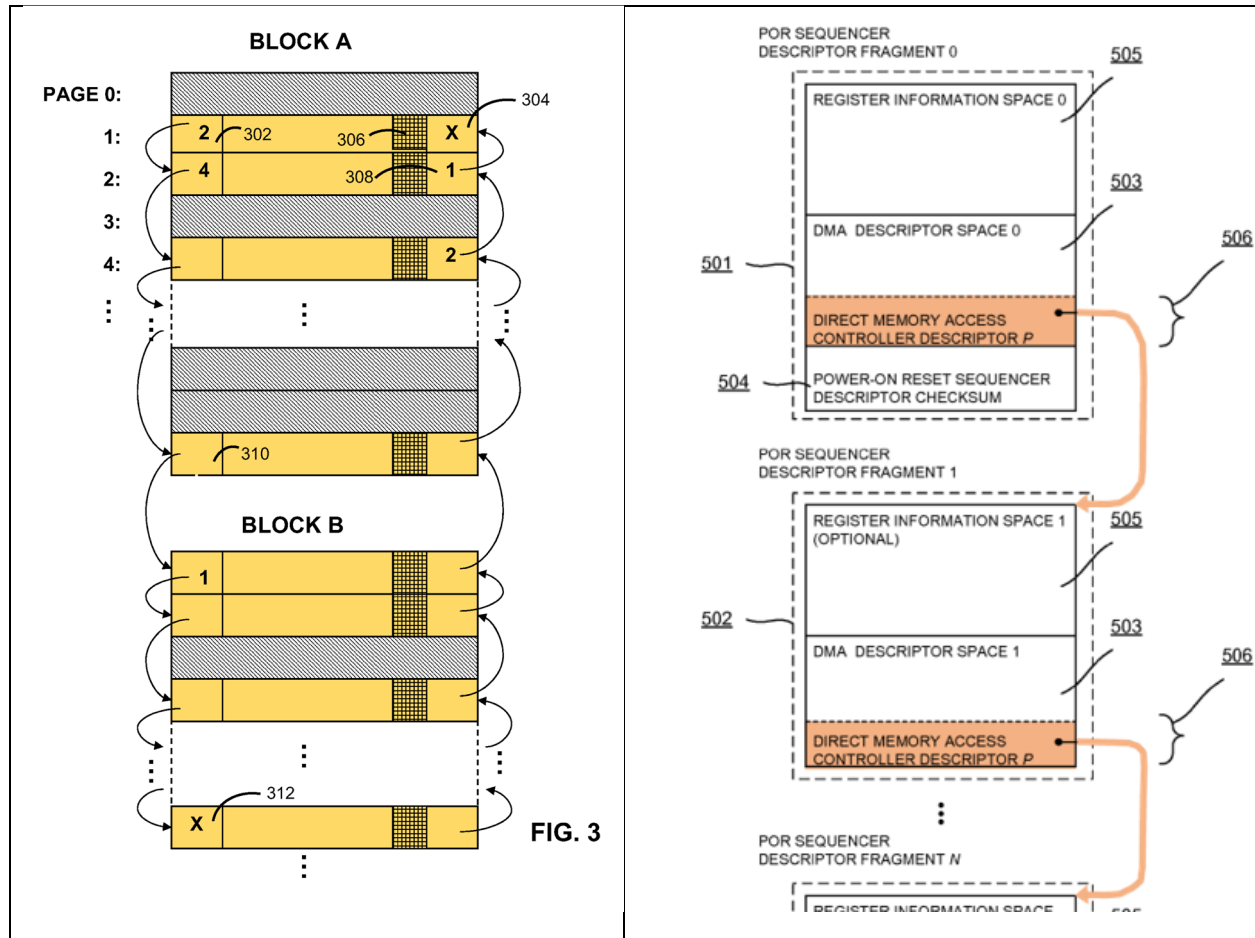
processor registers and boot techniques (INTEL-1009, 137; INTEL-1013, 1:25-28; INTEL-1005, 1:14-26). Wilson showed that a POSITA knew how to do this for a microcontroller that was similar to Post’s SoC. (INTEL-1003, ¶ 96.) The Federal Circuit has recognized that “nothing more is required to show a motivation to combine.” *Intel Corp. v. PACT XXP Schweiz AG*, -- F.4th --, 2023 WL 2469631, *5 (Fed. Cir. Mar. 13, 2023) (“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”).

A POSITA would thus have found it obvious to combine Post with Wilson’s teaching of including initial values for configuration registers in boot data. In the resulting combination, Post’s boot data stored in NVM 120 included information to initialize configuration registers of SoC 110 (*e.g.*, configuration registers of control circuitry 112).

Wherein the POR sequencer descriptor comprises ... Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments. Post disclosed storing boot data in fragments stored in different pages of nonvolatile

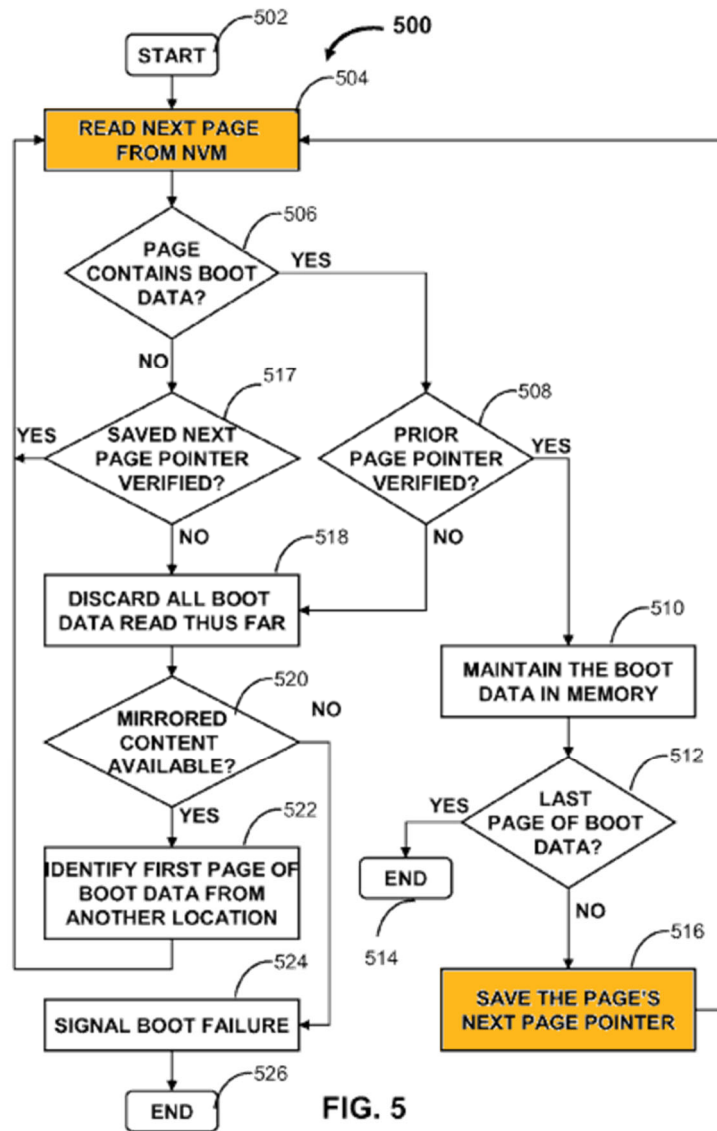
memory. “[B]oot data may be stored in a subset of the memory locations in the boot data storage area” and “the subset of pages [of NVM] selected to store boot data may be nonconsecutive.” (INTEL-1005, 4:46-47, 5:39-40; *id.*, 5:19-23; *id.*, 1:55-64.) The boot data was stored in a “linked-list format,” where “each of the boot data pages can store both data ... and metadata ... includ[ing] a next page pointer 302 and a prior page pointer 304.” (*Id.*, 6:19-21, 28-31; *id.*, 6:22-24.) 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.) Boot data pages could be in the same block or distributed across different blocks of different storage devices. (*Id.*, 6:48-53 (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”).)

Post’s Figure 3 (shown below, with boot data pages annotated in orange) illustrated the linked-list format of the boot data pages. Post and the ’084 patent (Fig. 5 excerpted below) describe similar linked-list formats, in which descriptors point to a next stored fragment:



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

Post disclosed that pointers were used to fetch each fragment of boot data stored in NVM 120 into memory 114. Figure 5 showed the process “for loading boot data into memory.” (INTEL-1005, 9:14-15; *id.*, 8:9-12, 9:7-11.) This began with “the electronic device [reading] a page from the nonvolatile memory” into memory 114 and “determin[ing] whether the current page contains boot data.” (*Id.*, 9:19-26; *id.*, Fig. 5 (step 504), 9:14-15, 9:3-6.)



If the current page contained boot data, Post’s device checked the current page’s prior-page pointer to ensure that it correctly pointed back to the most recent boot data page. (*Id.*, 9:33-38.) If that pointer information was correct, the device maintained the boot data from the current page. (*Id.*, 9:39-46.)

Post’s device then used next page pointers to retrieve additional boot data pages. It “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 indicate[d] that there are no more pages of boot data,” the loading process stopped. (*Id.*, 9:49-54.) If the pointer indicated that there was another boot data page, the device “save[d] the current page’s next page pointer for use in processing subsequent pages” and fetched the next page. (*Id.* 9:56-60, 9:60-62 (“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.”).)

Post further taught that “pages [of NVM 120] may be selectively read such that some or all of the non-data pages are skipped over.” (*Id.*, 10:48-49.) A POSITA would have understood this to indicate that the device used the next-page pointer of a verified boot data page to read the next page that contained boot data—Post disclosed no other way to selectively identify the next boot data page—and at a minimum would have found this obvious, because it would have been an efficient and straightforward use of the pointer metadata and would have sped up and reduced the amount of volatile memory used to perform the boot process. (INTEL-1003, ¶¶ 104-107.) In sum, Post disclosed metadata pointers (“*descriptors*”) stored in nonvolatile memory that were *used to fetch* additional fragments of boot data (“*other POR sequencer descriptor fragments*”) by, for example, being used to determine whether to read additional pages from NVM 120, to determine whether to retain or

remove a page from NVM 120, and to selectively read only pages that contained boot data. (INTEL-1003, ¶¶ 102-103, and 108.)

Although Post taught that the control circuitry included “any suitable components, circuitry, or logic” (*id.*, 3:36-38), it did not specifically identify Direct Memory Access (DMA) components. Kao taught techniques for “booting an embedded system” in which “boot codes and controller codes stored in a non-volatile memory may be loaded by a DMA engine.” (INTEL-1004, 2:65-3:3; *id.*, 9:10-13 (“when the PowerUp_Trigger signal is received, the DMA engine 120 may automatically load the boot and controller codes 132, 134”), Fig. 1 (showing boot codes 132, controller codes 134 in nonvolatile NAND flash memory).)

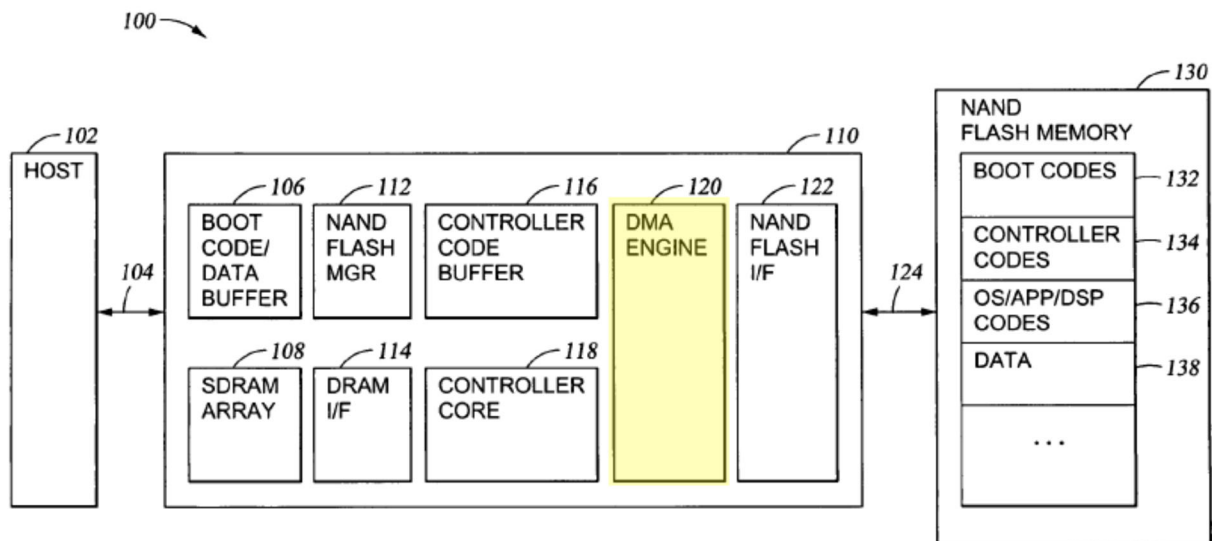


Fig. 1

The DMA engine used data within the boot codes to load controller codes from the nonvolatile NAND flash memory into a DRAM device 110. (*Id.*, 10:28-35 (“the controller codes 134 may be loaded using data stored in the boot codes 132 ... by loading the data in the boot codes 132 directly into a register in the DMA engine 120”), 10:36-50 (describing process for obtaining source address and size of controller codes from data in boot codes), 5:25-32 (“The controller codes 134 may be loaded ... using instructions in the boot codes 132”).) The DMA engine’s transfer of data from nonvolatile memory occurred “without interaction with the host.” (*Id.*, 3:19-21.)

A POSITA would have found it obvious to modify Post in view of Kao’s teachings, such that Post’s controller circuitry 112 included a DMA engine that used Post’s pointer metadata to load pages of boot data subsequent to the first identified boot data page from nonvolatile memory 120 into RAM 114. (INTEL-1003, ¶ 111.) A POSITA would understand that in this combination, the next-page pointer metadata—as used by a DMA engine in identifying a next page of boot data—would be a “DMA descriptor.” (INTEL-1003, ¶ 111.) Kao provided a teaching, suggestion, or motivation for making this combination because Kao teaches that using DMA to perform memory transfers offloads work from the processor, thus reducing the processor’s workload in booting an embedded system and also 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, ¶ 112.)

These teachings from Kao are consistent with other teachings in the art that DMA techniques improve the 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 the workload of the processor to be a “universal” motivation within the field of embedded systems, which further motivates the combination of Kao’s teachings with Post. *Intel*, 2023 WL 2469631, at *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 because 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 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, ¶ 114.)

This combination would also have been obvious because Kao disclosed that its DMA techniques had been used to improve one device (an embedded system including one or more processors, INTEL-1004, 3:22-27), and a POSITA would have recognized that these techniques improved Post’s similar device (an SoC including one or more processors and NVM (*see* INTEL-1005, 3:33-38, 2:60-3:2; *supra* Sec. VI.B.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, ¶ 115.) Additionally, both Post and Kao addressed the problem of transferring data from NVM during a boot of an embedded system, and Kao’s DMA engine 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 this combination as involving the simple substitution of one known element (DMA techniques and circuitry, as taught 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, ¶ 116.)

Wherein the POR sequencer descriptor comprises ... a system firmware.

Post disclosed that the boot data in NVM 120 included firmware, such as a first-stage bootloader. “The program code may be executed to begin loading additional

information from NVM 120 into memory 114. This additional information may include firmware, such as a first stage bootloader....” (INTEL-1005, 3:55-61.) “[T]he firmware (or other data) retrieved from NVM 120 by the mask ROM code may sometimes be referred to as ‘boot data.’” (*Id.*, 3:55-65.)

ii. Detecting a POR sequencer descriptor in a nonvolatile memory;

Post disclosed that its boot data and associated metadata were stored in nonvolatile memory 120. (INTEL-1005, 3:52-65, 4:27-29, 5:9-23.) Post also disclosed searching for and identifying boot data (“*detecting a POR sequencer descriptor*”) in the nonvolatile memory. (INTEL-1005, 8:34-35, 7:34-50, 8:7-9:2; *id.*, 6:22-24.) Post’s electronic device “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.)

Further, the electronic device 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, 6:64-7:14 (’084 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.”.)

4. Claim 1[c]. copying the POR sequencer descriptor from the nonvolatile memory to a random access memory (RAM);

Post disclosed and rendered obvious copying boot data and associated metadata (which, as modified in view of the teachings of Kao and Wilson, was a “POR Sequencer Descriptor”) from nonvolatile memory to a random access memory (“RAM”). (INTEL-1003, ¶ 120.) 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 a 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 ¶¶ 121-124.) 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.”) (emphasis added), 5:19-20 (“memory 220 may illustrate the state of memory 114 (FIG. 1)”); INTEL-1003, ¶¶ 123-124.)

5. Claim 1[d]. verifying an integrity of the POR sequencer descriptor; and

Post disclosed using next page pointers 302, prior page pointers 304, and boot data page indicators 306 to *verify an integrity* of the boot data. (INTEL-1003, ¶¶ 126-130.) “[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, ¶¶ 127, 128, 131.) Post also used error correcting codes to verify an integrity of the boot data. (*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.”); INTEL-1003, ¶¶ 129, 131.) Each of these techniques is an example of Post verifying an integrity of the POR sequencer descriptor. (INTEL-1003 ¶¶ 126-131.)

6. Claim 1[e]. detecting any error in the POR sequencer descriptor;

Post used error correction codes in conjunction with pointer metadata to detect “uncorrectable errors” in the boot code (*“any error in the POR sequencer descriptor”*) indicating that “boot data ha[d] been lost”:

If the error correction results in uncorrectable errors, the boot data for page 2 may be unobtainable, and the electronic device may not have all of the boot data necessary for device operation (unless another copy of the boot data is located elsewhere). In this case, the electronic device may signal a boot failure. Without next page pointer 302 (or other suitable metadata), it would be unclear whether the failed verification indicates that boot data has been lost (as would be the case with page 2) or whether the failed verification indicates that the page is actually a non-data page (as would be the case if verification were attempted on page 3). Thus, the linked-list format provided by the next page indicators can ensure that all of the pages containing boot data are accounted for.

(INTEL-1005, 6:54-7:5; *id.*, 8:52-66 (detecting error of first boot data page being lost); *id.*, 10:8-32 (identifying “a sign of errors in the boot data”); INTEL-1003 ¶¶ 132-134.)

7. Claim 1[f]. wherein the computer-implemented method allows the embedded system to verify an integrity of user data and program code used in a boot process in order to provide a resilient boot up sequence and wherein the computer-implemented method reduces an amount of the nonvolatile memory used in the boot process.

Wherein the computer-implemented method allows the embedded system to verify an integrity of user data and program code used in a boot process in order

to provide a resilient boot up sequence. As described above (*supra* Secs. VI.B.5-6), Post described using error correcting codes and pointer metadata to verify an integrity of boot data. (INTEL-1005, 6:54-7:5, 8:52-66, 10:8-11, ; *id.*, 6:22-24, 8:9-12, 8:27-29, 9:7-11; INTEL-1003, ¶¶ 126-134.) This integrity verification was performed by control circuitry 112, a part of Post’s “*embedded system*.” (INTEL-1005, 8:9-12 (processes of Figs. 4-5 performed by control circuitry 112). Post’s boot data was “any information ... loaded from the NVM based on code stored in a boot ROM” and included “a bootloader” (“*program code used in a boot process*”). (*Id.*, 1:33-43; *id.*, 3:53-65).) As Post’s first-stage bootloader was used to load the device-specific kernel and operating system (*id.*, 3:57-61), this was also *user data*. (INTEL-1003, ¶¶ 136-137.)

Additionally, Post taught that users updated the device firmware—including the boot data, which in turn included the first stage bootloader and “other data”—using a restore operation. (*Id.*, 8:66-9:2 (“the user can perform a restore operation to update the firmware (including the boot data) stored in the non-volatile memory”), 10:30-34; *id.*, 3:57-65 (“firmware (or other data) retrieved from NVM 120 ... referred to as ‘boot data’”).) This user-updated “other data” within the boot data was “*user data and program code*.” (INTEL-1003, ¶¶ 138-139.)

Post further taught that its integrity verification techniques *provided a resilient boot up sequence*. (INTEL-1003, ¶¶ 140-143.) For example, Post described

using an error correcting code on the contents of a page that was expected to contain boot data. (INTEL-1005, 6:54-59.) “If the error correction results in uncorrectable errors, the boot data for page 2 may be unobtainable, and the electronic device may not have all of the boot data necessary for device operation (unless another copy of the boot data is located elsewhere).” (*Id.*, 6:60-64.) Post taught “signal[ing] a boot failure” in that event. (*Id.*, 6:64-65; *id.* 8:62-64 (“Because the electronic device is missing boot data, which may be critical for device operation, the electronic device may signal a boot failure at step 416.”).) In response to a boot failure signal, Post’s device “stop[ped] the bootup process and may display an error message to the user.” (*Id.*, 8:64-66.) The user could then perform a restore operation to “update the firmware (including the boot data)” in nonvolatile memory. (*Id.*, 8:67-9:2.) The electronic device could also “load[] a redundant copy of the boot data from another location in the NVM.” (*Id.*, 7:48-50; INTEL-1003, ¶¶ 142-143.)

By verifying integrity of the boot data, alerting users to errors, and allowing for appropriate firmware updates or other corrective actions, Post’s techniques provided a resilient boot up sequence. (*Id.*, 9:39-46 (“checks on the validity and accuracy of the boot data” were “beneficial, because accurate boot data may be critical to the operation of the electronic device”); *id.*, 1:33-38 (disclosed techniques “ensure[d] the reliable storage and retrieval of such boot data, thereby allowing for consistent and successful startup of an electronic device”); INTEL-1003, ¶ 143.)

Wherein the computer-implemented method reduces an amount of the nonvolatile memory used in the boot process. Post disclosed that the “NVM 120 can be flash memory” and “can include multiple integrated circuits.” (INTEL-1005, 3:1-14; *id.*, 4:33-34 (“The boot data storage area can include any suitable number of integrated circuits, blocks, or pages.”), 6:22-27 (example of Figure 3 showed “blocks in the boot data storage area of NVM 120” that were “in different integrated circuits or NVM packages”).)

For nonvolatile memory that included multiple integrated circuits, Post’s linked-list technique allowed the amount of nonvolatile memory used in the first integrated circuit to be reduced by distributing boot data fragments across other integrated circuits. (*Id.*, 6:38-53 (“the next page pointer [of a boot data page] can reference another [boot data] page in ... the same or a different integrated circuit”); INTEL-1003, ¶ 147.) As illustrated in Post’s Figure 3, Post described linked boot data pages in a first integrated circuit of non-volatile memory (block A), where a next-page pointer 310 in one boot data page referenced a boot data page in a second integrated circuit of non-volatile memory (block B). (INTEL-1005, 6:22-27, Fig. 3.) Because of this linked-list structure, the first integrated circuit of non-volatile memory had a reduced amount “*of the nonvolatile memory used in the boot process*” because other pages of the boot data were stored in a different integrated circuit of non-volatile memory. The ’084 patent describes reducing an amount of nonvolatile

memory in this same way: a first nonvolatile memory 215 (which could be a flash PROM device) had a first POR sequencer descriptor fragment, but other fragments were located on other non-volatile flash memory storage devices 209. (INTEL-1001, 1:33-37, 4:57-5:2, 5:6-11, Fig. 2; INTEL-1003, ¶ 148.)

Additionally, Post's robust integrity verification techniques would have allowed a user to forego redundant copies of the boot data (INTEL-1005, 4:36-41), further reducing the amount of nonvolatile memory used in the boot process. (INTEL-1005, 1:64-67, 6:60-64, 10:24-28; INTEL-1003, ¶ 149.)

Further, Post disclosed storing boot data and associated metadata in NVM 120, which was "commonly used in electronic device[s] for mass storage" (INTEL-1005, 1:6-8), rather than in boot ROM 116 that contained program code and a basic NVM driver 117. (*Id.*, 3:52-4:5.) For example, Post's first stage boot loader and pointer metadata were stored in NVM 120 rather than in boot ROM 116. (*Id.*, 3:52-4:5, 5:58-66.) This allowed the size of boot ROM 116 to be reduced compared what it would have been, had it been storing all boot data and associated metadata. (INTEL-1013, 1:57-60; INTEL-1003, ¶ 146.)

Wilson also taught that the boot data in NVM could be compressed, further reducing the amount of nonvolatile memory that was used in the boot process. (INTEL-1013, 3:30-50, Abstract, 1:61-64, 2:4-8, Fig. 3; INTEL-1003, ¶ 145.) The

combination of Post/Wilson/Kao *reduced an amount of the nonvolatile memory used in the boot process* in these additional ways. (INTEL-1003, ¶¶ 144-149.)

C. Claim 5. The method of claim 1, further comprising activating a POR sequencer error handler if an error is detected in the POR sequencer descriptor.

Post described *activating a POR sequencer error handler if an error is detected in the POR sequence descriptor*. Post taught that its system activated a two-step process for handling errors when they were detected in the boot data. First, when a complete copy of the boot data could not be verified, the boot data collected was discarded and the system attempted to find a redundant copy of the data stored elsewhere in the NVM.

If ... the electronic device fails to confirm the presence of boot data when boot data should be present or fails to verify the prior page indicator, this may be a sign of errors in the boot data. Thus, at step 518, the electronic device may discard all of the boot data read thus far. The availability of mirrored content may then be determined at step 520 to see whether boot data can be obtained from another location in the boot data storage area.

(*Id.*, 10:8-16; *id.*, 8:9-12.) If redundant boot data was available, “the electronic device can begin loading boot data from the other area of the NVM to replace the discarded boot data.” (*Id.*, 10:15-19.) If not, then Post’s system moved to the second part of its error handling process—where, “at step 524, ... a boot failure may be signaled.” (*Id.*, 10:24-30.) The system then “stop[ped] the bootup process and may display an error message to the user.” (*Id.*, 10:30-32; INTEL-1003, ¶¶ 152-154.)

**D. Claim 8. The method of claim 1, further comprising:
performing a register initialization phase in the embedded
system.**

Wilson taught loading initial register values into system 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, ¶¶ 156-157.) For the reasons described in connection with claim 1[b] (*supra* Sec. VI.B.3.i), a POSITA would have found it obvious to *use* “the information to initialize configuration registers,” as explained for claim 1[b], to perform a register initialization phase. (INTEL-1003, ¶¶ 155, 158.)

**E. Claim 9. The method of claim 1, further comprising:
copying system firmware from a storage device to the RAM.**

Post disclosed *copying system firmware from a storage device to the RAM*. (INTEL-1003, ¶ 159.) Post’s first-stage bootloader was used to copy a second stage bootloader (“*system firmware*”). (INTEL-1005, 3:57-65 (“first stage bootloader ... 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.*,

1:39-43.) A POSITA would have understood that second-stage bootloader was stored in NVM 120 because NVM 120 was the only non-volatile storage Post described, and because Post recognized that non-volatile memory commonly served as mass storage. (*Id.*, Fig. 1, 1:6-8 (“NAND flash memory, as well as other types of non-volatile memories (NVMs), are commonly used in electronic device for mass storage.”)).

Further, because the “boot data storage area” of NVM 120 was dedicated to storing the boot data (*id.*, 4:27-30), the second stage bootloader would have been stored outside of that area—for example, in an additional integrated circuit of NVM 120. (*Id.*, 3:12-14.) Post therefore taught storing the second stage bootloader in an integrated circuit of NVM 120 (“*storage device*”) distinct from the integrated circuit storing boot data. At the very least, a POSITA would have found it obvious for the second stage bootloader to be stored in a different integrated circuit, and this would at least have been obvious to try, because there were only two possible storage locations for the second stage bootloader—on an integrated circuit that also stored boot data, or on one or more integrated circuits separate from those storing boot data—and there would have been a reasonable expectation of success in doing so, because Post demonstrated that a POSITA knew how to store and retrieve boot data and bootloader code from multiple non-volatile memory integrated circuits. (INTEL-1003, ¶¶ 161-162.)

Post also disclosed that the second stage was “load[ed]” (“*cop[ied]*”) by the first stage bootloader and was “loaded in memory 114” for execution by control circuitry 112. (INTEL-1005, 3:38-41, 3:57-61.) Memory 114 was RAM. (*Id.*, 3:44-45.)

F. Claims 19 and 20

1. Claim 19[pre]. An article of manufacture, comprising:

Intel does not concede that the preamble is limiting. (INTEL-1003, ¶ 164.) To the extent it is, 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 electronic device or system.” (INTEL-1005, 2:61-67.) Each of these was an example of an “article of manufacture.” (INTEL-1003, ¶ 165.)

2. Claim 19[a]. a non-transitory computer-readable medium having stored thereon instructions operable 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, ¶¶ 166-168.) These teachings render obvious claims 1[a]-1[e], as discussed (*supra* Secs. VI.B.2-6), which are substantively identical to the limitations of claims 19[b]-1[f] (the method of claim 19).

3. Remaining limitations of claims 19 and 20

The remaining limitations of claims 19 and 20 recite limitations that are analogous to those of claims 1 and 5 and are obvious for the reasons discussed above. (INTEL-1003, ¶¶ 169-176.) To summarize:

Claim Limitation²	Basis for Obviousness
Claim 19[b]	<i>See</i> Claim 1[a]
Claim 19[c]	<i>See</i> Claim 1[b]
Claim 19[d]	<i>See</i> Claim 1[c]
Claim 19[e]	<i>See</i> Claim 1[d]
Claim 19[f]	<i>See</i> Claim 1[e]
Claim 20	<i>See</i> Claim 5

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 '084 patent is asserted in the following litigation:

BiTMICRO LLC v. Intel Corporation, 23-cv-00625-JCS (NDCA).

The '084 patent is also related to U.S. Pat. No. 10,120,694, which is being challenged in a concurrently-filed petition. *See* IPR2023-00787.

² Appendix B to INTEL-1003 lists the challenged claims in full.

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 20005-3960, 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 12,427 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 9,858,084 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

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