

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTEL CORPORATION,
Petitioner

v.

BiTMICRO LLC,
Patent Owner.

IPR2023-00785
U.S. Patent No. 9,858,084

**DECLARATION OF DONALD ALPERT, PH.D.
IN SUPPORT OF *INTER PARTES* REVIEW OF U.S. 9,858,084**

TABLE OF CONTENTS

I.	INTRODUCTION	1
II.	QUALIFICATIONS AND EXPERTISE	3
III.	COMPENSATION	6
IV.	PREVIOUS EXPERT WITNESS EXPERIENCE.....	6
V.	MATERIALS CONSIDERED AND BASIS OF OPINIONS	6
VI.	LEVEL OF ORDINARY SKILL IN THE ART.....	8
VII.	UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES.....	10
VIII.	OVERVIEW/TUTORIAL REGARDING TECHNOLOGY.....	13
	A. Embedded Systems and Their Components.....	13
	B. Power-On Reset and Boot	15
	C. System Registers	17
	D. Direct Memory Access (“DMA”)	18
IX.	THE CHALLENGED ’084 PATENT	25
	A. Summary of the ’084 Patent.....	25
	B. Prosecution History of the ’084 Patent	29
	C. Claim Construction	31
X.	CLAIMS 1, 5, 8-9, 19 AND 20 ARE RENDERED OBVIOUS BY THE COMBINATION OF POST, WILSON, AND KAO.....	32
	A. Post	32
	B. Kao	37
	C. Wilson.....	38
	D. Overview of the Combination	41
	E. Independent Claim 1	42
	1. Claim 1 Preamble.....	42
	2. Claim 1, Limitation a	43
	3. Claim 1, Limitation b.....	48
	a. Power on reset [POR] sequencer descriptor.....	49
	b. Detecting a POR sequencer descriptor	71

4.	Claim 1, Limitation c	72
5.	Claim 1, Limitation d	77
6.	Claim 1, Limitation e	82
7.	Claim 1, Limitation f	84
	a. First part	84
	b. Second part	90
F.	Dependent Claim 5	94
G.	Dependent Claim 8	98
H.	Dependent Claim 9	101
I.	Independent Claim 19	103
	1. Claim 19 Preamble	103
	2. Claim 19, Limitation a	104
	3. Claim 19, Limitation b	106
	4. Claim 19, Limitation c	107
	5. Claim 19, Limitation d	107
	6. Claim 19, Limitation e	107
	7. Claim 19, Limitation f	108
	8. Claim 19, Limitation g	108
J.	Dependent Claim 20	109
XI.	CONCLUSION	109

I. INTRODUCTION

1. My name is Donald Alpert, and I have been retained by Perkins Coie LLP on behalf of Intel Corporation (“Intel” or “Petitioner”) in the above-referenced *inter partes* review proceeding to evaluate United States Patent No. 9,858,084 to Manlapat et al. (“the ’084 Patent”) against certain references that predate the earliest possible priority date of March 15, 2013 (the “Earliest Priority Date”) for the ’084 Patent. The ’084 Patent is attached as Exhibit INTEL-1001 to the Petitioner’s Petition for *Inter Partes* Review of U.S. Patent No. 9,858,084 based on United States Patent No. 8,799,555 (“Post”), United States Patent No. 7,861,073 (“Wilson”), and United States Patent No. 7,490,177 (“Kao”). I am informed that the Petitioner seeks review of claims 1, 5, 8, 9, 19, 20 of the ’084 Patent (collectively, the “challenged claims”). I have also been informed that the ’084 Patent is currently assigned to BiTMICRO LLC (“BiTMICRO” or “Patent Owner”). As detailed in this declaration, it is my opinion that each of the challenged claims is rendered obvious by prior art references that predate the Earliest Priority Date.

2. I am also providing a declaration in support of Petitioner’s Petition for *Inter Partes* Review of U.S. Patent No. 10,120,694, which shares the same specification as the ’084 Patent and has highly similar claims. The Petition for *Inter Partes* Review of U.S. Patent No. 10,120,694 relies on the same Post, Wilson, and Kao prior art references as discussed in the ’084 Patent Petition. My declarations

for these matters may contain similar content given this similarity of subject matter and, where appropriate, I have incorporated materials from one of my declarations into the other.

3. I am over 18 years of age. I have personal knowledge of the facts stated in this Declaration and if requested by the Patent Trial and Appeal Board (“PTAB” or “Board”), I am prepared to testify competently about my opinions expressed herein.

4. I have reviewed and am familiar with the specification and the claims of the '084 Patent. In general, I will cite to the specification of a United States patent using the following formats: (Patent No. at Col:Line Number(s)) or (Patent No. at Paragraph Number(s)). For example, the citation ('084 Patent at 1:1-10) points to the '084 Patent specification at column 1, lines 1-10. Also, for convenience, I use italics to denote limitations from the challenged claims.

5. All of the opinions contained in this Declaration are based on the documents I reviewed and my knowledge and professional judgment. In forming the opinions expressed in this Declaration, I reviewed the documents listed below. I have also reviewed and am familiar with the other materials referred to in this Declaration. To the best of my knowledge, the documents mentioned above are true and accurate copies of what they purport to be and are the kinds of materials that an individual

with expertise in this field at the relevant time period would reasonably rely on in formulating opinions, such as those set forth in this declaration.

6. I have been asked to provide my technical opinions regarding how a person of ordinary skill in the art (“POSITA”) would have understood the claims of the ’084 Patent at the time of the Earliest Priority Date. For purposes of whether the teachings of the prior art anticipate or render obvious the claims of the ’084 Patent, I have been asked to assume the date of March 15, 2013, the filing date of the earliest application in the priority chain of the ’084 Patent, for the analysis in this document. I have also been asked to provide my technical opinions on how concepts in the ’084 Patent specification relate to claim limitations of the ’084 Patent. In reaching the opinions provided herein, I have considered the ’084 Patent, its prosecution history, and the references cited above and have drawn as appropriate on my own education, training, research, knowledge, and personal and professional experience.

II. QUALIFICATIONS AND EXPERTISE

7. I am an independent consultant and the president of Camelback Computer Architecture LLC, through which I have provided computer architecture consulting services for twenty years. In formulating my opinions, I have relied on my knowledge, training, and experience in the relevant field, which I will summarize briefly. A more detailed summary of my background, education, experience, and publications is set forth in my curriculum vitae (“CV”), which is provided as

Appendix A. My CV also includes a list of all other cases in which, during the previous 4 years, I testified as an expert at trial or by deposition.

8. I have 45 years of academic and industrial experience in applying, designing, studying, teaching, and writing about microprocessors and computer systems. I received an Electrical Engineering Ph.D. degree in 1984 from Stanford University. I earlier received an Electrical Engineering B.S. degree from MIT in 1973 and an Electrical Engineering M.S. degree from Stanford University in 1978. I have taught classes in computer architecture at Stanford, Tel Aviv, and Arizona State Universities.

9. From 1976 to 1977, I worked at Burroughs Corporation, where I designed peripheral interface controllers, including those for serial data communications based on Intel 8080 microprocessor components. From 1980 to 1989, I was the lead architect for the design of three high-performance microprocessors at Zilog and National Semiconductor. Later, at Intel, I was the lead architect of the Pentium® Processor from 1989 to 1992 and of the 815 chipset from 1999 to 2000, both of which became the most widely used PC components of their time. The 815 chipset comprised two components: (1) a memory controller hub (MCH) that included a graphics controller and memory controller with interfaces to the CPU, 133 MHz SDRAM system memory modules, an optional, external graphics controller and (2) an I/O controller hub (ICH) that included various I/O controllers

(e.g., network, hard drive, USB) for system peripheral devices, as well as Direct Memory Access (DMA) controllers. Additionally, I served as co-manager for the Itanium processor design from 1993 to 1997.

10. I am a Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), and served as the chair of the IEEE Technical Committee on Microprocessors and Microcomputers from 1999 to 2000. I was the keynote speaker at the first Cool Chips conference, dedicated to the study of low-power microprocessors and systems. I have given invited lectures at several universities, and published ten papers in professional journals and conference proceedings. My paper entitled “Architecture of the Pentium Processor” was selected as best paper in IEEE Micro for 1993. I am a named inventor on over 30 U.S. patents that pertain to microprocessors, computer systems, and related technology.

11. I have reviewed the '084 patent, and I am familiar with its subject matter, which is within the scope of my education and professional experience. Based at least on my background in academia, industry, and consulting, I am familiar with the issues and technology relating to processors, chipsets, memory, and peripheral devices for computer systems. I have personally analyzed, developed, and tested such computer components and systems. More specifically, the Pentium® Processor and 815 chipset for which I was the lead architect at Intel implemented DMA controllers and various features for system initialization.

12. My curriculum vitae, including my publications and patents, is submitted herewith as Appendix A.

III. COMPENSATION

13. In connection with my work as an expert, I am being compensated at an hourly rate of \$650 for consulting services including time spent testifying at any hearings that may be held. I am also being reimbursed for reasonable and customary expenses associated with my work in this case. I receive no other form of compensation related to this case. My compensation is not dependent or otherwise contingent upon the specifics of my testimony or the outcome of this action.

IV. PREVIOUS EXPERT WITNESS EXPERIENCE

14. My CV attached as Appendix A to this Declaration contains a list of cases in which I have testified at trial, hearing, or by deposition within the preceding four years.

V. MATERIALS CONSIDERED AND BASIS OF OPINIONS

15. In forming the opinions set forth in this Declaration, I have considered and relied upon my education, experience, and knowledge of the relevant fields. I have also reviewed and considered the '084 Patent and its prosecution history. The '084 Patent and its prosecution history are attached as exhibits INTEL-1001 and INTEL-1002 to the above-referenced IPR petition. Additionally, except as noted, I have relied on and reviewed the following documents and materials cited in this

Declaration, all of which are attached as exhibits to the above-referenced IPR petitions.

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 (This document)
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-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-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”)

16. I reserve the right to supplement this Declaration and rely upon any additional information, deposition testimony, documents, or materials that may be provided to me or that are relied upon by any of Respondent's experts or witnesses.

17. I also reserve my right to supplement this Declaration and rely upon additional information that becomes available to me.

18. In connection with live testimony in this proceeding, should I be asked to provide it, I may rely on visual aids and demonstratives to demonstrate the bases of my opinions, such as claim charts, patent drawings, excerpts from patent specifications, file histories, interrogatory responses, deposition transcripts and exhibits, as well as charts, diagrams, videos and animated or computer-generated video. The demonstrative exhibits I may use to accompany any testimony I may give may also include documents, testimony, and other evidence cited in my Declaration, portions of such evidence, and other portions of my Declaration and the attached exhibits.

VI. LEVEL OF ORDINARY SKILL IN THE ART

19. I am told that certain issues relating to the validity of the '084 Patent must be judged from the perspective of a person of ordinary skill in the relevant art, as I discuss below. I have been asked to define the level of a "person of ordinary skill in the art" or "POSITA" at the time the alleged invention as claimed was made, that is, as of the Earliest Priority Date (March 15, 2013). In deciding the level of

ordinary skill, I have considered the following factors, which I have been informed are relevant to the determination of ordinary skill in the art:

- i) the levels of education and experience of persons working in the field;
- ii) the types of problems encountered in the art;
- iii) the prior art solutions to these problems;
- iv) the rapidity with which innovations are made; and
- v) the sophistication of the technology.

20. In my opinion, for the purposes of the '084 Patent, a POSITA, at the Earliest Priority Date, would have had a bachelor's degree in electrical engineering or computer engineering and at least two years of experience working in the field of embedded system design, including booting of such systems, or an equivalent combination of education, work, and/or experience. My opinion as to this level of ordinary skill in the art would be the same even if the Board were to determine that the time of the alleged invention is March 17, 2014, which is the filing date of the application that issued as the '084 patent.

21. My opinion below explains how a POSITA would have understood the technology described in the references I have identified in this declaration as of the Earliest Priority Date.

VII. UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES

22. I am not a lawyer and will not provide any legal opinions. Although I am not a lawyer, I have been advised that certain legal standards are to be applied by technical experts in forming opinions regarding the meaning and validity of patent claims.

23. I understand that a patent claim is invalid if it is anticipated or obvious in view of the prior art, and that a claim can be unpatentable even if all of the requirements of the claim cannot be found in a single prior-art reference. I further understand that invalidity of a claim requires that the claim be anticipated or obvious from the perspective of a person of ordinary skill in the art at the time the invention was made.

24. I have been informed that a patent claim is invalid if it would have been obvious to a person of ordinary skill in the art. In analyzing the obviousness of a claim, I understand the following factors may be considered: (1) the scope and content of the prior art; (2) the differences between the prior art and the claims; (3) the level of ordinary skill in the art; and (4) any so called “secondary considerations” of non-obviousness if they are present. Such secondary considerations include: commercial success of products covered by the patent claims; a long-felt need for the invention; failed attempts by others to make the invention; copying of the invention by others in the field; unexpected results achieved by the invention as

compared to the closest prior art; praise of the invention by the infringer or others in the field; the taking of licenses under the patent by others; expressions of surprise by experts and those skilled in the art at the making of the invention; and the patentee proceeded contrary to the accepted wisdom of the prior art. I am not aware of any evidence of secondary considerations of non-obviousness relevant to the '084 patent. I reserve the right to supplement this Declaration if Patent Owner introduces evidence of secondary considerations of non-obviousness.

25. I understand that to prove that prior art or a combination of prior art renders a patent obvious, it is necessary to:

- i) identify the particular references that, singly or in combination, make the patent obvious;
- ii) specifically identify which elements of the patent claim appear in each of the asserted references; and
- iii) explain why a person of ordinary skill in the art would have combined the references, and how they would have done so, to create the inventions claimed in the patent.

26. I further understand that exemplary rationales that may support a conclusion of obviousness include:

- i) combining prior art elements according to known methods to yield predictable results;

- ii) simple substitution of one known element for another to obtain predictable results;
- iii) use of known technique(s) to improve similar devices (methods or products) in the same way;
- iv) “obvious to try” – choosing from a finite number of identified, predictable solutions with a reasonable expectation of success;
- v) known work in one field of endeavor may prompt variations of the work for use in either the same field or a different field based on design incentives or other market forces if the variations are predictable to a person of ordinary skill in the art; and
- vi) some teaching, suggestion, or motivating in the prior art that would have led a person of ordinary skill in the art to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

27. I have been informed that, in considering obviousness, hindsight reasoning derived from the patent-at-issue may not be used.

28. I understand that the preamble to a patent claim is not generally considered limiting if the body of the claim defines a structurally complete invention and the preamble is used to provide context, such as a purpose or intended use for the invention. I understand, however, that a preamble may be limiting if it defines

terms used later in the body of the claim, or if the remainder of the claim text is otherwise incomplete without the preamble. I understand that if the preamble is not limiting it need not be considered as part of the analysis for determining whether the remainder of the claims are valid or infringed.

VIII. OVERVIEW/TUTORIAL REGARDING TECHNOLOGY

A. Embedded Systems and Their Components

29. The phrase “embedded system” is typically used to refer to a microprocessor that is integrated with other equipment to perform a specific function. The Microsoft Computer Dictionary (5th Ed.) defines embedded system as:

Microprocessors used to control devices such as appliances, automobiles, and machines used in business and manufacturing. An embedded system is created to manage a limited number of specific tasks within a larger device or system.

(INTEL-1007, at 190-191.) Other definitions are similar: “embedded systems represent a class of dedicated computer systems designed for specific purposes.”

(INTEL-1010 at 1.) The ‘694 patent focuses on the inclusion of a processor as the critical component of an embedded system: “The term embedded system will be used to refer to a single CPU or a multiprocessor system in this paper.” (INTEL-1001 at 1:26-28.)

30. Because an embedded system is based on a processor, they also naturally included components found in general-purpose computers, such as memories and storage devices. The prior art identifies processors, volatile RAMs and non-volatile memories as essential components of any embedded system. (INTEL-1009 at 195-209; INTEL-1012 at 61-72.) The treatises described the use of controllers for RAM and non-volatile memories. (INTEL-1009 at 110, 196-199, 204-205, 208; INTEL-1012 at 62, 66, 72.) They also detailed the use of the standard Serial ATA (SATA) interface to connect hard disks and SSDs (INTEL-1009 at 209-214; INTEL-1012 at 72-73) and USB and PCIe interfaces suitable for connecting storage devices generally (INTEL-1009 at 214; INTEL-1012 at 74-89). In its discussion of the prior art, the '694 patent acknowledges that embedded systems that incorporated RAM memory, non-volatile memories, and mass storage devices were conventional:

A conventional embedded system contains a nonvolatile memory which is used to store firmware that is read and executed by a CPU to initialize the system. The nonvolatile memory can be a read-only memory (ROM), one-time programmable memory (OTP), programmable read only memory (PROM), electrically erasable PROM (EEPROM), or a flash PROM (FPROM). The nonvolatile memory can be internal (on-chip) or external (off-chip) to the controller chip. The firmware stored in the nonvolatile memory can be a boot code loader, a basic input/output system (BIOS), an operating system (OS), or an application firmware. The boot code loader is a small code executed by a CPU to do minimum initialization of the

system. After initialization, the boot code loader copies the BIOS or OS or application firmware from a mass storage device external to the embedded system to the random access memory (RAM) of the system.

(INTEL-1001, 1:31-47.)

31. Other conventional features of embedded processors include DMA controllers, storage controllers, a CPU bus connecting the processor and memory devices, reset controllers, and interfaces for debugging. Each of these was already documented in the prior art. (K. Arnold, Embedded Controller Hardware Design (2001), at 117-124 (CPU bus), 175-178 (DMA controllers); INTEL-1009 at 137-138 (controllers for optical and magnetic storage devices), 166-168 (processor bus), 214-235 (DMA controllers); INTEL-1005, at 556 (reset controller); INTEL-1008 at 157-59 (debugging interface); INTEL-1010, at 51-52 (debugging interfaces); INTEL-1012, at 72-73 (host storage controllers), 469 (debugging interface).)

B. Power-On Reset and Boot

32. 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.”).)

33. 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-1009, 5 (“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.”), 16 (“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”).)

34. 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.”).)

C. System Registers

35. 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).” (*Id.*; 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.”.)

36. 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.”).)

D. Direct Memory Access (“DMA”)

37. 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”).). An example provided by the 2005 textbook Designing Embedded Hardware (INTEL-1008) explains why the processing workload for such processor-driven data transfers can be high:

Let’s say you want to read in 100M from disk and store it in memory. You have two options.

One option is for the processor to read one byte at a time from the disk controller into a register and then store the contents of the register to the appropriate memory location. For each byte transferred, the processor must read an instruction, decode the instruction, read the data, read the next instruction, decode the instruction, and then store the data. Then the process starts over again for the next byte.

(INTEL-1008, 20.)

38. Direct Memory Access (“DMA”) techniques were one way to offload this work from the core system processor. (*Id.* (“DMA is a way of streamlining transfers of large blocks of data between two sections of memory, or between memory and an I/O device.”).) The continuation of the example provided in the previous paragraph shows how DMA offloads work from the system processor:

Let’s say you want to read in 100M from disk and store it in memory. You have two options.

The second option in moving large amounts of data around the system is DMA. A special device, called a *DMA Controller (DMAC)*, performs high-speed transfers between memory and I/O devices. 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.

(*Id.* (emphasis original).)

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.”).) 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-1015, 32 (“DMA is a common

hardware feature for transferring data from one memory location to another without putting a load on the CPU.”.)

40. The 2008 industry reference book Embedded Hardware (INTEL-1009) illustrates typical DMA flows, in which a DMA controller performs transfers between system memory and/or peripherals:

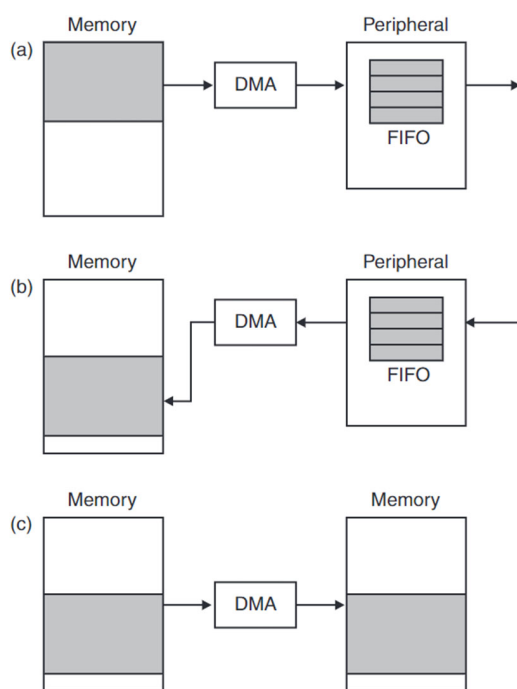


Figure 5.17: Typical DMA flows.

(INTEL-1009, 216.) DMA controllers can perform transfers of code as well as data. (*Id.* (“[A] DMA transfer doesn’t always have to involve data. We can use code overlays to improve performance, configuring the DMA controller to move code into L1 instruction memory before execution. The code is usually staged in larger external memory.”).)

41. DMA controllers need some basic information to perform a data or code transfer, for example a source and/or destination address for the data or code to be transferred, and an indication of the size of the data or code to be transferred:

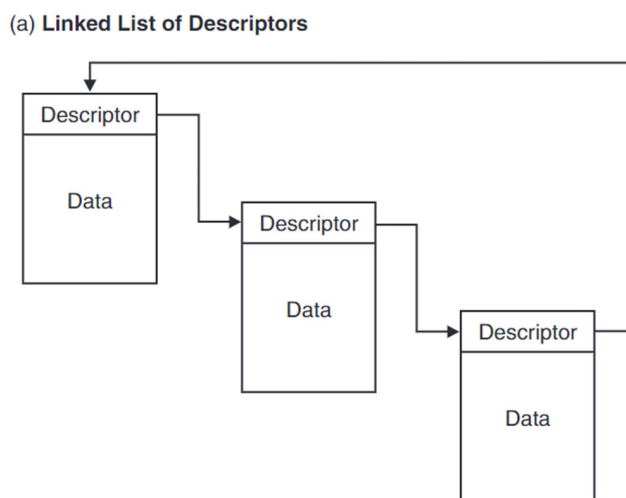
For any type of DMA transfer, we always need to specify starting source and destination addresses for data. In the case of a peripheral DMA, the peripheral's FIFO serves as either the source or the destination. When the peripheral serves as the source, a memory location (internal or external) serves as the destination address. When the peripheral serves as the destination, a memory location (internal or external) serves as the source address.

In the simplest MemDMA case, we need to tell the DMA controller the source address, the destination address and the number of words to transfer. With a peripheral DMA, we specify either the source or the destination, depending on the direction of the transfer.

(*Id.*, 208; *see also* INTEL-1008, 21 (“A DMA transfer begins by the processor configuring the DMAC for the transfer. This setup involves specifying the source, destination, and size of the data, as well as other parameters.”))

42. 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.)

43. Numerous examples of the use of DMA controllers in the boot process, and of DMA descriptors that identify data to be transferred by the DMA controller were known before the earliest filing date of the ’084 patent. For example, U.S. Patent No. 8,125,243, published on February 28, 2012, used a DMA controller (3:45-46) to transfer device configuration data from a non-volatile memory to device

configuration registers (3:62-65, 4:11-17). To carry out these transfers, the DMA controller acted on a series of DMA transaction descriptors stored in memory that identified each of the transfers of configuration data that were to be performed. (4:46-56.) U.S. Patent No. 8,996,926, filed on October 15, 2012, described a DMA controller used to transfer data (1:5-28) or boot code (5:41-44). The DMA controller used a chained set of DMA transaction control values, as illustrated in Figure 3A, that are chained together to define a sequence of transfer operations that the DMA controller was to perform.

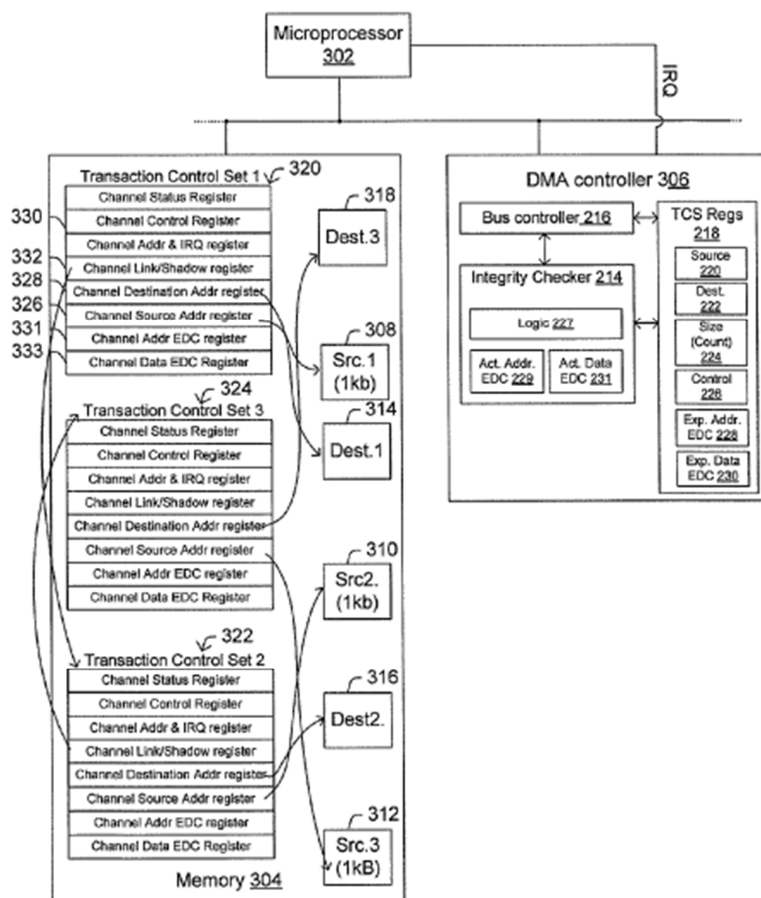


FIG. 3A

IX. THE CHALLENGED '084 PATENT

A. Summary of the '084 Patent

44. The '084 patent describes an embedded system with “a single or multiple Central Processing Units” and is directed to “a method and device for booting such a system directly from a storage device.” (INTEL-1001, 1:17-20.) As depicted in Figure 2 of the patent, the embedded system includes at least one CPU (in red, below), at least one DMA controller (in yellow), and a power on reset (POR)

sequencer (in blue) that “makes use of a POR Sequencer Descriptor.” (*Id.*, 2:49-50.)

The POR sequencer descriptor is “a preassembled descriptor that is stored in a nonvolatile memory of the system.” (*Id.*, 2:50-52.) It “contains register information and direct memory access (DMA) controller descriptors. The register information are [sic] read and used by the POR Sequencer to update the configuration registers of the system. The DMA descriptors are loaded by the POR Sequencer to the DMA controllers of the system.” (*Id.*, 2:53-58.)

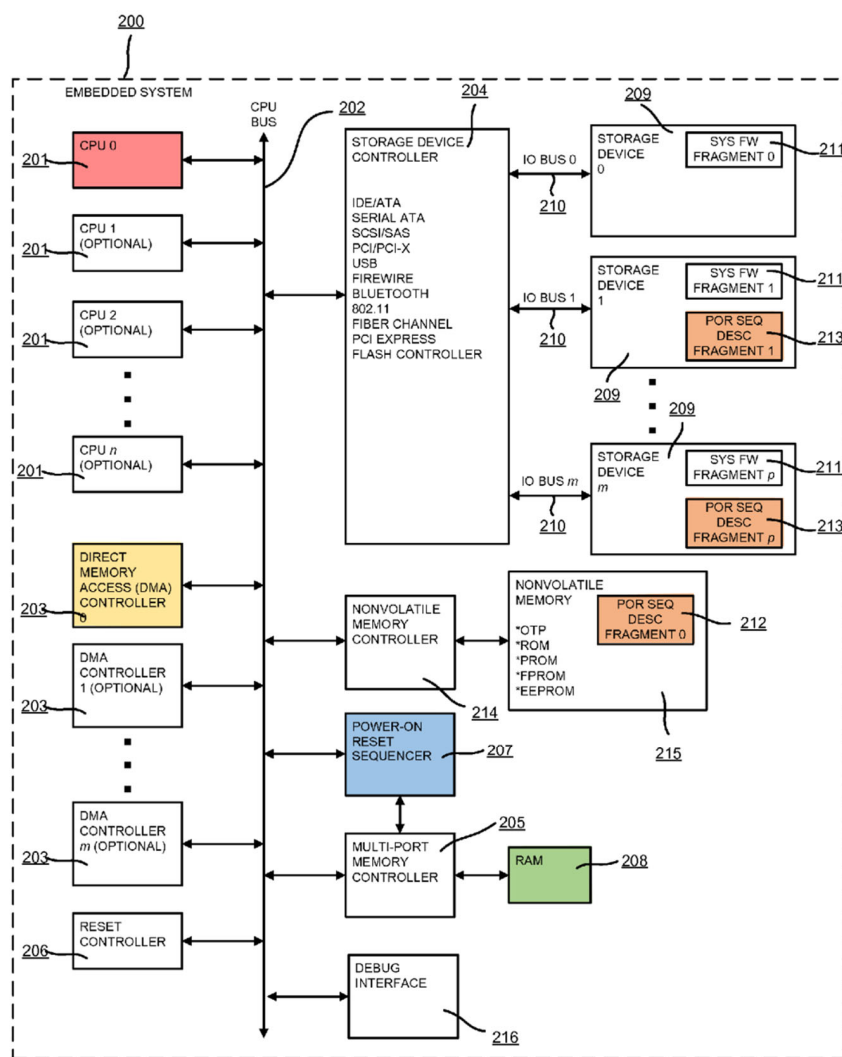
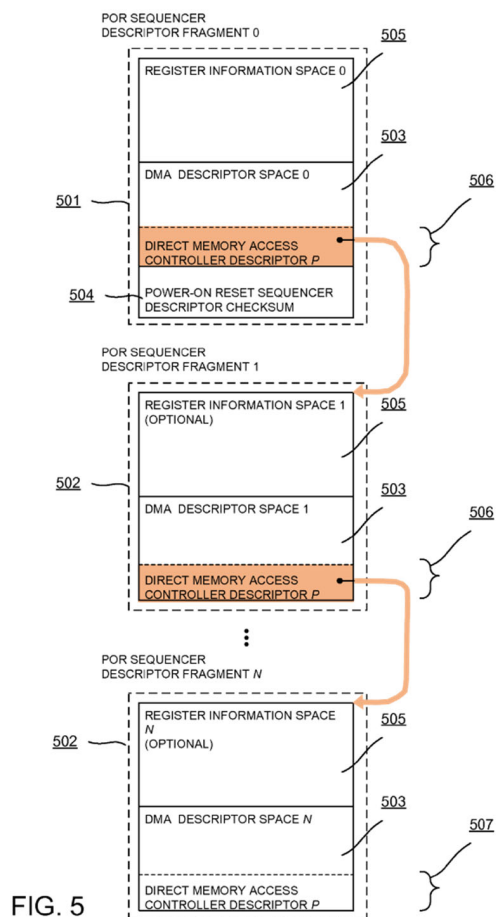


FIG. 2

45. The '084 patent proposes that “[w]hen the POR sequencer descriptor doesn’t fit in the nonvolatile memory of the system, the descriptor is divided into fragments and distributed across the nonvolatile memory and storage devices attached to the system.” (*Id.*, 6:34-37.) This is illustrated in Figure 2, which depicts multiple storage devices each containing a fragment of the POR sequencer descriptor. The fragments are connected in a “linked list,” as illustrated in Figure 5,

(*id.*, 4:30-31), where each fragment contains a DMA controller descriptor (in orange) that “points to the next POR sequencer descriptor fragment.” (*Id.*, 6:44-46.)



46. The POR sequencer “controls the overall boot process of the embedded system.” (*Id.*, 2:37-38.) At system power-on, it, and other system components, are brought out of reset. (*Id.*, 6:58-63.) The POR sequencer copies the first fragment of the POR sequencer descriptor to memory. (*Id.*, 6:66-7:2.) It then “checks the integrity of POR sequencer descriptor ... by comparing [a] computed checksum to the POR sequencer descriptor checksum” (*id.*, 7:2-5), which is contained, as shown in Figure 5, in the first fragment. (*Id.*, 6:40-42.) Using the method illustrated in

Figure 7 (*id.*, 7:19-20), the POR sequencer copies the register information from the fragment in memory to update the corresponding configuration registers in the embedded system. (*Id.*, 7:19-29.) When all of the register configuration information has been processed, the POR sequencer writes the DMA descriptor to the DMA controller and activates it so that it will initiate transfer of the next POR sequence descriptor fragment from storage to the RAM. (*Id.*, 7:29-37.) This process continues until the configuration register data in each of the POR sequence descriptor fragments has been processed. (*Id.*, 7:39-46.) A similar method, illustrated in Figure 8 (*id.*, 7:50-52), is used for retrieving firmware code, which may also be stored in multiple fragments, as shown in Figure 5. (*Id.*, 7:47-8:15.)

47. The '084 patent states that if a CRC error is found in the configuration register data “a POR sequencer error handler is activated.” (*Id.*, 7:7-9, *see also, id.*, 8:2-3 (if errors in the firmware are identified “an error handling routine will be executed”).) The nature of these error handling routines is not specified. If the POR sequencer cannot find the primary copy of the POR sequencer descriptor it “will try to detect a second copy.” (*Id.*, 7:9-11.)

B. Prosecution History of the '084 Patent

48. I have reviewed the file containing the prosecution history of the '084 patent. I did not identify anything in the file that significantly impacts my

interpretation of the claims or patentability analysis. Nevertheless, I will summarize the more significant interactions between the applicant and the US Patent Office.

49. Initially the examiner rejected the preliminarily amended claims (1-20), over U.S. Patent No. 7,921,237 to Holland. (INTEL-1002, at 115-119 (Non-Final Rejection sent Jan. 11, 2016).) In response, Applicant amended independent claims 1, 10 and 19 to insert this additional language defining components of the “POR sequencer descriptor” claim element:

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 sequence descriptor fragments, and a system firmware.

(*Id.*, at 103-114 (Amendment of 10 April 2016).)

50. The pending claims were allowed in 2016 (*id.*, at 88-93), but the applicant then filed multiple new information disclosures. In an office action sent on November 1, 2016, the examiner rejected claims 1 through 18 for patentability under section 101 but allowed claims 19 and 20. (*Id.*, at 65-70 (Office Action sent Nov. 1, 2016).) The applicant responded by adding this recitation 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.*, at 49-63 (Amendment dated Feb. 1, 2017).) Other modifications were made to claim 10, not at issue here. The examiner then issued a notice of allowance on May 12, 2017. (*Id.* at 45-48.) The applicant filed a request for continuing examination minor amendments to claim 19 on August 12, 2017. (*Id.*, at 28-42.) The examiner issued a final notice of allowability on August 21, 2017. (*Id.* at 23-27.)

C. Claim Construction

51. I understand that “claim construction” is the process of determining a patent claim’s meaning. I understand that the Board applies the same claim construction standard used in district courts which seeks to give claim terms their plain and ordinary meaning to a POSITA at the time of the alleged invention in view of the claims, specification, and prosecution history. I also understand that if the patent applicant gave a term a special meaning in the specification or during prosecution, then the special meaning should be used. Lastly, I understand that claim construction is a matter of law and that the Board will ultimately determine the final claim construction.

52. I do not believe that any term in the challenged claims requires construction.

X. CLAIMS 1, 5, 8-9, 19 AND 20 ARE RENDERED OBVIOUS BY THE COMBINATION OF POST, WILSON, AND KAO

A. Post

53. Post described a method for booting an embedded device such as an iPod, an iPhone or other cellphone, a computer, or other electronic device. (INTEL-1005, 2:60-67.) An exemplary device, shown in Figure 1, includes a system on a chip (SOC) 110 and a non-volatile memory (NVM) 120. The SOC includes control circuitry, which may include one or more processors (*id.*, 3:36-38), memory, a boot ROM, and an interface to the NVM. (*Id.*, 3:33-34.)

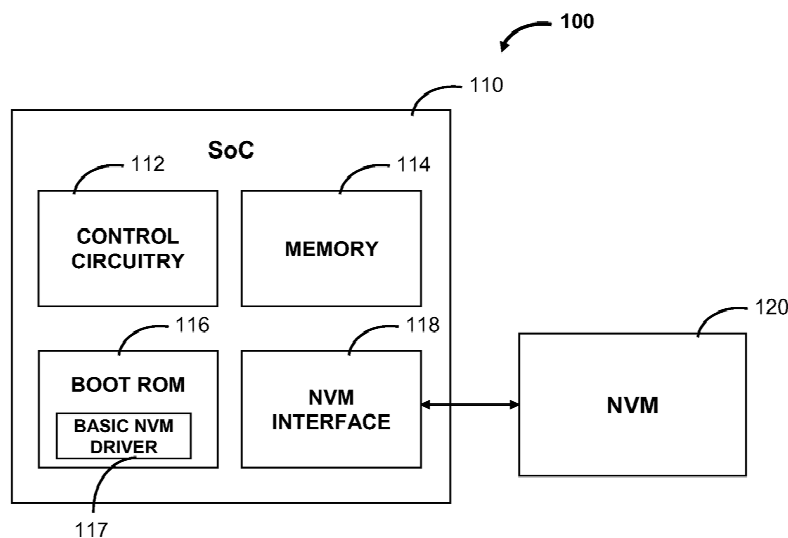
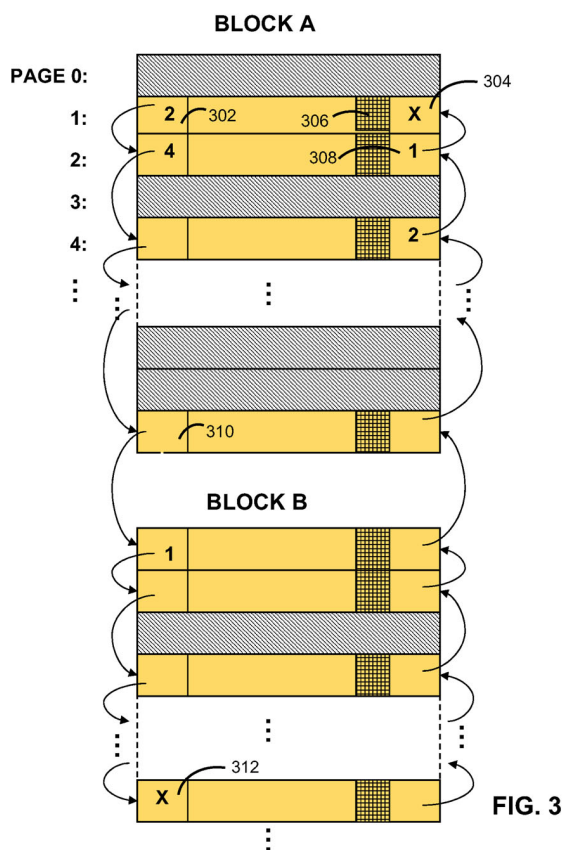


FIG. 1

54. Post describes a process for organizing boot data in the NVM. “[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.) The use of nonconsecutive pages of NVM is

illustrated in Post’s Figure 3, which shows a discontinuous sequence of NVM pages that contain boot data. Each page (highlighted in yellow) is linked to the next page of boot data by a next page pointer 302 and to a previous page of boot data by a prior page pointer 304. (*Id.*, 6:28-33, 6:38-46.) In addition to the pointers, a boot page indicator 306 is set in each page that contains boot data. (*Id.*, 6:35-37, 7:61-64.) As shown in Figure 3, a “next page pointer can reference another page in a different block of the same or a different integrated circuit.” (*Id.*, 6:51-53.)



55. “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; *see also*,

id., 3:53-55.) Operating pursuant to this code, the control circuitry loads boot data from the NVM 120 into the memory 114.

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 boot loader, 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. For clarity, ... the 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.)

56. A process implemented in the boot ROM code for retrieving boot data from the NVM is illustrated in Post’s Figures 4 and 5. (*Id.*, 8:15-20 (Fig. 4 illustrates the process of finding first page of boot data), 9:14-18 (Fig. 5 illustrates process for loading the rest of boot data).) The first step in the process is to find the beginning of the boot data. At steps 404 and 406 “the electronic device may begin executing mask ROM code [the program code],” and “read an initial page from the non-volatile memory.” (*Id.*, 8:27-31.) “[A]t step 408, the electronic device may determine whether the initial page contains boot data.” (*Id.*, 8:34-35.)

[T]his determination can be performed by employing an error correcting code ... to check for a valid codeword, or by checking ... to determine whether the initial page is programmed or unprogrammed. In addition[,] or alternatively, this determination can involve checking for a boot data indicator or a non-data indicator in the metadata

(*Id.*, 8:36-42.) If the first page examined does not contain boot code, in step 410 the device reads the next page from the NVM, and returns to step 408 to determine if it contains the first page of boot code. (*Id.*, 8:43-49.) This continues until the first page is found or the algorithm determines that it cannot be retrieved. (*Id.*, 8:53-64, 9:3-7.)

57. Once the device finds a page of boot data, Post's electronic device can use the process shown in Figure 5 for identifying later pages of boot data. The device retrieved pages from the NVM in sequence. Post's algorithm could use several methods to determine if a page of NVM data contained boot data. It could apply an error correcting code (ECC) to the data to determine if it contained valid data, check to determine if the page had been programmed or not, examine the boot data indicator metadata in the page, and examine the prior page pointer to determine if it pointed to the last previous boot data page identified or not. (*Id.*, 9:25-38.)

58. If the page was determined to not contain boot data, the control circuitry examined the next page pointer from the prior boot data page to confirm that it did not point to the current page. (*Id.*, 9:63-10:7.) If a retrieved page was determined not to contain boot data, it was not kept and was overwritten by the next page retrieved from the NVM. (*Id.*, 10:3-8.) By only keeping those pages determined to contain boot data, Post's system constructed a contiguous sequence of boot data pages in the memory 114, even if the boot data was stored in a discontinuous and

unpredictable manner in the NVM. This is illustrated in Post’s Figure 2, which shows how a sequence of boot data loaded only in “lower” NVM pages is copied to a continuous sequence of pages in the memory. (*Id.*, 5:39-52.)

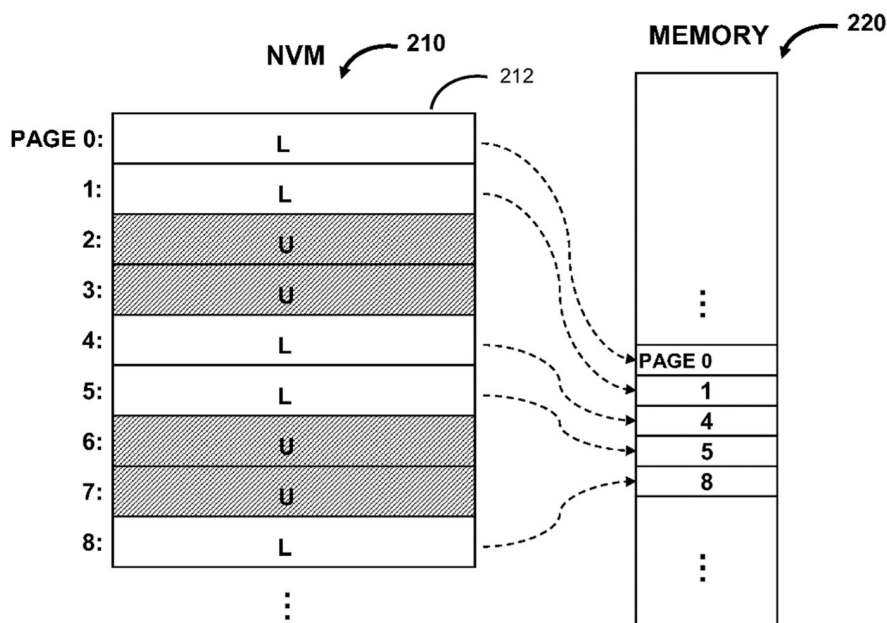


FIG. 2

59. In another boot data retrieval method described by Post, the next page pointers may be used to avoid the need to retrieve and examine each page from the NVM. (*Id.*, 1:62-64 (“Using these pointers, the electronic device can load the boot data and verify that pages expected to have boot data actually contain boot data (and vice versa).”), 10:48-49 (“pages may be selectively read such that some or all of the non-data pages are skipped over”).)

60. If the boot data is corrupted and cannot be corrected using the ECC codes (*id.*, 6:60-65), or the next page pointers or back page pointers are not consistent

with the addresses in the NVM where boot data pages are found (*id.*, 10:8-13), then Post's control circuitry may determine that it has failed to retrieve correct boot data. In that case Post's system discards the downloaded boot data and may seek to find an alternative "mirrored" copy in the NVM. (*Id.*, 10:14-19.) If none is available, the system may signal a boot failure. (*Id.*, 10:24-31.) In that case, "the electronic device may stop the bootup process and may display an error message to the user." (*Id.*, 10:30-32; *see also, id.*, 8:65-66 (error message if system cannot find a first page).) The user may be able to intervene to restore or update the boot data stored in the NVM using the process shown in Post's Figure 6. (*Id.*, 10:60-11:3.)

B. Kao

61. Kao described an embedded system 100, illustrated in Figure 1, that includes a host 102, a DRAM memory device 110 and a NAND flash memory device 130. (INTEL-1004, 3:45-51.) Like the Post patent, Kao's patent was concerned with the process of loading boot code for the embedded system from an external NVM. (*Id.*, 2:65-3:3.) An important teaching in Kao's patent was that a DMA engine should be used to transfer boot code from the NVM to a DRAM memory 110 that is accessible to the host. (*Id.*, 4:20-24 ("[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 and load controller codes 134 from the controller code images 234."); 11:17-19 ("each DMA transfer may occur between the NAND flash

memory 130 and the SDRAM array 108 without storing code in separate, special purpose buffers”).) As Kao described, using a DMA engine to carry out boot code transfers allows the boot code to be loaded “without interaction by the host.” (*Id.*, 3:18-21.) Utilizing the DMA engine to load boot code from the NVM offloads the processing workload necessary to boot the embedded system from the processor and “reduc[es] the complexity (and therefore the overall cost) of the host processor and the embedded system.” (*Id.*, 3:3-10.) Kao described that use of a DMA engine to carry out “automatic transfer of the boot code 132 from the NAND flash memory 130 ... without external control or input, e.g., by the host 102 or any external controller circuitry” was “known to those skilled in the art.” (*Id.*, 4:66-5:4.)

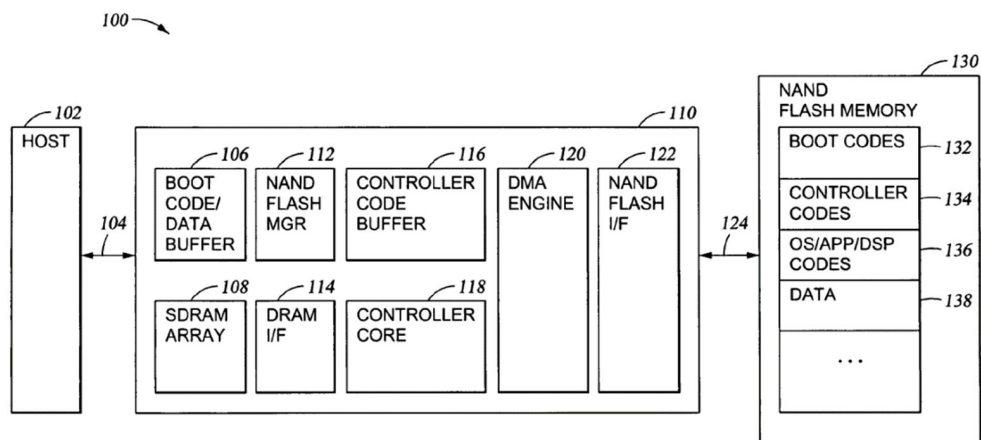


Fig. 1

C. Wilson

62. Like Post and Kao, Wilson was also concerned with the process of retrieving and using boot data as part of the boot up sequence for an embedded

electronic device, such as a mobile phone. (INTEL-1013, 1:55-2:8, 3:35-37.)

Wilson described a system, illustrated in Figure 1, that included a microcontroller 100 with a processor 101, a memory 103, a register file 102, a register load assist (RLA) engine 104 and an external non volatile memory 111 that stores boot data. (*Id.*, 2:53-67, 3:29-31 (“An external non-volatile memory, such as external ROM 111, can also be provided to store boot data.”).) Wilson described that boot-up of an electronic device generally requires device initialization data. “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.” (*Id.*, 1:25-28.) In Wilson’s system the device microcontroller includes a register file and “[t]he 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.” (*Id.*, 2:54-59; *see also, id.*, 4:59-61.) A person of ordinary skill in the art would be familiar with this concept of customizing the behavior of a processing device by setting control and configuration registers to specify properties of the system and how the microprocessor was to behave.

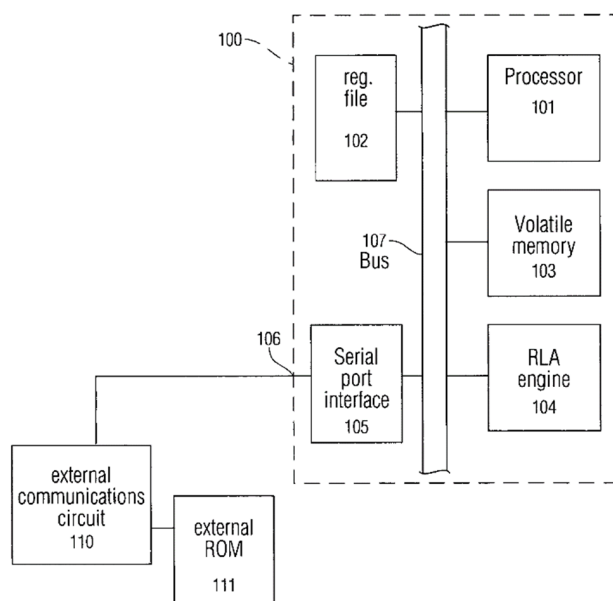


Fig. 1

63. In Wilson’s system, the initialization values for the microcontroller register file are stored in the boot data on the NVM in compressed form. (*Id.*, 5:47-6:2.) Wilson’s system first transferred the boot data from the NVM to the microcontroller memory. (*Id.*, 1:61-67, 5:57-6:2.) The RLA engine then decompressed the register initialization parts of the boot data, and stored these in the register file locations where they belonged. (*Id.*, 6:27-29.) Wilson acknowledged that the boot data in the NVM might include not only the register initialization values, but also other data to be stored in memory and instructions for the processor. (*Id.*, 6:42-44 (“[T]he compressed boot data can include other than initial register values. For example, it can include initial memory values, or initial instructions.”).)

This was consistent with the systems described in Post, where the boot data contains code that is to be executed by the control circuitry processor.

D. Overview of the Combination

64. The combination presented in this petition is based on Post's electronic device, as illustrated in Post's Figure 1. Post's device employs a boot data storage scheme illustrated in Post's Figure 3 (INTEL-1005, 6:19-8:6), methods for searching for and identifying boot data illustrated in Figures 4 and 5 (*id.*, 8:15-10:59), as well as a method for selectively reading NVM pages to skip over non-data pages. (*Id.*, 10:48-49.) Post's system searches for and identifies the particular pages in a non-volatile memory that contain boot data and copies them to the electronic device's RAM memory, as illustrated in Post's Figure 2. (*Id.*, 5:9-6:18.)

65. Kao describes the use of a DMA controller to transfer boot data from a non-volatile memory to a RAM memory in an embedded system. (INTEL-1004, 2:65-3:21, 4:20-30, 5:5-12, 11:16-22.) It would have been obvious to a POSITA at the time of Kao's publication and filing to use DMA operations in Post's electronic device for use in transferring boot data from the NVM to the memory.

66. Wilson teaches that boot data for microcontrollers in an embedded system like Post's normally contains register initialization data that must be used to set the initial configuration values in registers in a register file within the system during the boot process for the microcontroller before it begins ordinary operation.

(INTEL-1013, 1:24-33, 2:51-63.) Wilson uses a register load assist (RLA) engine to initialize the values of the microcontroller's register file based on initialization data stored in the system's boot data. It would have been obvious to a person of skill, at the time of Post's patent, that the boot data stored in Post's NVM memory should include system initialization values, like those described in Wilson. It would also have been obvious that Post's control circuitry should incorporate elements to perform the functions of Wilson's RLA engine, so that the control circuitry can decompress initialization values from the boot data retrieved from Post's NVM and store those values in a register file to configure Post's electronic device before normal operation of the processor in the control circuitry element.

E. Independent Claim 1

67. It is my opinion that claim 1 of the '084 Patent is rendered obvious by the combination of Post, Kao, and Wilson.

1. Claim 1 Preamble

A computer-implemented method, comprising:

68. Post disclosed *a computer-implemented method*. Post disclosed an “electronic device 100” such as “a desktop computer, a laptop computer, and any other suitable type of electronic device or system.” (INTEL-1005, 2:60-67.) The electronic device included a “system-on-chip 110” (SoC 110), which itself 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, 3:1-2 and 3:33-38.)

69. Post also disclosed various methods performed by the electronic device and its components (e.g., the control circuitry). For example, Post disclosed a “process for initiating bootup of an electronic device,” a “process for reading and processing boot data from a non-volatile memory,” and a “process for storing boot data in a non-volatile memory.” (INTEL-1005, 2:48-56, and Figs. 4-6.) And Post additionally disclosed that “[t]he steps of these illustrative processes may be performed by any suitable component or combination of components in an electronic device, such as by control circuitry 112 of FIG. 1. Where appropriate, these components may operate under the control of mask ROM code (e.g., from boot ROM 116) or from an operating system and NVM driver.” (INTEL-1005, 8:9-15.)

70. I discuss Post’s disclosed processes in further detail in the sections below addressing each claim element. A person of skill would have understood that control circuitry that includes one or more processors that operate under the control of code were using a *computer-implemented method*.

2. Claim 1, Limitation a

releasing components of an embedded system from reset;

71. Post disclosed that the “[e]lectronic device 100 can include system-on-a-chip (‘SoC’) 110 and non-volatile memory (‘NVM’) 120.” (INTEL-1005, 3:1-2.)

The SoC 110 included various components, including “control circuitry 112, memory 114, boot ROM 116, and NVM interface 118,” and “[c]ontrol circuitry 112 can include any suitable components, circuitry, or logic, such as one or more processors.” (INTEL-1005, 3:33-34 and 3:36-38.) The SoC 110 and NVM 120, which made up the electronic device 100, were collectively an *embedded system*.

72. This is consistent with the ‘084 patent, which defined the term “embedded system,” as including one or more processors. It stated that “embedded system will be used to refer to a single CPU or a multiprocessor system in this paper.” (INTEL-1001, 1:28-30.) Post’s electronic device 100 included one or more processors, and constituted an *embedded system* per the ‘084 patent. Since

73. Post’s examples of the types of devices represented by the electronic device 100 further illustrated that the device could be an *embedded system*. Post described that, for example, the “electronic device 100 can be or can include a portable media player (e.g., an iPod™ made available by Apple Inc. of Cupertino, Calif.), a cellular telephone (e.g., an iPhone™ made available by Apple Inc.), a pocket-sized personal computer, [or] a personal digital assistance (‘PDA’).” (INTEL-1005, 2:61-66.) A person of skill would have recognized these types of devices as examples of *embedded systems*.

74. Embedded systems typically include various special-purpose controllers. Examples of such controllers include video display controllers,

input/output (I/O) controllers, and power management integrated circuits (PMICs). These special-purpose controllers can be implemented in the form of microcontrollers or other limited-function processors.

75. While embedded systems generally include multiple microcontrollers or other limited-function processors, they typically include at most a few general-purpose processors. The general-purpose processor, also referred to as the system's central processing unit (CPU), may be responsible for executing software of the embedded system, including the embedded system's operating system (OS) and applications.

76. A POSITA would understand that Post's disclosed control circuitry 112, and the one or more processors therein, controls operations of the SoC (e.g., for bootup of electronic device 100), and therefore is alike the one or more limited-function processors or controllers found in an embedded system. The POSITA would further understand that Post's electronic 100 would typically include a CPU, separate from the control circuitry 112, which is initialized during bootup of the electronic device. These different components (e.g., the control circuitry and a CPU) would be released from reset at different times. For example, the control circuitry 112 could facilitate the bootup of the CPU.

77. Post disclosed that an electronic device, such as an embedded system, may be powered up or rebooted. (INTEL-1005, 1:16-18 ("When an electronic device

is being powered up or rebooted, an operating system can be booted up from a NVM of the electronic device.”).) Post therefore taught the reset of an embedded system. Post also disclosed *releasing components of an embedded system* (e.g., control circuit 112 and NVM interface 118) *from reset*. As discussed above, the SoC 110 disclosed by Post included control circuitry 112. (INTEL-1005, 3:33-34.) The SoC 110 also included an NVM interface 118. (INTEL-1005, 3:33-34.) Both of the control circuitry 112 and NVM interface 118 were components of the SoC 110, and therefore components of the *components of the embedded system* disclosed by Post.

78. When an embedded system is initially powered, or reset later, components of the system enter a “reset state.” The reset state resets the internal circuits and state of the components to a known starting point, and prevents the components from initiating normal operation. Additionally, the components of the embedded system may be individually held in the reset state until it is appropriate for the component to begin operation. For example, components used in the boot and initialization phase of an embedded system may exit the reset state shortly after the assertion of a system reset indication, whereas other components may not exit the reset state until late in the boot process (e.g., to be initialized). Other components may not be released from reset and become operational until the operating system has taken control of the system and basic applications are running. Examples of this include peripheral components in a mobile phone, like a cellular radio modem or

GPS receiver. When a component begins operations after a system reset, it suggests that the component has exited the reset state.

79. Post disclosed that “at bootup of electronic device 100, control circuitry 112 may operate under the control of program code stored in boot ROM 116.” (INTEL-1005, 3:41-43.) The program code is “is executed by control circuitry 112 on bootup of electronic device 100.” (INTEL-1005, 3:53-55.) For the control circuitry 112 to execute the program code during bootup of the electronic device 100, the control circuitry 112 must be functioning. That is, Post disclosed that the control circuitry 112 was *released from reset* during bootup of electronic device 100.

80. Post also disclosed that under control of the program code, control circuitry 112 “beg[an] loading additional information from NVM 120 into memory 114” (INTEL-1005, 3:52-57), including, “with the aid of NVM interface 118 ... boot data from NVM 120” (INTEL-1005, 3:57-4:5). As Post disclosed that the NVM interface 118 was used by the control circuitry 112 to load boot data from NVM 120, the NVM interface 118 must have been functioning as well during bootup. Further, it would have been obvious for the NVM interface 118 to be initialized and/or prepared for operation (i.e., to have been reset). That is, Post disclosed that NVM interface 118 was also *released from reset* during bootup of electronic device 100.

81. Post further disclosed 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 ... [a]t step 406, the boot ROM data may instruct the electronic device to read an initial page from the non-volatile memory.

(INTEL-1005, 8:25-31.) In light of Post's disclosure of the role of the control circuitry and NVM interface in transferring boot data from nonvolatile memory during bootup (INTEL-1005, 3:52-4:5), a POSITA would have understood the above to further demonstrate that the components were released from reset before performing the described operations.

3. Claim 1, Limitation 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;

82. The combination of Post, Kao, and Wilson disclosed limitation 1[b]. Below, in subsection a, I address how the combination rendered obvious the POR sequencer descriptor elements of claim 1[b]. And in subsection b, I address how the combination rendered obvious the element of detecting the POR sequencer descriptor in a nonvolatile memory.

a. Power on reset [POR] sequencer descriptor

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;

83. Post described a boot sequence performed after device 100 was initially powered on. “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.) This boot sequence, which is also known as a power-on reset sequence, made use of boot data and associated pointer metadata stored in NVM 120 of the device 100. Specifically, at system bootup, control circuitry 112 of the device 100 executed program code that enabled the control circuitry to “begin loading additional information from NVM 120 into memory 114.” (*Id.*, 3:41-43, 3:52-57.) This information included “firmware, such as a first stage bootloader” that control circuitry 112 then used to “load a second stage bootloader or the kernel and operating system of electronic device 100.” (INTEL-1005, 3:57-63.) Post referred to this “firmware []or other data” collectively as “boot data.” (*Id.*, 3:63-65.) After the boot data was loaded into memory 114, “control of control circuitry 112 [w]as transferred to the operating system.” (*Id.*, 4:5-7; *see also, 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.)

84. As I explain below, Post's boot data and associated metadata, combined with Kao and Wilson, disclosed a POR sequencer used during a power-on reset sequence. Post's power-on reset sequence, using the taught boot data and metadata, enabled the retrieval of other information in the same manner as the '084 patent's POR sequencer descriptor was described as being processed and used to control the boot-up sequence. Further, it would have been obvious to combine Post's boot data and associated metadata with the identified teachings of Wilson and Kao. Each of the features of the POR sequencer, as recited in this claim limitation, is addressed in turn below.

the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system

85. As I explain below, Wilson taught *information to initialize configuration registers of the embedded system* included in Wilson's boot data stored in nonvolatile memory. Therefore Post's boot data and associated metadata, when combined with Wilson's boot data (and the configuration register information therein), disclosed *wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system*.

86. Wilson taught that microcontroller systems typically required that initialization values be loaded into device configuration registers during the system boot-up process.

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, load instructions in its instruction memory, etc.

Some microcontrollers include on board non-volatile memory, such as read-only memory (ROM), which is used in the boot up process. More specifically, the microcontroller loads initial register values, instructions etc., from the internal ROM.

(INTEL-1013, 1:24-33; see *also, id.*, 4:59-61 (“In order for microcontroller 100 to commence ordinary operation, it can be required that certain predefined values be loaded in the registers of register file 102.”).) A person of skill would be very familiar with the need to load initialization values in system device registers, such as Wilson’s register file 102, and would recognize these registers as *configuration registers*.

87. Wilson’s patent disclosed a “[m]icrocontroller 100 [that] can include a programmable processor 101.” (INTEL-1013, 2:53-54.) The microcontroller 100 additionally included a “[r]egister file 102 [that] can include various registers which are used by the processor 101 in the ordinary operation of the microcontroller.” (INTEL-1013, 2:54-56.) Figure 1 of Wilson, illustrating the microcontroller 100, processor 101, and register file 102, is reproduced below. Wilson additionally disclosed that “microcontroller [was] used to control a portable device, such as, for example a mobile phone featuring a multifunction touch screen.” (INTEL-1013,

3:35-44.) “[T]he microcontroller can be used to display images on a screen and each five-bit register of the register file 200 can represent a color value for a pixel on a screen.” (INTEL-1013, 3:56-62.) Thus, in this example, the registers in Wilson’s register file were used to configure a display.

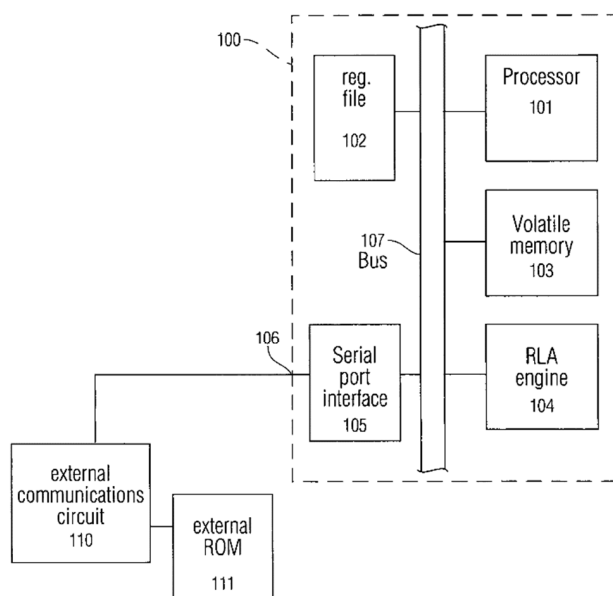


Fig. 1

88. Wilson taught that the configuration registers for system devices could be loaded with information contained in boot data retrieved from the external non-volatile memory. In Wilson’s system boot data was stored in an external non-volatile memory, ROM 111 in compressed form. (*Id.*, 5:35-39 (“The external communications circuit can ... receive boot data that has been compressed by another entity, or obtain the compressed boot data from a non-volatile memory (such as ROM 111) where it has been saved in its compressed form.”).) During boot-up,

a register load assist (RLA) engine in Wilson's system "accesses the compressed boot data from the volatile memory, decompresses it and uses it to load the various registers in register file 102." (*Id.*, 6:27-29; see also, *id.*, 2:60-63 ("the initial values can be stored in the register file as part of an initialization or boot operation. Therefore, these initial values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure.")).) Wilson, therefore, taught that a microcontroller's boot data included "initial values" to be loaded into the system registers during the boot process. And Wilson further taught that the boot data could be stored in an external non-volatile memory, such as ROM or flash. (See, e.g., Wilson, 3:29-34.)

89. The "initial values" in Wilson's boot data, loaded in the registers of Wilson's register file 102 (i.e., configuration registers) during a boot-up procedure, were *information to initialize configuration registers*. For example, in an application involving "microcontrollers for portable electronic devices" (*id.*, 2:42-44) the register initialization values might be used to set the control parameters for the central processor for the device or parameters for specific components within the device, such as display parameters. (*Id.*, 3:56-62 (configuration data can be five-bit pixel values for a screen), 9:1-6 (disclosure encompasses "loading initial register values during a boot up procedure" and "load[ing] processor instructions in a volatile memory")).) Wilson also was clear that these methods were generally applicable.

(*Id.*, 2:44-47 (“it should be understood that the present invention is generally applicable to any microcontrollers, or any similar circuits that require a boot sequence which in turn requires a significant amount of boot data”).)

90. While Post does not directly address the existence of configuration registers and the need to initialize these registers with initialization data during the boot up process, a person of skill would know from Wilson that the configuration registers and initialization data are characteristic of embedded systems. (*Id.*, 1:24-33.) It would be obvious to the person of skill to incorporate, in Post, Wilson’s teaching of storing initialization values as part of the boot data (*id.*, 6:50-54 (“Compressed boot data 500 is shown. ... [It] may include a plurality of initial register values intended for the register file.”).) Additionally, it would be obvious for Post’s control circuitry to use these initial register values to program configuration registers, or a configuration register file, in Post’s system. Wilson’s patent refers to the need to initialize registers in a microcontroller and load instructions for the microcontroller (*id.*, 1:25-28 (“allows a microcontroller to ... load initialization values in its registers, load instructions in its instruction memory, etc.”)) a person of skill would understand that the counterpart to Wilson’s microcontroller in Post’s system was the control circuitry 112, including its “components, circuitry, or logic, such as one or more processors.” (INTEL-1005, 3:36-38.) Because the registers of these components would need to be initialized

with appropriate values before the processors and other components in Post's control circuitry 112 could begin ordinary operations, those registers are *configuration registers*. A person of skill would have further understood that initialization values stored in the boot data was *information to initialize configuration registers of the embedded system*. As noted, Wilson gave the specific example of data for initializing registers to establish pixel values in a display in a small electronic device. Post specifically identified small electronic devices that featured displays, such as mobile phones (e.g., an iPhone) or pocket-sized computers, as target applications. (*Id.*, 2:61-67.) It would be obvious to a person of skill that such devices would benefit from the ability to configure system values such as the initial pixel settings for a display, as Wilson disclosed. In addition, the fact that boot code typically contained the data required to initialize embedded system components would already be well-known to a person of skill in the art. (INTEL-1016, 555 ("Boot code differs in length and functionality depending on ... the components of the actual platform that need initialization. The same (minimal) general functions are performed by boot code across the various platforms, which are basically initializing the hardware, which includes disabling interrupts, initializing buses, setting the master and slave processors in a specific state, and initializing memory.").)

91. Therefore, the combination of Post and Wilson, in which Post's boot data in nonvolatile memory 120 additionally included information to initialize

configuration registers of control circuitry 112 (including its processors, controllers, etc.) as taught by Wilson, disclosed *wherein the POR sequencer descriptor comprises information to initialize configuration registers of the embedded system.*

92. A person of skill would have found it obvious to modify Post in view of Wilson's teachings, such that Post's boot data in nonvolatile memory 120 included information to initialize configuration registers of control circuitry 112 (including its processors).

93. 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 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 person of skill would have looked to Wilson for implementation details.

94. Additionally, both Post and Wilson described use of a SoC or microcontroller to control similar devices, such as 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. (INTEL-1013, 6:49-7:3.) A person of skill would have been motivated to similarly design Post's SoC 110, including control circuitry 112, to realize the same benefits of performing specific predefined functions beneficial to the same type of device (e.g., a mobile phone with display), which would also require initialization of unorthodox configuration registers, as taught by Wilson.

95. Further, a person of skill 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 person of skill would have viewed as a further teaching, suggestion, or motivation for this combination. (INTEL-1013, 5:10-11, 5:25-26, 5:46-47, 6:42-44.) A person of skill would also have had a reasonable expectation of success in making this combination, because the use of processor registers and techniques for initializing registers at system bootup were familiar in the art. (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 ...”).)

96. A person of skill would have found additional motivation to combine Post with Wilson’s teachings regarding inclusion of register initialization values in boot data to program configuration registers or register files in the fact that Wilson described 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). It was well-known by a person of skill that “[t]he same ... general functions are performed by boot code across the various [processor architecture] platforms, which are basically initializing the hardware ... and initializing memory.” (INTEL-1016, 555). Therefore, a person of skill reading Post would have immediately recognized that Post’s SoC would have included hardware that needed to be initialized at boot, including registers that needed to be initialized. Adding register initialization information to Post’s boot data and using that data to program a register file in Post’s SoC would have been within the ability of a person of skill. 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.) Wilson’s patent provides a clear model for how to apply these methods to a microcontroller like Post’s SoC.

97. Therefore, a person of skill would 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) and Post's control circuitry transferred this information to configuration registers after it was retrieved from the NVM.

the POR sequencer descriptor comprises ... Direct Memory Access (DMA) descriptors used to fetch other POR sequencer descriptor fragments

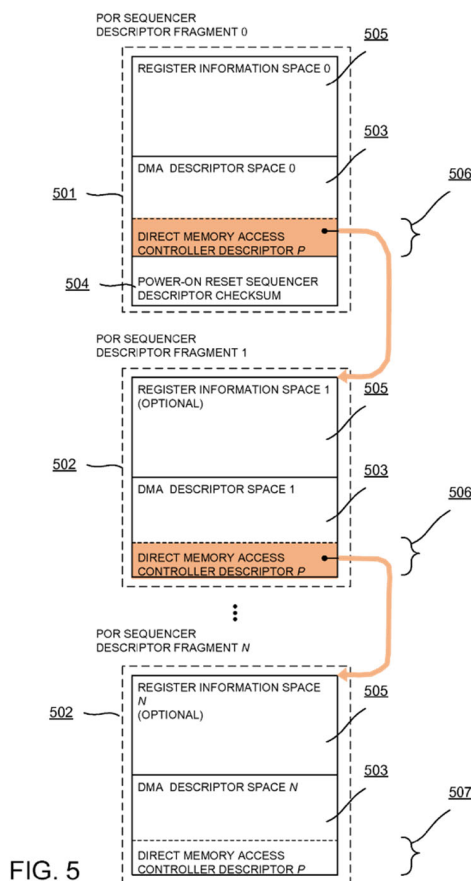
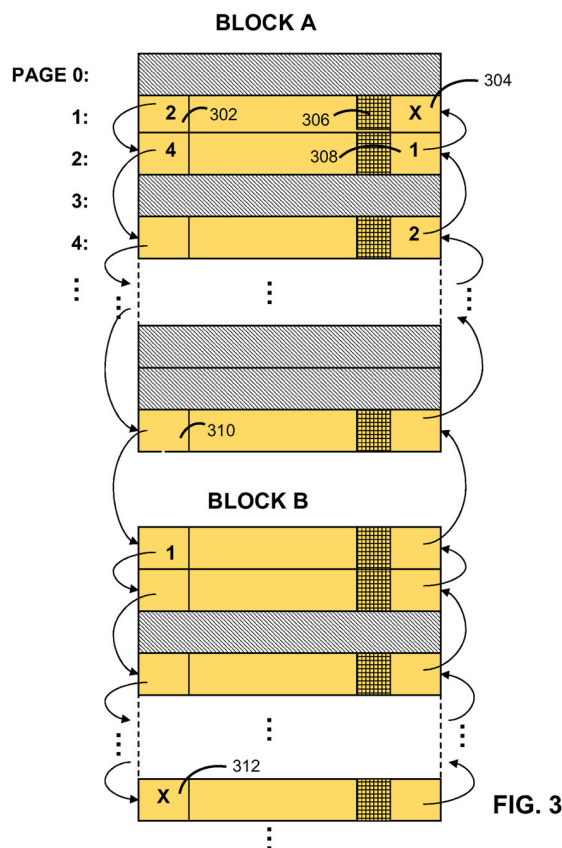
98. In Post's electronic device, the "boot data may be stored in a subset of the memory locations in the boot data storage area" and "the subset of pages [of NVM] selected to store boot data may be nonconsecutive." (INTEL-1005, 4:46-47, 5:39-40; *id.*, 5:19-23; *id.*, 1:55-64.) These boot data "fragments" (e.g., a page of boot data) were stored in a "linked-list format," as illustrated in Post's Figure 3. Post disclosed that "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.) Post described the next page pointer with reference to Figure 3.

Looking first at the linked-list formed by the next page pointers, the next page pointers in blocks A and B can each reference the next boot page in the sequence. Next page pointer 302 of page 1, for example, can provide the address of page 2. Because the pages containing boot data may be nonconsecutive, a next page pointer for a

particular page does not necessary[il]y point to the very next page in the sequence. For instance, the next page pointer for page 2 may reference page 4, thereby skipping over non-data page 3. The stored address of a next page pointer (or any other types of pointers described in this disclosure) can take on any suitable form. For example, the next page pointer can be an address referencing another page in the same block (as with next page pointer 302). Alternatively, the next page pointer can reference another page in a different block of the same or a different integrated circuit (as with next page pointer 310).

(*Id.*, 6:38-53.)

99. Figure 3 of Post, which illustrated the linked-list format of the boot data pages, is reproduced below. Post's Figure 3 has been annotated to shade the boot data pages orange. Figure 5 of the '084 patent, illustrating descriptor fragments, is also reproduced below. Post (e.g., as illustrated in Figure 3) and the '084 patent describe a similar linked list format, in which descriptors point to a next stored boot data page / fragment:



100. Post used the next page pointers to fetch each individual boot data pages (i.e., a *POR sequencer descriptor fragment*), initially stored in the NVM 120, into memory 114. Post’s Figure 5, reproduced below, illustrated a process “for loading boot data into memory” (i.e., from NVM into memory). (INTEL-1005, 9:14-15.)

101. Post disclosed that the process for loading boot data into memory began with “the electronic device [reading] a page from the nonvolatile memory” into memory and “determin[ing] whether the current page contains boot data.” (INTEL-1005, 9:19-26; *see also* 9:14-15 and 9:3-6.) These boot data pages would have been loaded into Post’s memory 114. (*See, e.g.*, INTEL-1005, 3:53-61 (“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. This additional information may include 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.”.)

102. If the page loaded from nonvolatile memory into memory was determined to contain boot data, Post’s electronic device verified “whether the [current page’s] prior page pointer point[ed] back to the expected page (i.e., the most recently read and/or processed boot data page.” (INTEL-1005, 9:33-38.) That is, Post disclosed confirming that the current page’s prior page pointer correctly pointed back to the most recent prior page of boot data. If the prior page pointer information was confirmed, Post’s electronic device maintained the boot data, from the current page, in memory. (INTEL-1005, 9:39-46.)

103. Once confirmed as a boot data page, Post’s electronic device “examine[d] the next page pointer” of the current page “to determine whether the pointer reference[d] a subsequent page of the non-volatile memory.” (INTEL-1005, 9:47-52.) If “the pointer indicate[d] that there are no more pages of boot data,” meaning the current page is the last page of boot data, the process for loading boot data from the nonvolatile memory into memory ends. (INTEL-1005, 51-54.) If,

however, the next page pointer referenced a subsequent page of nonvolatile memory, indicating the current page was not the last page of boot data, Post's electronic device "save[d] the current page's next page pointer for use in processing subsequent pages ... [and] continue[d] reading pages from the non-volatile memory to obtain the rest of the boot data." (INTEL-1005, 9:56-62.)

104. Post disclosed two methods of identifying and retrieving boot data fragments from the NVM. In the first method, Post read all of the NVM pages in sequence and retained only those identifies as boot data pages. (*Id.*, 10:43-47 ("In other embodiments, for example, the electronic device may read all of the pages in a specified boot data storage area of the non-volatile memory. The electronic device may then post-process the read pages using steps similar to those shown in processes 400 and 500.")) In the second method, Post disclosed that "pages may be selectively read such that some or all of the non-data pages are skipped over." (*Id.*, 10:48-49.) That is, the electronic device could use the next page pointers from a previous boot data page to identify the next boot data page in the NVM and retrieve it directly, skipping over the non-boot data pages.

105. A POSITA would have understood that the next page pointers were used to skip over non-boot pages when Post's second method of retrieving boot data was used. Post described that the next page pointer in nonconsecutive boot data blocks "skips over" non-data blocks to point to the next boot data block. (*Id.*, 6:44-

46.) Therefore, when Post refers to the second method of retrieving boot data and states that “pages may be selectively read such that some or all of the non-data pages are skipped over” (*id.*, 10:48-49), it is obvious that the next page pointers are the mechanism that is to be used to make this possible. Additionally, Post refers to the linked list of next page pointers as a type of “map,” (*id.*, 6:19-21 (“FIG. 3 illustrates ways in which a linked-list format may be used to store a map of which pages are currently used as boot data pages.”)), that may be used to jump directly to boot data pages and bypass bad data blocks. (*Id.*, 6:8-13 (“[E]ach particular NVM unit may have a specific set of bad blocks that are accumulated due to factory defect or through use. In these situations, the map can be used by the electronic device to identify which blocks are bad and to skip over those blocks when retrieving boot data.”))

106. Post disclosed no mechanism, other than the “‘next page pointer’ pointing to the next page that contains boot data” (INTEL-1005, 1:58-59), for performing the “skipping” function and it would have been obvious to a person of skill that the next page pointers could be used for that purpose.

107. In the electronic device taught by Post, where boot data pages are intentionally stored nonconsecutively in nonvolatile memory, it would have been inefficient to exhaustively fetch pages from an area of nonvolatile memory, only to then post-process the fetched pages to identify which contain boot data. That

approach would add to the power-on reset sequence (e.g., fetching and discarding non-boot data pages) and/or require a greater amount of storage to temporarily store the pages fetched from nonvolatile memory. In contrast, using the disclosed next page pointers to direct the fetching of pages from the nonvolatile memory would have been advantageously efficient, as it would have sped up the power-on reset sequence and/or reduced the amount of storage (e.g., volatile memory) used. Moreover, it would have been straightforward to use the next page pointers and associated metadata in such a manner - and consistent with what Post taught about the boot data using a “linked-list” to identify the next boot data page in nonvolatile memory. (*See, e.g.*, INTEL-1005, 1:55-59.) Traversing a linked list, by using the next page pointer of each list element to locate and obtain the next list element, was conventional and would have been well-known to a person of skill.

108. Post, therefore, disclosed next page pointers that served as descriptors of the next block of boot data (i.e., POR sequencer descriptors) and disclosed that they were *used to fetch* other fragments of boot data (*other POR sequencer descriptor fragments*).

109. Post did not describe a specific mechanism for transferring boot data pages from the NVM 120 to memory 114, but stated that the control circuitry could incorporate “any suitable components, circuitry, or logic.” (INTEL-1005, 3:36-38.) Kao taught “booting an embedded system,” where “boot codes and controller codes

stored in a non-volatile memory may be loaded by a DMA engine.” (INTEL-1004, 2:65-3:3.) Figure 1 from Kao, reproduced below, illustrates the DMA engine 120, as well as boot codes 132 and controller codes 134 stored in nonvolatile NAND flash memory 130:

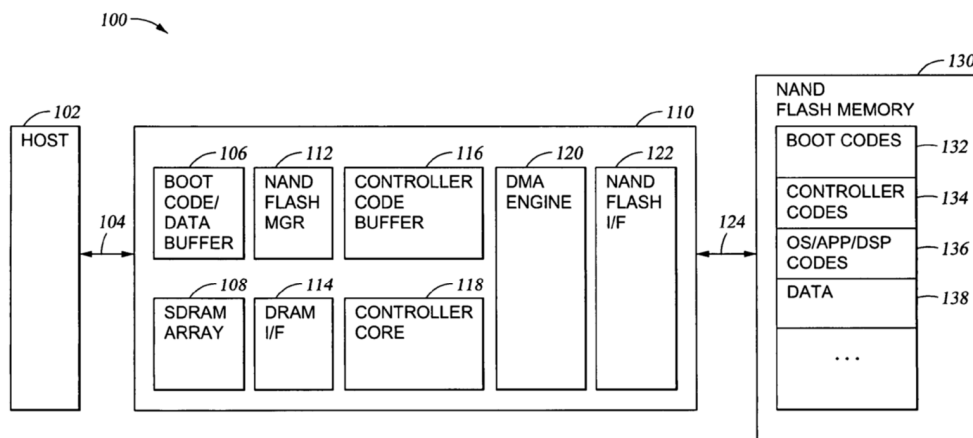


Fig. 1

110. Kao taught that its DMA engine would use data contained within one part of the boot data in the NAND flash memory to direct the loading of another part of the boot data into a DRAM device. Kao described that “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.” (INTEL-1004, 10:28-33, *see also, id.*, 5:25-32 (“The controller codes 134 may be loaded ... using instructions in the boot codes 132”).) Kao further taught that “the boot codes and controller codes may be loaded by the DMA engine into the volatile memory chip without interaction with the host.” (INTEL-1004, 3:19-21.)

111. A POSITA would have found it obvious to combine Post and Kao by including in Post's controller circuitry 112 a DMA engine for retrieving boot data from the NVM 120. In the combination where Post was modified to include Kao's teachings of a DMA engine, it would have been obvious to a person of skill that Post's next page pointer metadata would provide the data required to program the DMA engine to transfer the next page of boot data from nonvolatile memory 120 into RAM 114. Because Post teaches that each page of boot data includes a next page pointer that identifies the location of the next boot data page, this mechanism would allow for DMA retrieval of each boot data page after the first page had been detected. The next page pointer metadata would be recognized by a person of skill as a *DMA descriptor*.

112. A person of skill would have understood Kao to provide a teaching, suggestion, or motivation for making this combination. Kao teaches that using DMA is advantageous because it offloads work from the processor, reducing the processor's workload in booting an embedded system, while 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.”.)

113. A person of skill would have recognized that Kao’s teaching regarding the benefits of DMA for boot data transfer were consistent with what was already well known in the art about the benefits of DMA techniques. It was known, for instance, 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.”).) Reducing the workload of an embedded system’s processor was an omnipresent design goal in the field of embedded systems, which would have made it all the more compelling for persons of skill to make a 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).

114. A person of skill would have had a reasonable expectation of success in making this combination. Post and Kao taught similar embedded systems, with similar components within their respective embedded system (INTEL-1005, 3:33-34 (“System-on-a-chip 110 can include control circuitry 112, memory 114, boot ROM 116, and NVM interface 118.”); INTEL-1004, 3:49-51 (“[T]he system 100 may include a host 102, volatile memory (DRAM device 110), and non-volatile memory (NAND flash memory 130).”), 3:22-23 (“Embodiments of the invention are described below with respect to an embedded system including a host processor”).) And both Post and Kao taught that their respective embedded systems could be found in similar devices, such as phones. (INTEL-1005, 2:60-67; INTEL-1004, 1:15-17.) Further, 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”).)

115. A person of skill would have found further motivation to combine Post and Kao by adding a DMA engine to Post because Kao’s DMA techniques had already been used to improve one device (an embedded system including one or

more processors, INTEL-1004, 3:22-27). A person of skill 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)) 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.) Additionally, both Post and Kao addressed the same 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.").

116. Additional motivation to combine Post and Kao would have come from the fact that this combination as employed the simple substitution of a known element, in the form of Kao's DMA techniques for retrieving boot data, for another, *i.e.*, using the core circuitry 112 to transfer boot data, as taught by Post. A person of skill would have recognized 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)

POR sequencer descriptor comprises ... a system firmware

117. 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; *see also, id.*, 3:63-65 (“[T]he firmware (or other data) retrieved from NVM 120 by the mask ROM code may sometimes be referred to as ‘boot data.’”).) Post therefore disclosed *wherein the POR sequencer descriptor comprises ... a system firmware.*

b. Detecting a POR sequencer descriptor

detecting a POR sequencer in a nonvolatile memory

118. Post disclosed that its boot data and associated metadata, which comprised a POR sequencer descriptor, were stored in nonvolatile memory 120. (INTEL-1005, 3:52-61, 4:27-28, 5:9-13.) Wilson also disclosed that its boot data, which included initial register values, were stored in a nonvolatile memory. (INTEL-

1013, 2:62-63, 3:29-34.) The combination of Post, Wilson and Kao would also employ boot data that was contained *in a nonvolatile memory*.

119. Post also disclosed a process “for initiating bootup of an electronic device,” that “identif[ied] which memory location of a non-volatile memory contain[ed] the first page of boot data (e.g., bootloader data).” (INTEL-1005, 8:16-20.) Post’s method was to “read an initial page from the non-volatile memory” and “determine whether the initial page contains boot data.” (INTEL-1005, 8:29-31, 8:34-42.) If boot data was detected, “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.” (INTEL-1005, 8:45-49.) In the resulting process, Post’s electronic device read pages from nonvolatile memory 120 until it detected the first page of boot data was located. Post therefore disclosed *detecting the POR sequencer*.

4. Claim 1, Limitation c

copying the POR sequencer descriptor from the nonvolatile memory to a random access memory (RAM);

120. Post disclosed copying boot data, including the metadata contained within each page of boot data (INTEL-1005, 6:28-37), from a nonvolatile memory to a random access memory (“RAM”). Post described that control circuitry 112 copied boot data from nonvolatile memory 120 into memory 114. (*Id.*, 3:55-65 (“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 ... and the firmware (or other data) retrieved ... may sometimes be referred to as ‘boot data’”), 5:13-23 (“FIG. 2 provides a graphical view of the operation performed to load boot data from block 212 to memory 220 during bootup”).) Post identified memory 114 as 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 described that “memory 114 may act as the main memory for any processors implemented as part of control circuitry 112.” (*Id.*, 3:49-51.) A person of skill would recognize that individual addresses in the main memory should be accessible to the processor. This enables the processor to access data in the most efficient manner.

121. In the combination of Post and Kao, when a DMA engine was used to effect a transfer of pages of boot data from Post’s NVM 120 to the RAM memory 114, the various bits of data contained in each boot code page would be copied together. For example, the transfer of page 1 of boot data illustrated in Post’s Figure 3 would move each of next page pointer 302, boot code data (unnumbered), boot data page indicator 306, and prior page pointer 304 into the RAM memory 114. Transfer of page 2 would move the next page pointer from that page, the boot code data for page 2, the data page indicator for page 2 and prior page pointer 308, into the RAM memory 114.



- 74 -

100 (FIG. 1) has finished loading the boot data from memory 114.”)) As shown in Figure 2, the boot data may be stored in the NVM in non-contiguous pages (depicted in white in Figure 2), but Post’s device copied these pages so that they ended up in contiguous pages in the RAM memory.

[T]he electronic device can be configured such that ultimately only information from the nonconsecutive boot data pages of NVM 210 end up in consecutive pages of memory 220. This way, once control of the electronic device is transferred to memory 220, the boot data is available in a form suitable for execution by control circuitry.

(*Id.*, 5:45-50.)

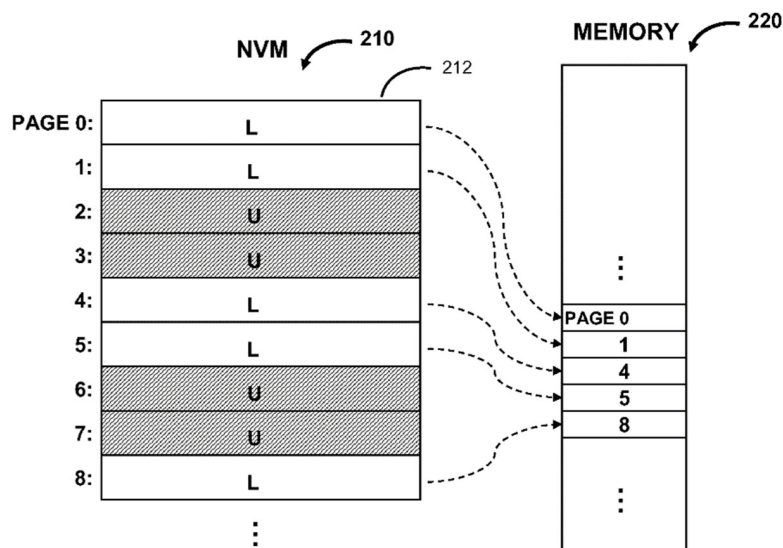


FIG. 2

123. Because Figure 2 shows that the pages of boot data are copied from NVM 120 (210 in Figure 2, *see, id.*, 5:19-20) to RAM memory 114 (220 in Figure 2, *see, id.*, 5:20-21), it would be evident to a reader of ordinary skill that the metadata

contained in each page, including the next page pointer, boot data page indicator and the prior page pointer, would also be copied to the RAM memory 114. (*Id.*, 6:28-31 (“[E]ach of the boot data pages can store both data (e.g., boot data) and metadata. The metadata for page 1, for example, can include a next page pointer 302 and a prior page pointer 304.”), 6:35-37 (“The metadata in each boot data page can further include a boot data page indicator, such as boot data page indicator 306.”).)

124. Because Post referred to processing and transferring “pages” of boot data, (*id.*, 5:43-52 (“the electronic device can be configured such that ultimately only information from the nonconsecutive boot data *pages* of NVM 210 end up in consecutive *pages* of memory 220. ... [T]he loading of boot data can be accomplished by incrementally processing and validating *pages* as the *pages* are being read.” (emphasis added))), a person of skill would naturally understand that Post’s system copied complete pages of boot data, including the metadata contained in those pages, from NVM to memory. In addition, a person of skill would recognize that one of the reasons for moving boot data from NVM to RAM memory was to allow it to be accessed more quickly than it could be accessed from the NVM. This rationale would apply to transfer of Post’s boot page metadata, including the next page pointers, no less than it did to transfer of the other parts of the boot page.

125. As discussed with regard to element b, above (page 48), the POR sequencer descriptor in Post’s system included the boot data and the metadata

originally contained in the various pages of the NVM that was subsequently copied to the RAM memory 114.

5. Claim 1, Limitation d

verifying an integrity of the POR sequencer descriptor; and

126. Post's electronic device verified the integrity of each page of boot data from the NVM as it was transferred. There were multiple checks conducted by Post's device to verify the boot data and metadata contained in Post's NVM pages, which, as discussed above (see discussion of element b, above at page 48), constituted a POR sequencer descriptor. Each of the checks discussed below functions independently to *verify the integrity* of Post's boot data, which operates as *the POR sequencer descriptor* in Post's system.

127. Post's method performed verification when the electronic device was looking for the first page of boot data. It examined the prior page pointer to determine if that pointer indicated that this was the first page. This verification process was represented in step 412 of Figure 4.

At step 412, the electronic device can examine the prior page pointer from the metadata of the current page. In particular, at step 414, the electronic device can determine whether the prior page pointer indicates that the current page is the first page of boot data. If the prior page pointer points to a previous page instead of showing that the current page is the first page, this is an indication that the electronic device was previously unsuccessful at reading the first boot data page. That is, the current page may be a

second or subsequent page used to store boot data, and the first page of boot data may have been lost due to error.

(*Id.*, 8:52-62.) The electronic device also used the prior page pointer when it searched for subsequent pages of NVM data. When it encountered a subsequent NVM page that appeared to contain boot data, it verified that the next page pointers of the prior page identified this subsequent page as the next page in the sequence of boot pages. This process was depicted in step 508 of Figure 5.

If the electronic device determines that the current page contains boot data, the electronic device can verify the prior page pointer of the current page data. That is, at step 508, the electronic device can confirm whether the prior page pointer points back to the expected page (i.e., the most recently read and/or processed boot data page).

If, at step 508, the prior page pointer can be verified, the electronic device can maintain the boot data from the current page in memory. Thus, by performing steps 506 and 508 before deciding to maintain the boot data, the electronic device can perform several checks on the validity and accuracy of the boot data stored in the non-volatile memory. This may be beneficial, because accurate boot data may be critical to the operation of the electronic device.

(*Id.*, 9:33-46.)

128. Post's electronic device also verified the POR sequencer descriptor from the NVM using the boot data page indicator 306. (*Id.*, 7:63-67 ("These boot data indicators can provide another way in which the electronic device can verify that a page is a boot data page. In some embodiments, also for verification purposes,

the non-data pages may be programmed with non-data indicators to show that these pages do not contain boot data.”), 8:34-42 (“Then, at step 408, the electronic device may determine whether the current page contains boot data. ... [T]his determination can involve checking for a boot data indicator or a non data indicator in the metadata of the current initial page.”), 9:25-33 (“Then, at step 506, the electronic device may determine whether the initial page contains boot data. ... [T]his determination can involve checking for a boot data indicator or a non data indicator.”))

129. Post’s electronic device also used error correction codes and page program status data to verify the integrity of boot data. (*Id.*, 6:57-61 (“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. If the error correction results in uncorrectable errors, the boot data for page 2 may be unobtainable ...”), 8:34-40 (“Then, at step 408, the electronic device may determine whether the initial page contains boot data. In some embodiments, this determination can be performed by employing an error correcting code on the initial page data to check for a valid codeword, or by checking a program status of the initial page to determine whether the initial page is programmed or unprogrammed.”), 9:25-31 (“Then, at step 506, the electronic device may determine whether the current page contains boot data. In some embodiments, this determination can be performed by employing an error correcting code on the current page data to check for a valid codeword, or by

checking a program status of the initial page to determine whether the initial page is programmed or unprogrammed.”).)

130. Post also used the prior page pointer and next page pointer metadata to verify the integrity of the boot data. When Post retrieved pages from the NVM that were identified as not containing boot data, these pages were dropped or overwritten in the RAM memory. For example, when searching for the first page of boot data, Post’s algorithm maintained the transferred page of boot data only when the page was determined to be the first page of boot data. (*Id.*, 9:3-6 (“if the electronic device determines that the current page is the first page of boot data, the electronic device can maintain the boot data in memory at step 420”).) Otherwise, the electronic device read the next page. (*Id.*, 8:43-46 (“If, at step 408, the electronic device determines that the initial page does not contain boot data, process 400 can continue to step 410. At step 410, the electronic device may read the next page of the non-volatile memory,”).) And, when searching for subsequent pages, Post’s algorithm similarly kept the copied NVM pages only when the prior page pointer confirmed that they contained boot data. (*Id.*, 9:36-40 (“at step 508, the electronic device can confirm whether the prior page pointer points back to the expected page (i.e., the most recently read and/or processed boot data page). [¶] If, at step 508, the prior page pointer can be verified, the electronic device can maintain the boot data from the current page in memory.”).) If the current page was identified as not containing

boot data, the algorithm checked to determine if the next page pointer of the last identified boot page suggested, inconsistently, that the current page was the next boot page.

[I]f the electronic device determines that the current page does not contain boot data, the current page may be a non-data page. Thus, at 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, however, at steps 517 or 508, 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.

(*Id.*, 9:63-10:13.) Using the previous and next page pointer to assist in the identification of boot data and to accept or reject NVM pages ensures the integrity of the boot data that is retrieved.

131. Once each of the boot code pages was retrieved from the NVM, verified using next page pointers, boot data page indicators and error correction codes, the entire boot code would be assembled in the RAM memory, as shown in Post's Figure 2. Post's use of next page pointers, boot data page indicators and error correction codes, each served to assist in verification of boot data integrity. These checks conducted by Post served to ensure that the boot data had been correctly identified and copied from the NVM. NVM pages that did not contain boot data were dropped

and/or overwritten, and error correction was applied to the data copied to the RAM memory to ensure that the data appearing in the RAM memory was an accurate copy of the boot data originally stored to the NVM.

6. Claim 1, Limitation e

detecting any error in the POR sequencer descriptor;

132. Post's electronic device used error correction codes to *detect any error* in the boot data, which operates as *the POR sequencer descriptor* in Post's system. When it searched for the first page of boot data using the algorithm shown in Figure 4 (*id.*, 8:18-20 ("Process 400 [depicted in Figure 4] may be performed to identify which memory location of a non-volatile memory contains the first page of boot data")), Post's system performed error checking on each page of data in the NVM that was a candidate for first page of boot data to determine if it contained valid data.

[A]t step 408, the electronic device may determine whether the initial page contains boot data. In some embodiments, this determination can be performed by employing an error correcting code on the initial page data to check for a valid codeword...

(*Id.*, 8:34-38.) When Post's electronic device looked for subsequent boot code pages in the NVM, it also applied error correction codes to those NVM pages.

[A]t step 506, the electronic device may determine whether the current page contains boot data. In some embodiments, this determination can be performed by employing an error correcting code on the current page data to check for a valid codeword...

(*Id.*, 9:25-29.)

133. Post described that if the previous page pointer in the prior page of boot data identified a particular NVM page as containing boot data, but the data in that page contained uncorrectable errors that could not be corrected using the error correcting codes for that page, then the electronic device would either need to find an alternative copy of the boot data or signal a boot failure.

For example, since next page pointer 302 points to page 2 [in Figure 3] page 2 is expected to contain boot data. 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. 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.

(*Id.*, 6:56-65.) This demonstrates that detecting, and then correcting, any errors in a valid page of boot data was a necessary part of Post's process.

134. Kao also taught that error correction codes (ECC) should be applied to boot data read from an NVM. "The ECC utilities may be used to ensure that data received from the NAND flash memory 130 has been correctly transmitted and possibly correct errors in the transmission, if any." (INTEL-1004, 5:49-52.) It would have been obvious to a person of skill that, in the combined system in which a DMA engine was used to retrieve boot data from an NVM, error correction codes

should be applied to the data. In addition to the advantage identified in Post, where ECC codes allow the electronic device to detect pages that do not contain valid data, Kao identifies the additional benefit of protecting the boot data against errors that may arise during transmission from the NVM.

7. Claim 1, Limitation f

a. First part

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

135. I have already described in the discussion of limitation d, above (page 77), how Post's computer-implemented method allows the embedded electronic device *to verify the integrity* of the boot data that operates as *the POR sequencer descriptor* in Post's system. The error detection performed by the combination of Post and Kao and discussed in connection with element e, above (page 82), also serves to verify that the boot data copied from the NVM and assembled in the RAM memory, as shown in Figure 2, has integrity.

136. The boot data copied to the RAM memory comprises *user data and program code that is used in a boot process*. Post describes that the boot data in the NVM includes firmware comprising a first stage boot loader that can be executed to load a second stage boot loader that either comprises, or initiates, the kernel

functions and an operating system specific to the electronic device. Performance of these functions is identified by Post as part of the “bootup procedure” for the device.

When an electronic device is being powered up or rebooted, an operating system can be booted up from a NVM of the electronic device. The bootup of the operating system can proceed in multiple stages. For example, the electronic device can be configured to first execute code from a mask read-only memory (“ROM”). The code can include instructions to load a bootloader, such as a first stage bootloader, from a flash memory to a volatile memory. The task of booting up a second stage bootloader and/or the kernel and operating system of the electronic device may then be transferred from the mask ROM code to the first stage bootloader. The successful completion of this bootup procedure is typically critical for the electronic device to function properly.

(INTEL-1005, 1:16-28.)

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 boot loader, 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. For clarity, the program code stored in boot ROM 116 may sometimes be referred to as “mask ROM code,” and the 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.)

137. A person of ordinary skill would understand the first and second stage boot loaders, and the operating system for the electronic device, to be *user data*.

Post made this explicit, as it described that these code elements could be revised by the user by in-field updates to the NVM data.

Because, unlike a mask ROM, the non-volatile memory is reprogrammable by an electronic device's operating system and NVM interface, the boot data may be changed or updated without the inconvenience of switching boot ROMs. Process 600 may be performed during a firmware update as part of a restore process. The restore process may be performed at a factory during manufacture or repair of the electronic device, ***or may be initiated by the end user to update the electronic device.***

(*Id.*, 10:62-11:3 (emphasis added); *see also, id.*, 4:45-47 (“During a firmware update or other restore procedure on electronic device 100, boot data may be stored in a subset of the memory locations in the boot data storage area.”).) The process by which a user may update the boot data contained in the NVM is illustrated in Post’s Figure 6. (*Id.*, 10:60-62.)

138. A person of skill would also understand the boot data to comprise *program code*. Post describes that the boot data includes code, which it generally refers to as a first stage bootloader, that can be executed by a processor in Post’s control logic. (*Id.*, 3:36-41 (“Control circuitry 112 can include any suitable components, circuitry, or logic, such as one or more processors. Control circuitry 112 may operate under the control of a program, such as an application, operating system, a NVM driver, or a bootloader loaded in memory 114.”), 3:55-61 (“The program code [in the boot ROM] 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 boot loader, 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.”.)

139. Because the boot data, comprised *program code and user data*, the various procedures used to verify that the boot data was correct can be appropriately characterized as *verifying the integrity of program code and user data*.

140. Post’s procedures for boot up also provided *a resilient boot process*. They allowed the system to store boot data in whatever locations in the NVM were most advantageous for ensuring that the data is robustly stored.

During a firmware update or other restore procedure on electronic device 100, boot data may be stored in a subset of the memory locations in the boot data storage area. The pages in the subset may be chosen to increase storage reliability of the boot data, and may be selected using any suitable technique. For example, if NVM 120 is configured using MLC, boot data may be stored only in the lower pages of the boot data storage area. Because the lower pages of NVM 120 may have lower error rates than the upper pages, this technique may provide increased storage reliability. In some embodiments, pages that are known to be less reliable in a particular NVM may be excluded from the subset. For example, the last one or more pages in a block may be excluded. These and other approaches may be used individually or in combination to select a subset of pages for storing boot data.

(*Id.*, 4:45-59; *see also, id.*, 1:43-51, 5:26-29 (“boot data is shown as being stored only in the lower pages (e.g., pages 0, 1, 4, 5 and N) of block 212, which can improve the reliability of the system”).) Storing data in NVM locations that are less prone to errors makes the data more likely to be valid, making the boot process more likely to be successful and, therefore, more *resilient*.

141. Post’s use of error correcting code also allowed errors in the boot data that may be introduced by the processes of storing and reading the data from the NVM to be corrected. (*Id.*, 6:57-61 (“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. If the error correction results in uncorrectable errors, the boot data for page 2 may be unobtainable”), 8:35-38 (“this determination can be performed by employing an error correcting code on the initial page data to check for a valid codeword”), 9:27-29 (“this determination can be performed by employing an error correcting code on the current page data to check for a valid codeword”).) The ability to correct errors also makes the boot process more *resilient*.

142. Finally, Post’s procedures allowed for use of a second copy of the boot data if the data in the first copy cannot be corrected. (*Id.*, 1:65-67 (possibility that “a redundant copy of the boot data is stored elsewhere in the boot data storage area”), 4:38-41 (“For example, a copy of the boot data may be stored in each of several integrated circuits (e.g., one copy in each block of a superblock), or in each of several

blocks of the same integrated circuit.”), 6:63-64, 7:48-50 (upon error correction failure, “[t]he electronic device may therefore signal a boot failure or may try loading a redundant copy of the boot data from another location in the NVM”), 10:24-28 (“Returning to step 520, the NVM interface may determine that mirrored content is not available, such as if only one copy of the boot data is stored in the NVM or if the electronic device has already unsuccessfully attempted to load all other copies of the boot data.”).) Post’s procedures also permitted a user to update or restore the boot data, and therefore the boot procedure, as needed, even after the electronic device is in the field. (*Id.*, 10:62-11:3 (“Because, unlike a mask ROM, the non-volatile memory is reprogrammable by an electronic device's operating system and NVM interface, the boot data may be changed or updated without the inconvenience of switching boot ROMs. Process 600 may be performed during a firmware update as part of a restore process. The restore process may be performed at a factory during manufacture or repair of the electronic device, or may be initiated by the end user to update the electronic device.”); *see also, id.*, 11:4-67 (describing the boot code update / restoration process in detail).)

143. Collectively, these procedures for allowing data to be stored in the most error-resistant locations in the NVM, to identify and correct errors, to use a backup copy of the boot data, and to revise and restore the boot data, produced a *resilient boot up procedure*.

b. Second part

and wherein the computer-implemented method reduces an amount of the nonvolatile memory used in the boot process.

144. The system that combines the elements of Post's electronic device with the DMA process disclosed in Kao and the register file initialization process disclosed in Wilson implements a reduction in the amount of nonvolatile memory used in the boot process in several ways.

145. The first way that the combined system reduced an amount of nonvolatile memory used in the boot process was in its use of compression to reduce the size of the stored boot data. Wilson described that boot data should be stored in an external NVM in compressed form. (INTEL-1013, 1:63-65 ("The boot data can be originally stored in compressed form in the external non-volatile memory or it can be compressed by the external circuit."), *see also, id.*, 5:38-42 ("Since the [b]oot data is usually defined at the time of manufacture of a device and does not change during the life of the device, it may be suitable to save it in its compressed form in a static memory.")) In addition, when the initialization data that was to be written to the register file was smaller than the width of NVM containing the boot data multiple registers values it could be stored in a single, shared, memory line. (*Id.*, 6:54-67.) In Wilson's system, the register load assist (RLA) decompressed the boot data, separated the stored register initialization values and stored them in the correct locations in the register file. (*Id.*, 6:27-29, 7:4-16.) Compressing the boot data and

packing the initialization values to make efficient use of the NVM storage had the effect of reducing an amount of the NVM memory that was required to store boot data. The combined system in which Post's control circuitry performed the decompression and register transfer functions disclosed in Wilson would result in more efficient NVM utilization in the same way.

146. The second way that the combined system reduced an amount of nonvolatile memory used in the boot process was in its use of a boot ROM to store an initial boot code header. Post disclosed a multi-stage boot process in which the control circuitry initially executed code in the boot ROM that allowed retrieval of a first stage bootloader from an external NVM. The first stage bootloader then was executed to carry out retrieval of a second stage of boot code, including perhaps the operating system.

[T]he electronic device can be configured to first execute code from a mask read-only memory ("ROM"). The code can include instructions to load a bootloader, such as a first stage bootloader, from a flash memory to a volatile memory. The task of booting up a second stage bootloader and/or the kernel and operating system of the electronic device may then be transferred from the mask ROM code to the first stage bootloader.

(INTEL-1005., 1:19-26.) This multi-stage process minimized the complexity and size of code that would otherwise need to be stored in the mask ROM (aka "boot

ROM”; *see, id.*, 1:43, 3:61-63), thereby reducing an amount of nonvolatile memory used in the boot ROM for the boot process.

147. The third way that the combined system reduced an amount of nonvolatile memory was in the possible use of multiple devices to store the boot data. Post disclosed that the nonvolatile memory that stores the boot data “can include multiple integrated circuits, where each integrated circuit may have multiple blocks.” (INTEL-1005, 3:12-14, *see also, id.*, 4:33-34 (“The boot data storage area can include any suitable number of integrated circuits, blocks, or pages.”).) Because the boot data pages in Post’s system included next page pointers that identified where the next page of boot data is stored, (*id.*, 6:38-53) the boot data need not be stored in a continuous sequence in a single device. User multiple devices and pointers to chain the boot data across multiple devices allows the user to reduce an amount of space used to store boot data in any particular storage device.

148. This was very similar to the way this claim limitation is embodied in the ’084 patent. The patent describes that “[w]hen the POR sequencer descriptor [i.e., the boot data] doesn’t fit in the nonvolatile memory of the system, the descriptor is divided into fragments and distributed across the nonvolatile memory and storage devices attached to the system.” (INTEL-1001, 6:34-37.) The total amount of storage required does not change, but the size of the nonvolatile memory necessary to contain the first fragment of boot data, is greatly reduced.

149. Finally, Post's method of storing data might also reduce the amount of nonvolatile memory used in the boot process if Post's teaching to use error correcting codes to protect boot data (INTEL-1005, 9:25-29), and to store boot data in the pages in an NVM that are most resistant to errors (*id.*, 4:45-59, 5:26-35, 5:66-6:13), make the boot data sufficiently resistant to error that it is not necessary to store multiple copies of the boot data in the NVM. Both Post and the '084 patent describe that multiple copies of the POR sequencer descriptor data can be stored in NVM. (INTEL-1001, 7:9-12, INTEL-1005, 1:64-67.) But Post also contemplates the possibility that only one copy of the POR sequencer descriptor boot data is stored in the NVM (*id.*, 10:24-26) and provides a mechanism that allows users to reload or repair the boot data if that copy cannot be retrieved. (*Id.*, 10:28-35.) Because Post's method provides more robust methods for storing POR sequencer descriptor boot data, and a reliable way of restoring it if it is corrupted, a person of skill would recognize that it is possible to store only a single copy. Avoiding the need to store multiple copies of the POR sequencer descriptor would significantly *reduce the amount of nonvolatile memory used in the boot process.*

F. Dependent Claim 5

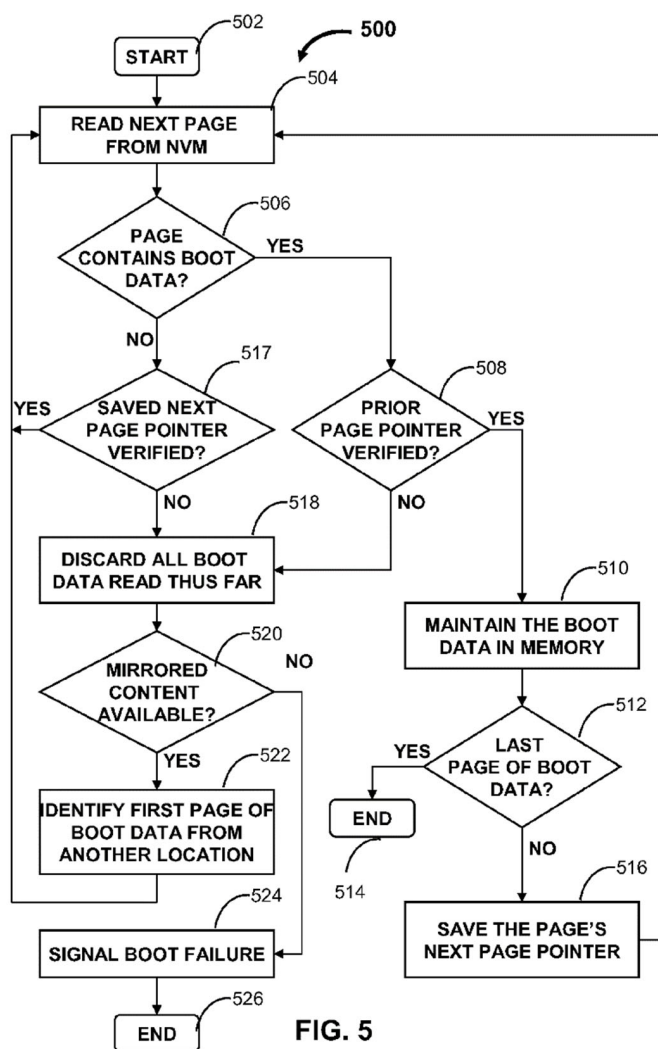
The method of claim 1, further comprising activating a POR sequencer error handler if an error is detected in the POR sequencer descriptor.

150. It is my opinion that claim 5 of the '084 Patent is rendered obvious by the combination of Post, Kao and Wilson.

151. Post's electronic device might detect errors when it used error correction codes (ECC) during the process of searching for boot code pages in the NVM. Post identified three scenarios where the boot data might be determined to be corrupted. The first was where the ECC codes for a page of retrieved data identified uncorrectable errors in the data. (INTEL-1005, 6:60-65 ("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.").) The second situation occurred at step 508 (shown in Figure 5) when Post's system identified the current page of data from the NVM as a boot data page, but this was not the page pointed to by the next page pointer of the previous boot data page. The third situation occurred at step 517 when the current page of data from the NVM was identified as a non-boot data page, but the next page pointer of the previous boot data page identified the current page as the next boot data page.

If, however, at steps 517 or 508, 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.

(*Id.*, 10:8-13.)



152. Post identified a two-step *POR sequencer error handler* process that was used to respond when it was determined that there was an error in the boot code

that prevented the POR sequencer from successfully retrieving boot data and booting the electronic device. The first recovery response was to try to find, and use, an alternative copy of the boot code.

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. If mirrored content is available, the electronic device can begin loading boot data from the other area of the NVM to replace the discarded boot data. In particular, at step 522, the electronic device can identify the first boot data page from another location, such as by using process 400 of FIG. 4, and may then return to step 504 to finish loading boot data from the other location.

(*Id.*, 10:14-23; *see also, id.*, 7:48-50 (“The electronic device may therefore signal a boot failure or may try loading a redundant copy of the boot data from another location in the NVM.”).) If a redundant mirrored copy was not available (or retrieval of that copy also fails), the second step of the error handler process was to alert the user and seek user intervention in the process.

Returning to step 520, the NVM interface may determine that mirrored content is not available, such as if only one copy of the boot data is stored in the NVM or if the electronic device has already unsuccessfully attempted to load all other copies of the boot data. In this case, process 500 may move from step 520 to step 524, at which point a boot failure may be signaled. For example, the electronic device may stop the bootup process and may display an error message to the user. Process 500 may then end at step 526 so that the user can perform a restore operation to update the firmware (including the boot data) stored in the non-volatile memory.

(*Id.*, 10:24-35.) A similar failure might be signaled to the user if the first page of boot data encountered in the NVM did not have a prior page pointer indicating that there was no earlier boot data page.

If the prior page pointer points to a previous page instead of showing that the current page is the first page, this is an indication that the electronic device was previously unsuccessful at reading the first boot data page. ... 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. For example, the electronic device may stop the bootup process and may display an error message to the user. Process 400 may then end at step 418 so that the user can perform a restore operation to update the firmware (including the boot data) stored in the non-volatile memory.”.)

(*Id.*, 8:56-9:2.)

153. Post also detailed how users could respond to the boot failure indication by updating the boot data stored in the NVM. (*Id.*, 10:66-11:3 (“Process 600 may be performed during a firmware update as part of a restore process. The restore process may be performed at a factory during manufacture or repair of the electronic device, or may be initiated by the end user to update the electronic device.”).)

154. The POR sequencer error handler that Post described is comparable to the error handler described in the ’084 patent. The patent describes only that “[i]f an error is found, a POR sequencer error handler is activated in block 612.” (INTEL-1001, 7:7-9.) The error handler is identified in the flow chart in the ’084 patent’s

Figure 6, but no details about the operation of the error handler are detailed. A person of skill in the art at the time of the '084 patent would understand Post's multi-stage process for handling uncorrectable errors as a type of *POR sequencer error handler*, as claimed.

G. Dependent Claim 8

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

155. It is my opinion that claim 8 of the '084 Patent is rendered obvious by the combination of Post with Kao and Wilson.

156. The combined system wherein Post's electronic device uses a DMA controller, as taught in Kao, for boot data transfers from NVM, and includes register configuration information in the boot data, as taught in Wilson, includes *performing a register initialization phase*. Wilson describes that boot up in an embedded system generally involves register initialization.

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, load instructions in its memory, etc.

(INTEL-1013, 1:24-28.)

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. 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. 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. Therefore, these initial values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure.

(*Id.*, 2:53-63.)

157. Figure 3 in Wilson's patent identifies how the register initialization process fits within the larger device boot up process. In the first steps depicted (300, 302 and 304), the boot data was retrieved, potentially from an NVM (*id.*, 5:42-45 ("In another embodiment the boot data can be stored in other types of non-volatile memory, such as, for example FLASH memory.")) and stored in a volatile memory. (*Id.*, 5:60-61 ("The serial port interface can then store data field 402 of the data block in volatile memory 103.")) A "register load assist" (RLA) engine in Wilson's system accessed the stored initialization data to decompress it and copy it to the appropriate system registers. (*Id.*, 6:27-29 ("At step 308, the RLA engine accesses the compressed boot data from the volatile memory, decompresses it and uses it to load the various registers in register file 102.")) The processor was then able to perform the subsequent stages of the boot process that can only be performed after the system has been initialized.

At step 310, the RLA engine has completed its task of decompressing the boot data, and performs a function that indicates that this stage of the boot process is complete. For example, the RLA engine may activate processor 101 by issuing an interrupt which indicates that the boot up process is over and the processor can proceed with ordinary operation. Alternatively, ... it can issue an interrupt to the microprocessor which indicates that a specific stage of the boot up process is completed in order to allow the microprocessor to proceed with additional stages.

(*Id.*, 6:29-41.)

158. Based on Wilson's teachings it would be obvious to a person of skill that the combined system would use Wilson's teachings about the utility of boot codes that include register initialization information to initialize the processor (Wilson's "microcontroller") before the processor began additional boot up operations that depend upon system configuration registers being programmed with initialization values found in the boot data. (INTEL-1013 at 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. ... Therefore, these initial values can be a part of boot data that is loaded in the microcontroller during a boot-up procedure."), 6:27-29 ("At step 308, the RLA engine accesses the compressed boot data from the volatile memory, decompresses it and uses it to load the various registers in register file 102.")) A person of skill would identify this phase of boot up operations, during which the

register initialization information stored in the boot data is stored in the initialization registers for the device, as a *register initialization phase*.

H. Dependent Claim 9

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

159. It is my opinion that claim 9 of the '084 Patent is rendered obvious by the combination of Post, Kao and Wilson.

160. In Post's electronic device the boot data that is copied from NVM to RAM memory 114 includes "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 boot loader, 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. For clarity, the program code stored in boot ROM 116 may sometimes be referred to as "mask ROM code," and the firmware (or other data) retrieved from NVM 120 by the mask ROM code may sometimes be referred to as "boot data."

(INTEL-1005, 3:55-65.) Post's NVM was a storage device (*id.*, 3:2-8 ("NVM 120 can include a NAND flash memory" or other types of memory), and memory 114 was a random access memory (RAM). (*Id.*, 3:44-45 ("Memory 114 can include any suitable type of volatile memory, such as random access memory ('RAM').").)

161. Post also taught that the boot data could be stored in multiple integrated circuit devices. (*Id.*, 3:12-14 ("NVM 120 can include multiple integrated circuits,

where each integrated circuit may have multiple blocks”), 4:33-34 (“The boot data storage area can include any suitable number of integrated circuits, blocks, or pages.”).) A person of skill, therefore, would recognize from Post that parts of the POR sequencer descriptor may originate in different NVM devices before they are transferred from those NVM circuits to the RAM.

162. Furthermore, Post taught that the boot data included a “first stage boot loader, 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.*, 3:58-61.) A person of ordinary skill in the art would identify a second stage bootloader, and the kernel or operating system that it loads, as types of firmware. This is also consistent with the way “firmware” is employed in the ’084 patent. (INTEL-1001, 1:39-41 (“The firmware stored in the nonvolatile memory can be a boot code loader, a basic input/output system (BIOS), an operating system (OS), or an application firmware.”).) Post does not identify the second stage bootloader, kernel and operating system as components of the boot data. (INTEL-1005, 1:39-42.) Post further states that boot data is “loaded from the NVM based on code stored in a boot ROM.” (*Id.*, 1:40-43.) Because the second stage bootloader, kernel and OS are loaded by the first stage bootloader, (*id.*, 3:57-61), which was stored in the NVM, not the boot ROM, (*id.*, 1:41-42 (“a first stage bootloader ... that is loaded from the NVM”)), a person of skill would understand that the second stage

bootloader, kernel and OS are not part Post’s “boot data.” Post further taught that the boot data was stored in a “boot data storage area” within the NVM. (*Id.*, 4:27-30.) A person of skill would understand from these statements in Post that the second stage bootloader, kernel and OS are not contained within the same “boot data storage area” as the first stage bootloader and other boot data. That would suggest that the second stage bootloader, kernel and OS are stored in a different memory, or at least a different part of the NVM. As a result, when these items are copied by the first stage bootloader to the RAM memory, it is obvious that they would, or at least may, originate from a different storage device in Post’s system, including, perhaps, from a different NVM device than the one that contains the “boot data storage area.”

I. Independent Claim 19

163. It is my opinion that claim 13 of the ‘084 Patent is rendered obvious by the combination of Kao and Post.

1. Claim 19 Preamble

An article of manufacture, comprising:

164. I do not believe that a person of skill would understand this preamble to be limiting. The first limitation (a) of the claim provides that the claim describes “computer-readable medium having stored thereon instructions” that implement a specified method. The suggestion in the preamble that what follows is “[a]n article

of manufacture” does not provide antecedent basis and is not necessary to complete the description of what follows in the rest of the claim.

165. If, nevertheless, the preamble is regarded as a claim limitation, Post disclosed an article of manufacture. Post’s electronic device may be one of many manufactured consumer devices including “a portable media player (e.g., an iPod ...), a cellular telephone (e.g., an iPhone ...), a pocket-sized personal computer, a personal digital assistan[t] (“PDA”), a desktop computer, a laptop computer, and any other suitable type of electronic device or system.” (INTEL-1005, 2:61-67.) Additionally, Post’s electronic device includes control circuitry 112, which “can include any suitable components, circuitry, or logic, such as one or more processors,” (*id.*, 3:34-38), memory 114, boot ROM 116, and non volatile memory 120. (*Id.*, 3:1-2, 3:33-34.) A person of skill would recognize that each of these are manufactured items.

2. Claim 19, Limitation a

a non-transitory computer-readable medium having stored thereon instructions operable to permit an apparatus to perform a method comprising:

166. At boot-up, the control circuitry in Post’s electronic device executed program code stored in the boot ROM to search for, identify and retrieve boot code from the non volatile memory. (*Id.*, 1:37-43 (“A NVM provided in an electronic device may include a boot data storage area in which boot data is stored. ‘Boot data’

may refer to any information, such as a bootloader (e.g., a first stage bootloader), that is loaded from the NVM based on code stored in a boot ROM (e.g., a mask ROM).”), 3:53-57 (“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.”).) The control circuitry included one or more processors (*id.*, 3:36-38), and a boot ROM was a computer-readable medium, as it provided program code to the control circuitry processors. (*Id.*, 3:41-43 (“At bootup of electronic device 100, control circuitry 112 may operate under the control of program code stored in boot ROM 116.”).)

167. The manner in which the program code stored in the boot ROM was used by Post’s control circuitry to perform the methods of the remainder of this claim is discussed in detail in each element below. Generally, the boot ROM implemented the procedures for searching, identifying and retrieving boot data from the non volatile memory (NVM) in accordance with the procedures illustrated in Post’s Figures 4 and 5. (*Id.*, 8:7-15 (“Referring now to FIGS. 4-6, flowcharts of illustrative processes are shown in accordance with various embodiments of the invention. The steps of these illustrative processes may be performed by any suitable component ... such as by control circuitry 112 of FIG. 1. ... [T]hese components may operate under

the control of mask ROM code (e.g., from boot ROM 116) or from an operating system and NVM driver.”).)

168. Combining Post with Kao, with the result that the DMA engine for copying boot data from an NVM disclosed in Kao was incorporated into Post’s electronic device, would not change the resulting system’s reliance on the program code stored in Post’s boot ROM to implement the procedures recited in claims 19 and 20. Kao describes that the DMA engine operates in accordance with data transfer parameters provided to it. (INTEL-1004, 8:15-31, 9:16-33 (parameters for transfer of data may come from a variety of sources).) Post’s method requires specific pages of data to be retrieved from the NVM in accordance with the procedures illustrated in Post’s Figures 4 and 5 or in accordance with a process by which pages are selectively read so that non-data pages are skipped over. (INTEL-1005, 10:48-49.) It would be obvious to a person of skill, therefore, that the DMA engine parameters for retrieving the necessary pages must be supplied by the control circuitry that manages the boot code search and retrieval process, as directed by the code contained in the boot ROM.

3. Claim 19, Limitation b

releasing components of an embedded system from reset;

169. This limitation is also recited in claim 1, limitation a. For the same reasons explained above (at page 43) for claim 1, limitation a, the combination of

Kao and Post rendered obvious “*releasing components of an embedded system from reset.*”

4. Claim 19, Limitation c

detecting a power-on reset (POR) sequencer descriptor in a nonvolatile memory;

170. This limitation is also recited in claim 1, limitation b. For the same reasons explained above (at page 48) for claim 1, limitation b, the combination of Kao and Post rendered obvious “*releasing components of an embedded system from reset.*”

5. Claim 19, Limitation d

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;

171. This limitation is also recited in claim 1, limitation c. For the same reasons explained above (at page 72) for claim 1, limitation c, the combination of Kao and Post rendered obvious “*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.*”

6. Claim 19, Limitation e

copying the POR sequencer descriptor from the nonvolatile memory to a random access memory;

172. This limitation is also recited in claim 1, limitation d, except that the latter recites the acronym for “random access memory.” For the same reasons explained above (at page 77) for claim 1, limitation d, the combination of Kao and Post rendered obvious “*copying the POR sequencer descriptor from the nonvolatile memory to a random access memory.*”

7. Claim 19, Limitation f

verifying an integrity of the POR sequencer descriptor; and

173. This limitation is also recited in claim 1, limitation e, except that the latter omits the conjunction “and.” For the same reasons explained above (at page 82) for claim 1, limitation e, the combination of Kao and Post rendered obvious “*verifying an integrity of the POR sequencer descriptor; and.*”

8. Claim 19, Limitation g

detecting any error in the POR sequencer descriptor.

174. This limitation is also recited in claim 1, limitation f. For the same reasons explained above (at page 84) for claim 1, limitation f, the combination of Kao and Post rendered obvious “*detecting any error in the POR sequencer descriptor.*”

J. Dependent Claim 20

The article of manufacture of claim 19, wherein the instructions are further operable to permit the apparatus to activate a POR sequencer error handler if an error is detected in the POR sequencer descriptor.

175. It is my opinion that claim 20 of the '084 Patent is rendered obvious by the combination of Kao and Post.

176. The limitation recited in claim 20 is substantially similar to the limitation in claim 5 requiring “*activating a POR sequencer error handler if an error is detected in the POR sequencer descriptor.*” For the same reasons explained above (at page 94) for claim 5, the combination of Kao and Post rendered obvious “*wherein the instructions are further operable to permit the apparatus to activate a POR sequencer error handler if an error is detected in the POR sequencer descriptor.*”

XI. CONCLUSION

177. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

178. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be

true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Executed this 31st day of March 2023 in Los Altos, California.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Donald Alpert". The signature is fluid and cursive, with the first name "Donald" and last name "Alpert" clearly distinguishable.

Donald Alpert, Ph.D.

APPENDIX A



Donald Alpert, Ph.D.
Camelback Computer Architecture, LLC
2310 Homestead Road
Suite C1 #411
Los Altos CA 94024
(650) 386-5301

Donald Alpert, Ph.D., is an independent consultant, specializing in microprocessors and computer systems. For more than a decade in his role as president of Camelback Computer Architecture LLC, Dr. Alpert has provided clients with expert services for patent litigation, technology assessment, and product development. He has over 40 years of wide-ranging academic and industry experience that enable him to provide expert consulting services in the following areas:

- Computers, Computer Systems
- Computer architecture
- Microprocessors
- Semiconductors
- PC Systems
- Embedded Systems
- Design Validation and Debug
- Performance Evaluation
- Cache Memory
- Instruction Sets
- Media Processing
- Power Management

Patent Analysis: Dr. Alpert has served as an expert on dozens of patent litigation cases, handling both infringement and invalidity issues. Dr. Alpert has worked as both a consulting expert and testifying expert assisting attorneys representing major high-tech corporations. He has substantial experience testifying in depositions and at trials before U.S. Federal Courts and the International Trade Commission. Dr. Alpert also consults to assess patents for acquisition and for pre-litigation review. See [here](#) for a summary of testifying experience.

Product Development: Dr. Alpert is a widely recognized innovator with over 30 patents. Dr. Alpert has worked at some of the most ground-breaking technology firms, including Zilog, National Semiconductor, and Intel, where he was the principal architect of the Pentium® Processor. See [here](#) for more information and example projects.

Employment Experience

Camelback Computer Architecture, LLC, Sacramento, CA

Consultant

2/02-Present

Computer architecture consulting services to assess technology, develop products, and protect IP.

Intel Corporation, Chandler, Arizona

Principal Engineer

12/99-1/02

Led architecture team of 10 responsible for definition and performance evaluation of Itanium™ Processor Family microprocessors.

Principal Engineer

5/98-12/99

Led architecture team of 4 responsible for technical strategy and product definition of PC graphics processors and chipsets.

Stanford University, Stanford, California

Consulting Assistant Professor

9/97-4/98

Responsible for teaching undergraduate and graduate classes in computer architecture and participating in processor-design research projects.

Intel Corporation, Santa Clara, California

Engineering Manager

10/93-6/97

Responsible for microarchitecture definition and architecture validation of the Itanium™ microprocessor, which was the first implementation of Intel's 64-bit extension architecture. Led a team of 60 engineers.

Architecture Manager

1/89-10/93

Responsible for the Pentium® Processor technical definition. Worked with customers and Intel marketing to identify product requirements and with the design engineering team to evaluate design options. Led a team of seven engineers responsible for creating technical specifications and for modeling and simulating performance using traces from DOS, Windows, and UNIX execution. After completing the Pentium Processor, led an interdisciplinary team of 30 engineers to define a new 64-bit instruction set architecture.

National Semiconductor Corporation, Herzlia, Israel

Architecture Manager

5/85-1/89

Responsible for defining new microprocessors in the Series 32000 family, including the NS32532 and "Swordfish" high-performance microprocessors. Led a team of five engineers working in the following areas: modeling and simulating high-performance microprocessors, evaluating novel computer architectures, and writing technical specifications for new products.

Zilog, Incorporated, Campbell, California

Component Architect

6/80-4/85

Worked on the Z80,000 CPU, a 32-bit microprocessor that integrated an instruction pipeline, cache, and memory management unit. Responsible for specifying the non-privileged instruction set, evaluating performance, and writing the reference manual for the microprocessor.

Burroughs Corporation, Pasadena, California

Associate Engineer

7/76-7/77

Designed computer peripheral interface controllers using MSI-TTL and 8080 microprocessors.

Alpex Computer Corporation, Danbury, Connecticut

8/75-6/76

Consulting engineer working on design of video games.

Action, Washington, DC

6/73-6/75

Peace Corps volunteer teaching mathematics and physics in a secondary school in Ghana, West Africa.

Education

Stanford University
Graduate study in computer design. National Science Foundation Graduate Fellowship.
Awarded MSEE in 1978 and PhDEE in 1984.

M.I.T.	9/69-6/73
Member of Tau Beta Pi and Eta Kappa Nu. Awarded BSEE in 1973.	

Professional Activities

Member of Microprocessor Report editorial board (2002-present)

Anthem College Commencement Address (18-Dec-2009)

Panel session about history of microprocessor development at National Semiconductor Corporation, Computer History Museum (2008).
[\[http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf\]](http://archive.computerhistory.org/resources/access/text/Oral_History/102658246.05.01.acc.pdf)

IEEE Technical Committee on Microprocessors and Microcomputers, Chair (1999-2000) and Vice-Chair (2001-present).

Lectures about the Itanium™ Processor Family at U.C. Irvine (1999, 2000).

Cool Chips I, Keynote speaker (Tokyo, 1998)

IEEE Hot Chips Symposium Program Co-chair (1994), member of program committee (1993, 1995).

Lectures about the Pentium® Processor at Stanford University, University of Illinois, and Cornell (1993-1994).

Appeared in Stanford UVC Educational Video production about the Pentium Processor (1993).

IEEE International Symposium on Computer Architecture member of program committee (1996) and Session Chair for Cache Design (1989).

Teaching Experience

Faculty Associate at Arizona State University teaching undergraduate class on assembly language programming (Spring/2001) and computer architecture (Spring/2002).

Consulting Associate Professor at Stanford University teaching undergraduate and graduate classes in computer architecture (Fall/1997, Winter/1998).

Consulting Professor at Tel Aviv University teaching graduate class in computer architecture (Spring/1987)

Peace Corps volunteer in a secondary school in Ghana, West Africa teaching mathematics and physics (1973-1975).

Don Alpert, "How Microprocessors Upset the Computer Industry," Microprocessor Report, December 11, 2006.

Don Alpert, "Evolving PC Chip-Set Topologies," Microprocessor Report, April 17, 2006.

Don Alpert, "Will Microprocessors Become Simpler?" Microprocessor Report, November 17, 2003.

Don Alpert, "Itanium Processor Status Report," Microprocessor Report, July 28, 2003.

Don Alpert, "Scalable MicroSupercomputers," Microprocessor Report, March 17, 2003.

Don Alpert, "ColdFire Goes Fully Superscalar," Microprocessor Report, October 28, 2002.

Donald Alpert and Alan Jay Smith, "Guest Editor Introduction," IEEE Micro special issue on Hot Chips VI, April 1995.

Donald Alpert and Dror Avnon, "Architecture of the Pentium Processor," IEEE Micro, June 1993, pp. 11-21. Selected best paper in IEEE Micro for 1993.

D. Alpert, A. Averbuch, and O. Danielli, "Performance Comparison of Load/Store and Symmetric Instruction Set Architectures," Proceedings, 18th International Symposium on Computer Architecture, June 1990, pp. 172-181.

Benjamin Maytal, Sorin Iacobovici, Donald Alpert, et al, "Design Considerations for a General-Purpose Microprocessor," Computer, January 1989, pp. 66-76.

Donald Alpert and Michael Flynn, "Performance Trade-offs for Microprocessor Cache Memories," IEEE Micro, August 1988, pp. 44-54.

D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 Microprocessor," Proceedings, 1987 IEEE International Conference on Computer Design, October 1987, pp. 168-172. Awarded best presenter at Design and Test sessions.

D. Alpert, D. Biran, L. Epstein, et al, "Trends in VLSI Microprocessor Design," Proceedings, First Annual Conference on Computer Technology, Systems and Applications (CompEuro '87), May 1987, pp. 564-567.

Y Sidi, D. Alpert, D. Biran, et al, "Design Considerations of an Advanced 32-Bit Microprocessor," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

D. Alpert, "Trends in VLSI Microprocessor Design," Proceedings, 15th Conference of Electrical and Electronic Engineers in Israel, April 1987.

Donald Alpert, Michael J. Flynn, and Scott Wakefield, "Directly Executed Languages for VLSI Processor Design," Proceedings, 1983 IEEE International Conference on Computer Design, October 1983, pp. 609-612.

Donald Alpert, "Powerful 32-Bit Micro Includes Memory Management," Computer Design, October 1983, pp. 213-220.

Don Alpert, Dean Carberry, Mike Yamamura, et al, "32-Bit Processor Chip Integrates Major System Functions," Electronics, July 14, 1983, pp. 113-119.

Patents

4,802,085, Apparatus and method for detecting and handling memory-mapped I/O by a pipelined microprocessor
5,249,286, Selectively locking memory locations within a microprocessor's on-chip cache
5,263,153, Monitoring control flow in a microprocessor
5,416,913, Method and apparatus for dependency checking in a multi-pipelined microprocessor
5,442,756, Branch prediction and resolution apparatus for a superscalar computer processor
5,475,824, Microprocessor with apparatus for parallel execution of instructions
5,479,652, Microprocessor with an external command mode for diagnosis and debugging
5,481,751, Apparatus and method for storing partially-decoded instructions in the instruction cache of a CPU having multiple execution units
5,559,986, Interleaved cache for multiple accesses per clock cycle in a microprocessor
5,606,676, Branch prediction and resolution apparatus for a superscalar computer processor
5,617,554, Physical address size selection and page size selection in an address translator
5,621,886, Method and apparatus for providing efficient software debugging
5,638,525, Processor capable of executing programs that contain RISC and CISC instructions
5,657,253, Apparatus for monitoring the performance of a microprocessor
5,659,679, Method and apparatus for providing breakpoints on taken jumps and for providing software profiling in a computer system
5,669,011, Partially decoded instruction cache
5,671,435, Technique for software to identify features implemented in a processor
5,675,825, Apparatus and method for identifying a computer microprocessor
5,692,167, Method for verifying the correct processing of pipelined instructions including branch instructions and self-modifying code in a microprocessor
5,729,724, Adaptive 128-bit floating point load and store operations for quadruple precision compatibility
5,740,413, Method and apparatus for providing address breakpoints, branch breakpoints, and single stepping
5,764,959, Adaptive 128-bit floating point load and store instructions for quad-precision compatibility
5,774,686, Method and apparatus for providing two system architectures in a processor
5,790,834, Apparatus and method using an ID instruction to identify a computer microprocessor
5,802,605, Physical address selection and page size selection in an address translator
5,958,037, Apparatus and method for identifying features and the origin of a computer microprocessor
5,991,874, Conditional move using a compare instruction generating a condition field
6,052,801, Method and apparatus for providing breakpoints on a selectable address range
6,219,774, Address translation with/bypassing intermediate segmentation translation to accommodate two different instruction set architecture
6,385,718, Computer system and method for executing interrupt instructions in operating modes
6,408,386, Method and apparatus for providing event handling functionality in a computer system
6,584,558, Article for providing event handling functionality in a processor supporting different instruction sets
7,010,671, Computer system and method for executing interrupt instructions in two operating modes
7,389,403, Adaptive computing ensemble microprocessor architecture
7,802,073, Virtual core management

Donald Alpert - Expert Witness Testimony

PARTIES	DISTRICT	DIVISION	CASE NUMBER	TESTIMONY
Intergraph Hardware Technologies Company vs. Hewlett-Packard Company	Eastern District, TX	Marshall	2-02CV-312-TJW	Deposition 4/2004
MediaTek Inc. vs. Sunext Technology Co., Ltd.	International Trade Commission		Investigation No. 337-TA-523	Deposition 4/2005 Hearing 7/2005
Asyst Technologies, Inc. vs. Jenoptik AG	Northern District, CA	San Jose	98-20451-JF	Deposition 8/2006 Trial 1/2007
Cornell University vs. Hewlett-Packard Company	Northern District, NY	Syracuse	5:01-CV-1974	Deposition 9/2006 Trial 5/2008 Trial 7/2008
LG Electronics, Inc. vs. Quanta Computer Inc.,	Northern District, CA	Oakland	C 01-0287 CW	Deposition 7/2007
Network Appliance, Inc. vs. Sun Microsystems Inc.	Northern District, CA	San Francisco	C-07-06053-EDL	Markman 8/2008
HTC Corp. vs. Apple Inc.	International Trade Commission		Investigation No. 337-TA-721	Markman 10/2010 Deposition 3/2011 Hearing 5/2011
HTC Corp. vs. Apple Inc.	International Trade Commission		Investigation No. 337-TA-808	Markman 4/2012 Deposition 7/2012
Applied Micro Circuits Corp. vs. Sandforce, Inc.	Superior Court of California	Santa Clara County	Case No. 110CV190188	Deposition 3/2013
Intellectual Ventures vs. Motorola Mobility	District of Delaware		Case No. 1:11-cv-00908-SLR	Deposition 6/2013 Trial 1/2014 Retrial 3/2015
Maxim Integrated Products vs. Starbucks, et al.	Western District of Pennsylvania	Pittsburgh	Multi-District Litigation No. 2354	Deposition 7/2013 Deposition 12/2014
NXP vs. RIM	Middle District Of Florida	Orlando	Case No. 6:12-cv-00498-ACC-TBS	Deposition 9/2013 Trial 3-4/2014
Sandisk vs. Netlist	Patent Trial and Appeal Board		IPR2014-00970 IPR2014-00971	Deposition 2/2015 Deposition 6/2015
PLL Technologies vs. Xilinx	Patent Trial and Appeal Board		IPR2015-00148	Deposition 8/2015 Deposition 12/2015

Donald Alpert - Expert Witness Testimony

PARTIES	DISTRICT	DIVISION	CASE NUMBER	TESTIMONY
PUMA vs. Apple	Eastern District of Texas	Marshall	Case No.2:15-cv-00621	Deposition 8/2016
Intellectual Ventures vs. EMC	Patent Trial and Appeal Board		IPR2016-01106	Deposition 5/2017
SK Hynix vs. Netlist	Patent Trial and Appeal Board		IPR2017-00548	Deposition 8/2017
Intellisoft vs. Acer	Superior Court of California	Santa Clara County	Case No. 1-14-CV-272381	Deposition 10/2017
Visual Effect Innovations vs. NVIDIA	Northern District, CA	San Francisco	Case No. 3:17-cv-03187-VC	Deposition 12/2017
Qualcomm vs. Apple	International Trade Commission		Investigation No. 337-TA-1093	Deposition 7/2018 Hearing 9/2018
SK Hynix vs. Netlist	Patent Trial and Appeal Board		IPR2018-00303	Deposition 7/2018
TecSec vs. Cisco	Eastern District, VA	Alexandria	Case No. 1:10-cv-00115-LO-TCB	Deposition 9/2020
SK Hynix vs. Netlist	Patent Trial and Appeal Board		IPR2020-01044 IPR2020-01042	Deposition 3/2021
Microchip vs. HD Silicon Solutions	Patent Trial and Appeal Board		IPR2021-01567	Deposition 6/2022
Acer vs. ACQIS	Eastern District of Texas	Marshall	Case No. 2:21-cv-00275-JRG-RSP	Deposition 7/2022
Samsung vs. Netlist	Patent Trial and Appeal Board		IPR2022-00062 IPR2022-00064	Deposition 7/2022
Micron vs. Netlist	Patent Trial and Appeal Board		IPR2022-00263	Deposition 9/2022
Intel vs. BitMICRO	Western District of Texas	Waco	Case No. 6:22-cv-00335	Deposition 11/2022

APPENDIX B

APPENDIX B – U.S. PATENT NO. 9,858,084 CHALLENGED CLAIM LIMITATIONS

Claim Limitation	'084 Patent Claim Language
1[pre]	A computer-implemented method, comprising:
1[a]	releasing components of an embedded system from reset;
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;
1[c]	copying the POR sequencer descriptor from the nonvolatile memory to a random access memory (RAM);
1[d]	verifying an integrity of the POR sequencer descriptor; and
1[e]	detecting any error in the POR sequencer descriptor;
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.
5	The method of claim 1, further comprising activating a POR sequencer error handler if an error is detected in the POR sequencer descriptor.
8	The method of claim 1, further comprising: performing a register initialization phase in the embedded system.
9	The method of claim 1, further comprising: copying system firmware from a storage device to the RAM.
19[pre]	An article of manufacture, comprising:
19[a]	a non-transitory computer-readable medium having stored thereon instructions operable to permit an apparatus to perform a method comprising:
19[b]	releasing components of an embedded system from reset;

Claim Limitation	'084 Patent Claim Language
19[c]	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;
19[d]	copying the POR sequencer descriptor from the nonvolatile memory to a random access memory;
19[e]	verifying an integrity of the POR sequencer descriptor; and
19[f]	detecting any error in the POR sequencer descriptor.
20	The article of manufacture of claim 19, wherein the instructions are further operable to permit the apparatus to activate a POR sequencer error handler if an error is detected in the POR sequencer descriptor.