UNITED STATES PATENT AND TRADEMARK OFFICE

_____

BEFORE THE PATENT TRIAL AND APPEAL BOARD
_____

Intel Corporation
Petitioner

v.

Qualcomm Incorporated
Patent Owner of U.S. Patent No. 8,838,949
Claims 10-17

_____

Trial No. IPR2018-01335

_____

**DECLARATION OF BILL LIN, PH.D.
ON BEHALF OF PETITIONER**

TABLE OF CONTENTS

I, Bill Lin, Ph.D. declare as follows:

I.   BACKGROUND

1.   I am currently Professor and Vice Chair of Electrical and Computer Engineering at the University of California, San Diego (UCSD). I am also Adjunct Professor of Computer Science and Engineering at UCSD.

2.   My Curriculum Vitae, which states my qualifications more fully, is attached as Appendix A. A list of all cases in which I have testified as an expert at trial or by deposition in the last four years is also included in Appendix A.

3.   I received a Bachelor's of Science degree in 1985, a Master's of Science degree in 1988, and a Ph.D. in 1991, all in Electrical Engineering and Computer Sciences from the University of California, Berkeley.

4.   I joined UCSD in 1997, and I have been a tenured professor since 1999. My teaching and research has focused on computer architecture and computer network problems, including the design of multiprocessor and multi-core processor architectures, multiprocessor and multi-core processor interconnection buses and networks, network processors, systems-on-chips, and data networks. I regularly teach a senior-level design course on the design of advanced processors, and I have taught graduate courses in hardware/software co-design and advanced special topics in computer architecture.

5.   At UCSD, I am a Principal Investigator in the UCSD Center for Networked Systems (CNS). CNS brings together researchers to work on a range of challenges in the design of future networked systems. My contribution to CNS has been expertise in the design of computer architecture solutions for packet processing and computer networking. I am also a Principal Investigator in the UCSD Center for Wireless Communications (CWC). CWC brings together researchers to work on a range of challenges in the design of future wireless communications systems. My contribution to CWC has been expertise in the design of multi-core processor architectures for wireless communications and mobile computing.

6.   Prior to joining UCSD, I was the Head of the Systems and Communications Group at IMEC in Leuven, Belgium, where I led a team of researchers who worked on a range of computer design problems, including hardware/software co-design, processor interfaces, multiprocessor and multi-core processor design methodologies, and specialized processors for wireless communications and computer networking.

7.   During my career, I have received or worked on research efforts that received millions of dollars in research funding from both government agencies and industry, including funding for research in multi-core processor design,

system-on-chips, hardware/software co-design, packet processing, and computer networks.

8.   I have served as an Associate or Guest Editor for several journals published by the Association for Computing Machinery ("ACM") and the Institute of Electrical and Electronics Engineers ("IEEE"). I have also served as General Chair of several ACM/IEEE conferences, and on the Organizing or Steering Committees of many ACM/IEEE conferences, and I have served on the Technical Program Committees of numerous ACM/IEEE conferences. I am the author of over 170 peer-reviewed publications in the field of computer engineering dating to the 1980s, including journal articles, conference papers, book chapters, technical reports, and invited papers. A number of these publications have received best paper awards or distinguished paper citations. I have also given numerous invited and keynote talks around the world. A list of my publications within the last ten years is included in my CV.

9.   I am the inventor of five patents: U.S. Patent Nos. 8,443,444, 7,860,004, 7,672,005, 5,870,588, and 5,748,487.

10.   I have been retained by counsel for Intel Corporation ("Petitioner") as an independent expert witness for the above captioned Petition for *Inter Partes* Review of U.S. Patent No. 8,838,949 (the "'949 patent") (Ex-1101). I am being compensated at my normal hourly consulting rate of $550 for my work. My

compensation is not dependent on and in no way affects the substance of my statements in this Declaration.

11.   I have no financial interest in the Petitioner. I similarly have no financial interest in the '949 patent, and have had no contact with the named inventors of the '949 patent.

II.   MATERIALS CONSIDERED

12.   I have reviewed the specification, claims, and file history of the '949 patent. I understand that the '949 patent claims priority to U.S. Provisional Application No. 61/324,035, filed April 14, 2010, U.S. Provisional Application No. 61/316,369, filed March 22, 2010, U.S. Provisional Application No. 61/324,122, filed April 14, 2010, and U.S. Provisional Application No. 61/325,519, filed April 19, 2010.

13.   I have also reviewed the following references, all of which I understand to be prior art to the '949 patent:

- U.S. Patent Application Publication No. US2006/0288019A1 to Bauer et al. ("Bauer") (Ex-1109)

- U.S. Patent No. 7,356,680 to Svensson et al. ("Svensson") (Ex-1110).

- Korean Patent Application Publication No. 10-2002-0036354 to Kim ("Kim") (Ex-1111) (English language translation – Ex-1112).

- U.S. Patent Application Publication No. 2007/0140199 to Zhao et al. ("Zhao") (Ex-1113).

14.   In addition to the documents listed above, I have also reviewed the file history of the '949 patent, all of the documents listed in Petitioner's List of Exhibits in the accompanying petition and all other documents cited in this declaration.

III.   LEGAL PRINCIPLES

15.   I am not an attorney. For the purposes of this declaration, I have been informed about certain aspects of the law that are relevant to my opinions. My understanding of the law is as follows:

A.   Claim Construction

16.   I have been informed that claim construction is a matter of law and that the final claim construction will ultimately be determined by the Board. For the purposes of my analysis in this proceeding and with respect to the prior art, I have been informed that patents are currently reviewed in an *inter partes* review (IPR) proceeding under the "broadest reasonable interpretation" standard

(hereinafter "BRI standard"). I also have been informed that IPRs may soon be reviewed under what is known as "the Phillips standard."

17. I have been informed that the BRI standard refers to the broadest reasonable interpretation that a person of ordinary skill in the art would give to a claim term in light of the specification.

18. I have been informed that under the Phillips standard, claim terms are generally given their plain and ordinary meaning as understood by a person of ordinary skill in the art at the time of the invention, with the claim term read not only in the context of the particular claim in which the disputed term appears, but also in the context of the entire patent, including the specification.

19. I have been informed that the patentee can serve as his or her own lexicographer. As such, if a claim term is provided with a specific definition in the specification, that claim term should be interpreted in light of the particular definition provided by the patentee.

B. Anticipation

20. I have been informed and understand that a patent claim is invalid if it is "anticipated" by prior art. For the claim to be invalid because it is anticipated, all of its requirements must have existed in a single device or method that predates the claimed invention, or must have been described in a single publication or patent that predates the claimed invention. A patent claim may be "anticipated" if

each element of that claim is present either explicitly, implicitly, or inherently in a single prior art reference. I have also been informed that, to be an inherent disclosure, the prior art reference must necessarily disclose the limitation, and the fact that the reference might possibly practice or contain a claimed limitation is insufficient to establish that the reference inherently teaches the limitation.

C. Obviousness

21. I have been informed and understand that a patent claim is invalid if the claimed invention would have been obvious to a person of ordinary skill in the art at the time the application was filed. This means that, even if all of the requirements of a claim are not found in a single prior art reference, the claim is not patentable if the differences between the subject matter in the prior art and the subject matter in the claim would have been obvious to a person of ordinary skill in the art at the time the application was filed.

22. I have been informed and understand that a determination of whether a claim would have been obvious should be based upon several factors, including, among others:

- the level of ordinary skill in the art at the time the application was filed;

- the scope and content of the prior art; and

- what differences, if any, existed between the claimed invention and the prior art.

23. I have been informed and understand that the teachings of two or more references may be combined in the same way as disclosed in the claims, if such a combination would have been obvious to one having ordinary skill in the art. In determining whether a combination based on either a single reference or multiple references would have been obvious, it is appropriate to consider, among other factors:

- whether the teachings of the prior art references disclose known concepts combined in familiar ways, which, when combined, would yield predictable results;

- whether a person of ordinary skill in the art could implement a predictable variation, and would see the benefit of doing so;

- whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art;

- whether a person of ordinary skill would have recognized a reason to combine known elements in the manner described in the claim;

- whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and

- whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

24. I understand that one of ordinary skill in the art has ordinary creativity, and is not an automaton.

25. I understand that in considering obviousness, it is important not to determine obviousness using the benefit of hindsight derived from the patent being considered.

26. I have been informed and understand that a single reference alone can render a patent claim obvious, if any differences between that reference and the claims would have been obvious to a person of ordinary skill in the art at the time of the alleged invention—that is, if the person of ordinary skill could readily adapt the reference to meet the claims of the patent, by applying known concepts to achieve expected results in the adaptation of the reference.

D.   Means-Plus-Function Claims

27. I have been informed and understand that for means-plus-function limitations, a prior art reference or combination of references must disclose the identical function in the claim limitation and must disclose a structure that

performs the function that is either identical to or the equivalent of the structure in the specification of the challenged patent that performs the claimed function. I understand that a structure is equivalent if it performs the identical function in substantially the same way to achieve substantially the same result as the claim limitation at issue.

IV.   SUMMARY OF OPINIONS

28.   It is my opinion that every limitation of claims 10-17 of the '949 patent is disclosed by the prior art, and that claims 10-17 are rendered obvious by the prior art cited in this declaration.

V.   BRIEF DESCRIPTION OF THE TECHNOLOGY

A.   Multi-Processor Systems

1.   Processor-To-Processor Communications

29.   The '949 patent generally relates to communications between processors. Processors are common components in electrical devices that perform various functions to make the devices operate. Electrical devices may have multiple processors to handle different responsibilities. For example, a mobile telephone may include a "baseband" processor—which the '949 patent calls a "modem" processor—and an "application" processor. Ex-1101, 1:41-44.

30.   The baseband/modem processor typically performs tasks relating to the transmission and reception of data to/from other devices over a network such as

a wireless communication network. For example, Figure 5 of the '949 patent shows a mobile telephone 520 communicating with base stations 540 in a wireless communication system 500.
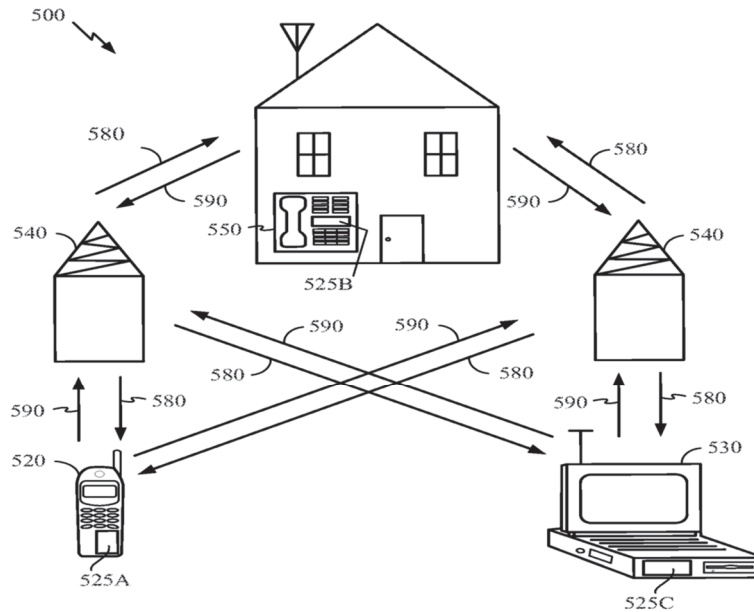
FIG. 5

Ex-1101, Fig. 5.

31.   The baseband/modem processor in the mobile telephone 520 is responsible for sending data to and receiving data from base stations 540. The base stations 540 facilitate communication between the mobile telephone 520 and other devices, such as a portable computer 530, in the wireless communication system 500. Ex-1101, 11:25-39, Fig. 5.

32.   The application processor typically runs applications and other computer programs on the mobile telephone—*e.g.,* email applications, video chat, text messaging, phone applications, GPS applications, etc.

33.   The baseband/modem and application processors need to communicate with each other. For example, when a user of a mobile telephone composes an email or text message using an application running on the application processor, the application processor must send the message to the baseband/modem processor so that the baseband/modem processor can transmit the message to the base station. Similarly, when a mobile telephone receives data from the base station, the baseband/modem processor receives the data and then transfers it to the application processor so that the user can view the data in an email or other application.

34.   The baseband/modem and application processors typically communicate with each other by sending pieces of data over a "bus." A bus, sometimes referred to as an "interface," is typically a set of wires over which processors send electrical signals to each other. For example, Figure 2 of the '949 patent shows two processors (Application Processor 204 and Modem Processor 210) connected by Inter-Chip Communication Bus 234.
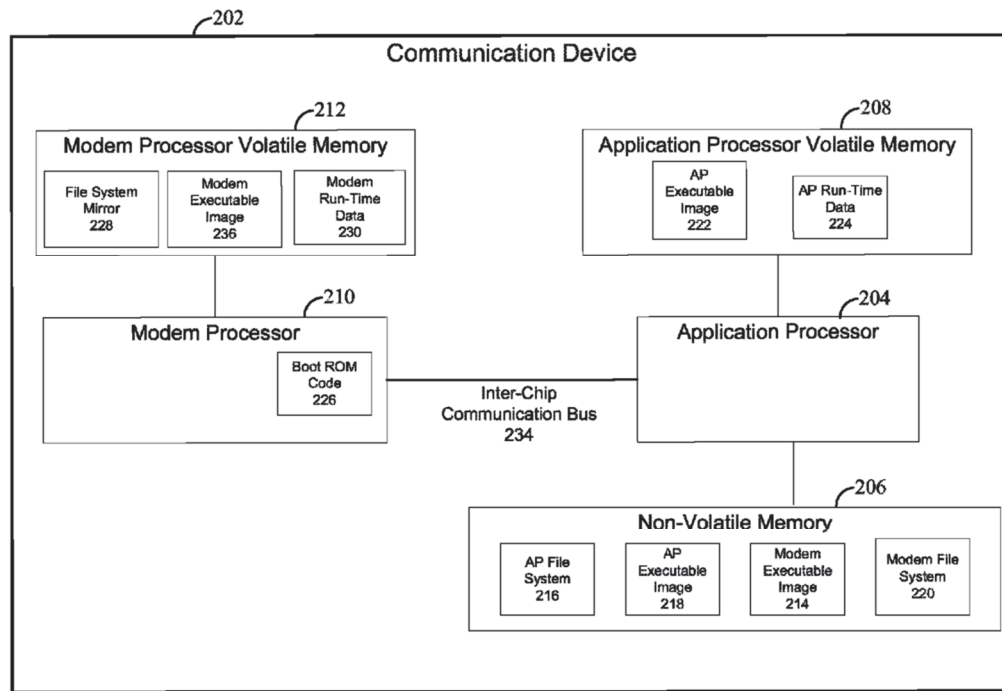
FIG. 2

Ex-1101, Fig. 2.

35.   Many different types of buses were known prior to the alleged invention of the '949 patent. To enable compatibility between processors of different manufacturers, buses usually operate according to one of a number of well-known standards. Standardized buses that were commonly used in mobile telephones and other multi-processor devices include High Speed Synchronous Interface (HSI), Universal Serial Bus (USB), USB High Speed Inter-Chip (HSIC), Mobile Industry Processor Interface (MIPI), Secure Digital Input/Output (SDIO), Universal Asynchronous Receiver-Transmitter (UART), Serial Peripheral Interface (SPI) and Inter-Integrated Ciruit (I^2C). *See, e.g.,* Ex-1101, 5:35-43; *see also* Ex-

1113, ¶32, Fig. 5 (disclosing interfaces such as "one or more universal serial bus (USB) interfaces, micro-USB interfaces, universal asynchronous receiver-transmitter (UART) interfaces, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth").

2. Processor Software Code

36.   A processor operates by executing software code that instructs the processor to perform specific operations. There are different types of software code for performing different types of operations. For example, when a processor is initially powered up, it typically executes "boot code" that instructs the processor to perform certain initialization operations. Such initialization operations may include determining what other devices may be connected to the processor and where such other devices may be located. For example, the boot code may instruct the processor to determine addresses associated with hardware peripherals, such as a keypad, a visual display, and memory.

37.   After the processor executes its boot code, it typically executes "program code" that instructs the processor to perform various operations that the processor has been designated to perform. For example, in the case of the above-described baseband/modem processor, the program code may instruct the baseband/modem processor to transfer received data to the application processor so that a user can view the data in an email or other application. In the case of the

above-described application processor, the program code may instruct the

application processor to send a message to the baseband/modem processor so that

the baseband/modem processor can transmit the message to the base station.

3. Characteristics of Memory

38. To be executed, software code must be stored in memory that is

accessible to the processor. The processor reads the code from the memory and

then executes the code. There are basically two types of memory—non-volatile

memory and volatile memory. Non-volatile (or persistent) memory is designed to

store code and other data regardless of whether power is being applied to the

memory. In contrast, volatile memory can only store code and other data when

power is being applied to the memory. That is, once power is removed from

volatile memory, all code and other data previously stored in the memory will be

lost.

39. Examples of non-volatile memory include electrically erasable

programmable read-only memory (EEPROM) and flash memory. These types of

non-volatile memory, as well as others, have characteristics that make them

suitable for long-term persistent storage. For example, non-volatile memory can

store code and other data for long periods of time after they have been initially

programmed regardless of whether power is being applied to the memory.

However, non-volatile memory typically costs more, provides lower performance

(*e.g.,* operates slower), and requires more space than volatile memory.

40.   Examples of volatile memory include random access memory (RAM),

dynamic RAM (DRAM) and static RAM (SRAM). These types of volatile

memory, as well as others, have characteristics that make them suitable for short-

term storage. For example, code and other data can be quickly stored and retrieved

from volatile memory, thereby increasing system performance. But any code or

other data stored in volatile memory is lost after power is removed from the

memory, so using volatile memory for long-term storage is typically not feasible in

devices that may lose power (*e.g.,* mobile telephones).

41.   A data buffer is typically used as a temporary storage area that allows

data to be moved from one location to another. The data buffer is often some

portion of volatile memory.

B.   Storing, Loading, and Executing Processor Software Code

      1.   Storing the Processor Software Code in Memory

42.   Software code is typically stored, at least initially, in non-volatile

memory. The code is often later transferred from non-volatile memory to volatile

memory, which is typically faster (and can be less expensive) than non-volatile

memory. It is common for system designers to have processors use a type of

volatile memory as a work space where the processor can execute software and

perform other processing functions. When coupled to a processor, engineers will

often refer to this type of volatile memory coupled to the processor as "system

memory."

43. Software code is typically packaged and stored in memory as a

software file or program called an "executable image" or "executable software

image." The '949 patent makes clear that executable software images were known

in the prior art, including "multi-segmented" images that included (1) one or more

headers, tables, or other structures that contain information about the overall image

and/or its underlying data, and (2) one or more segments containing code or other

data used by the image, which the patent refers to as "data segments."[1] Ex-1101,

2:14-16, 4:34-42.

2. Loading and Executing Multi-Segmented Software Images

44. Before a processor can execute a multi-segmented software image, the

processor usually must load the image into its system memory, from where it is

then executed. Most multi-segmented executable software images are designed to

be loaded in multiple steps. In the first step, the processor reads information in the

one or more headers, tables, and/or other structures of the software image. That

---

[1] References to "data" include code and/or data, and references to "data segment"

include segments containing code and/or data.

information might identify the type of image (*e.g.,* an image format such as

Executable and Linking Format (ELF)), the size of the image, the number and size

of any data segments in the image, the storage locations of the data segments, and

the locations in system memory where the data segments are to be loaded for

execution. In one or more other steps, the processor uses that information to load

the data segments into memory and execute the image.

45.   When transferred into memory, the data segments of a software image

can be stored either in contiguous (*i.e.,* continuous) memory locations or spread

across non-contiguous (*i.e.,* non-continuous) memory locations. "Scatter loading"

is a well-known loading process in which one or more portions of a software image

are loaded (or "scattered") into memory. When there are multiple portions of a

software image, the portions are loaded across either contiguous or non-contiguous

locations in memory. Given this aspect of scatter loading, a mapping mechanism

is typically needed to allow a processor (or other component loading the code or

other data) to know the destination locations where the various parts of the image

are to be stored in memory. Many prior art executable software image formats

(such as ELF) were designed for scatter loading—by including information in the

image about where segments of the image should be loaded in memory for later

execution.

3. Sharing Memory in Multi-Processor Systems

46. In order to reduce costs and space requirements in a multi-processor system, such as a mobile telephone having a baseband/modem processor and an application processor, program code for both processors may be stored in a single non-volatile memory. For example, the application processor may have direct access to non-volatile memory that stores program code for both the application processor and the baseband/modem processor. The application processor may also have direct access to volatile memory for storing its program code after power up.

47. The baseband/modem processor, on the other hand, may have direct access to only volatile memory and not non-volatile memory. Upon power up, therefore, the application processor may have to transfer program code from non-volatile memory to volatile memory so that the baseband/modem processor can use it. The application processor can transfer the baseband/modem processor's program code from the non-volatile memory connected to the application processor to the baseband/modem processor, which may then store the program code in the volatile memory connected to the baseband/modem processor.

48. The transfer of the program code is typically performed by transferring segments of code or other data over a bus, where it is then loaded into memory. A large software image may be split into smaller data segments to facilitate its transfer. Each segment of data is typically transferred with a header.

Each segment of data is typically received by a processor and stored in memory temporarily while the processor reads the information in the header to determine where the data payload should be later stored in the same or different memory. Thereafter, the processor stores the data at the destination address in memory.

C. Boot Loading

49. When a computing device is first powered on, one or more processors in the device typically load and execute software (sometimes called "boot code" or "boot software") to enable the processor(s) to begin to operate. Because a processor must execute its boot code each time it powers up, the boot code is often stored in a non-volatile memory that is coupled to the processor. In this arrangement, during boot up, the boot code is typically loaded and executed from system memory that is coupled to the processor.

50. In a multi-processor system, each processor can store its own boot code, as the '949 patent acknowledges is prior art. Ex-1101, 1:38-44 ("Processors may require some software code, commonly referred to as boot code, to be executed for [booting] up. In a multi-processor system, each processor may require respective boot code for booting up. As an example, in a smartphone device that includes an application processor and a modem processor, each of the processors may have respective boot code for booting up."), 1:51-56 ("For instance, a processor's boot code may be stored to the processor's respective non-

volatile memory (e.g., Flash memory, read-only memory (ROM), etc.), and upon power-up the boot code software is loaded for execution by the processor from its respective non-volatile memory.").

51.   The boot-up of a processor often occurs in multiple stages. As the first step, a primitive "boot loader" function usually loads and then executes a relatively small amount of boot code stored in a local boot ROM that the processor can access easily. This first stage enables the processor to begin performing basic functions. In one or more later stages, the processor then typically loads additional boot code (usually stored in a different, larger non-volatile memory), which enables the processor to perform more sophisticated functions. .

52.   As the '949 patent acknowledges, in the prior art, it was known that boot code for a processor could be stored in a non-volatile memory coupled to a different processor (especially for the later-stage boot code, which was often too large to store in ROM). Ex-1101, 2:9-13 ("In this type of system, the software (e.g., boot image) is downloaded from the first processor to the other processor(s) (e.g., to volatile memory of the other processor(s)), and thereafter the receiving processor(s) boots with the downloaded image.").

VI.   OVERVIEW OF THE '949 PATENT

53.   The application that issued as the '949 patent (Ex-1101) was filed on Mar. 21, 2011. The '949 patent claims priority to four provisional applications, the earliest of which was filed on Mar. 22, 2010.[2]

54.   The '949 patent is directed to scatter loading an executable software image from a memory connected to a primary processor to a memory connected to a secondary processor. Ex-1101, 1:24-33.

A.   Alleged Problem of the Prior Art

55.   According to the '949 patent, however, prior art systems and methods for transferring software code between processors were inefficient. In particular, the '949 patent states that when retrieving an image for a modem processor from a non-volatile memory coupled to the application processor, prior art devices required copying the entire software image into one part of system memory coupled to the modem processor, and then copying the image into another part of system memory when loading it for execution:

> [T]raditional loading processes require an intermediate step where the
>
> binary multi-segmented image is buffered (e.g., transferred into the

---

[2] For purposes of this declaration, I treat Mar. 22, 2010 as the effective filing date, but do not take any position regarding whether the '949 patent is fully enabled by any of its provisional applications.

system memory) and then later scattered into target locations (e.g., by

a boot loader). Aspects of the present disclosure provide techniques

that alleviate the intermediate step of buffering required in traditional

loading processes.

Ex-1101, 7:20-26.

56.   The '949 patent describes this double copy (or "extra memory copy")

approach as inefficient. Ex-1101, 7:27-30 ("Thus, aspects of the present disclosure

avoid extra memory copy operations, thereby improving performance (e.g.,

reducing the time required to boot secondary processors in a multi-processor

system)."); *see also id.*, 2:1-54, 9:42-56, 11:11-24. However, this alleged problem

was well-known in the prior art.

B.   Purported Solution of the '949 Patent

57.   The '949 patent does not claim to invent a new type of processor, a

new type of processor architecture, a new type of executable software image

format, or a new type of image header or data segment. The patent also does not

claim to invent the well-known idea of scatter loading executable software images

into system memory, including based on information contained in an image header.

All those things were well known in the prior art.

58.   Indeed, the '949 patent admits that many claimed features of the

patent are in the prior art, including:

- multi-processor systems in which a first processor uses non-volatile memory to store a software image (*e.g.,* boot code) for a second processor, and where the software image is downloaded from the first processor to the second processor (*e.g.,* to a volatile memory at the second processor) (Ex-1101, 2:1-13);

- that a software image would often comprise a header and multiple segments of code (*id.*, 2:14-16);

- that a transfer of a software image from a first ("primary") processor to a second ("secondary") processor may occur via a temporary buffer (also referred to as an intermediate buffer) (*id.*, 2:17-37);

- that a software image could be scattered (*i.e.*, scatter loaded) from a temporary buffer into the memory (*e.g.,* volatile memory) of a secondary processor (*id.*, 2:35-41);

- that the primary processor and its non-volatile memory may be implemented on a different chip from that of the secondary processor (*id.*, 2:42-45); and

- that each processor can have a non-volatile memory (*e.g.,* flash memory, ROM) that store executable images and file systems, including the processor's boot code such that upon power-up, the boot code is loaded from memory for execution by that processor (*id.*, 1:48-56).

59.   Instead, what the '949 patent claims to have invented is a new way to

avoid the "double copy" or "extra memory" approach described above. The

purported solution of the '949 patent is for a secondary (or modem) processor to

(1) first receive from the primary (or application) processor the "image header" of

an executable software image, and (2) then *separately* receive each "data segment"

of the image, each of which is then scatter loaded into the secondary processor's

system memory using the data segment's destination address from the earlier-

received image header—***all without first copying the entire image into the***

secondary processor's system memory. The data received by the secondary (or

modem) processor is temporarily stored in a hardware buffer separate from the

secondary processor's system memory. Ex-1101, 2:58-3:67, 9:42-56, 11:11-24.

However, this purported solution was well-known in the prior art.

60.   To illustrate this concept, Figure 3 of the '949 patent shows a primary

processor 301 (*e.g.,* an application processor) connected to a secondary processor

302 (*e.g.,* a baseband/modem processor) via an inter-chip communication bus 310

(*e.g.,* a High-Speed USB (HS-USB) cable).
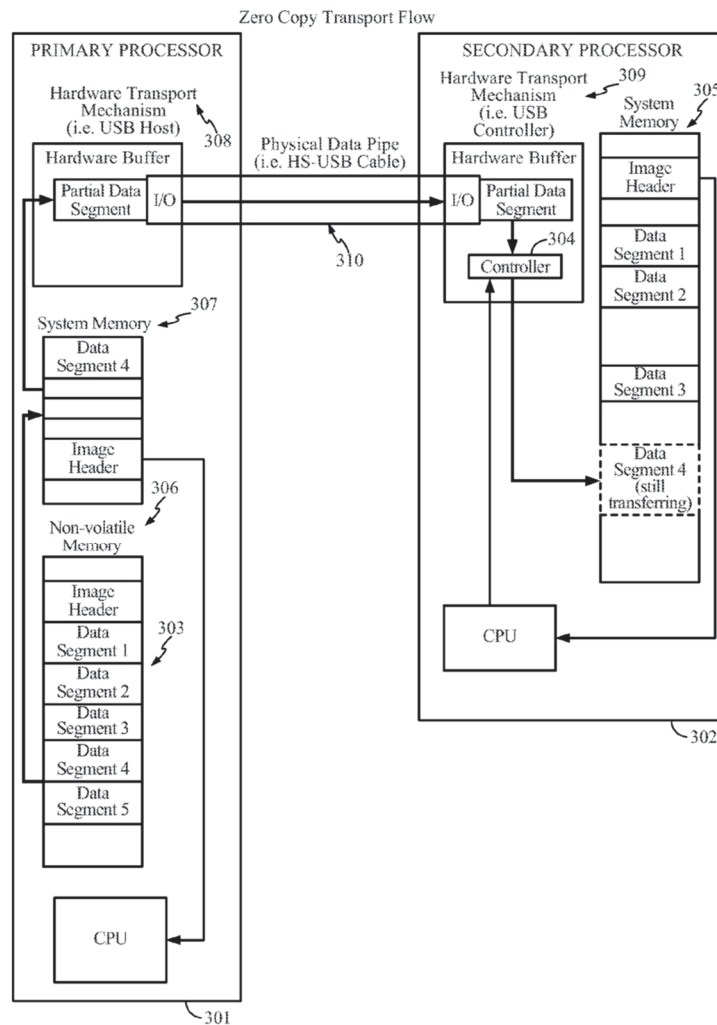
FIG. 3

Ex-1101, Fig. 3.

61.   The primary processor 301 comprises a non-volatile memory 306 storing a software image 303 for the secondary processor 302. Ex-1101, 7:60-8:18, Fig. 3. The software image 303 may be a multi-segment image that includes an image header and at least one data segment. *Id.,* 8:2-5, Fig. 3. The primary processor 301 also comprises a system (*i.e.*, volatile) memory 307 and a hardware transport mechanism 308 (*e.g.*, a USB host). *Id.*, 8:11-26, Fig. 3.

62.   The secondary processor 302 comprises a system (*i.e.*, volatile) memory 305 and a hardware transport mechanism 309 (*e.g.,* a USB controller). Ex-1101, 8:9-30, Fig. 3. The hardware transport mechanism 309 includes a scatter loader controller 304 for controlling the scatter loading of software image data segments received from the primary processor 301 into the system memory 305 of the secondary processor 302. *Id.,* 8:60-62, 9:21-27, Fig. 3.

63.   The '949 patent describes the scatter loading process in two stages. In the first stage, the image header of the software image 303 is transferred from the primary processor 301 to the secondary processor 302. Ex-1101, 8:9-12, Fig. 3. The image header of the software image 303 includes information that is used to determine where each data segment is to be loaded in the system (*i.e.*, volatile) memory 305 of the secondary processor 302. *Id.,* 8:18-21, Fig. 3. This information generally includes a list of addresses indicating where each data segment is to be loaded. *Id.*, 8:57-60, Fig. 3.

64.   The first stage of the scatter loading process begins by the secondary processor 302 requesting the image header from the primary processor 301 in accordance with a boot loader program running on the secondary processor 302. Ex-1101, 8:40-43, Fig. 3. Once requested, the primary processor 301 transfers the image header from the non-volatile memory 306 to the primary processor's system (*i.e.*, volatile) memory 307. *Id.,* 8:14-18, Fig. 3.

65.   Within the primary processor 301, the image header is transferred

from the system (*i.e.*, volatile) memory 307 to the hardware transport mechanism

308. Ex-1101, 8:24-26, Fig. 3. The image header is then transferred from the

hardware transport mechanism 308, over the inter-chip communication bus 310, to

the hardware transport mechanism 309 of the secondary processor 302. *Id.*, 8:9-

35, Fig. 3.

66.   In the second stage of the scatter loading process, the boot loader

program running on the secondary processor 302 programs the scatter loader

controller 304 with the information included in the image header. Ex-1101, 8:63-

67, Fig. 3. As discussed above, this information is used to determine where each

data segment is to be loaded in the secondary processor's system memory 305, as

well as the size of each data segment and its location within the primary processor

301. Thus, once programmed, the scatter loader controller 304 may transfer to the

system memory each data segment. *Id.,* 9:13-37, Fig. 3. With scatter loading, the

data segments may be placed into consecutive or non-consecutive locations within

the system memory. *Id.*, 9:12-15, 9:21-41. The primary processor 301 may

transfer each data segment to the secondary processor 302 over the inter-chip

communication bus 310. *Id.*, 8:26-30, Fig. 3.

67.   Once received at the secondary processor 302, each data segment is

temporarily stored not in system memory but in a hardware buffer 309. Ex-1101,

9:50-54, Fig. 3. Thereafter, the scatter loader controller 304 may load each data segment directly into the system (*i.e.*, volatile) memory 305 of the secondary processor 302 at the location determined from the information included in the image header and previously programmed into the scatter loader controller 304. Ex-1101, 9:21-37, Fig. 3. Thus, during the above-described scatter loading process, each data segment is transferred from the primary processor 301 to the secondary processor 302, and is not temporarily stored in the secondary processor's system (*i.e.*, volatile) memory 305. *Id.,* 9:42-56.

C. Prosecution History of the '949 Patent

68. The '949 patent was filed on Mar. 21, 2011 with twenty-four claims, including six independent claims. During prosecution, the Applicants amended several independent claims to incorporate the contents of cancelled claim 4, in addition to other features, to overcome the cited prior art.

69. Initially, the Examiner rejected all original claims of the '949 patent under 35 U.S.C. § 102(b) as being anticipated by PCT Publication No. WO 2006/077068 to Svensson et al. ("Svensson PCT").[3] Ex-1104, 2-5. The Examiner found that Svensson PCT discloses:

---

[3] Svensson PCT claims priority to Svensson. Svensson PCT at cover (Ex-1103); Svensson at cover (Ex-1110).

- a "secondary processor [*client processor 104*] comprising system memory [*DSPXRAM 110*] and a hardware buffer [*An intermediate storage area is defined within the memory 108*]";

- a "scatter loader controller…[*The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading, see page 7, last line - page 8, line 2*]";

- a "primary processor [*host processor102*] coupled with a memory [*non-volatile memory 106*]";

- an "interface…[*The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths…*]"; and

- "the executable software image comprises an image header and at least one data segment [*Fig. 3, it is clear that the executable software image comprises an image header and at least one segment*]."

Id., 2-3 (emphasis in original); *see also* Ex-1103, Fig. 1 (below).
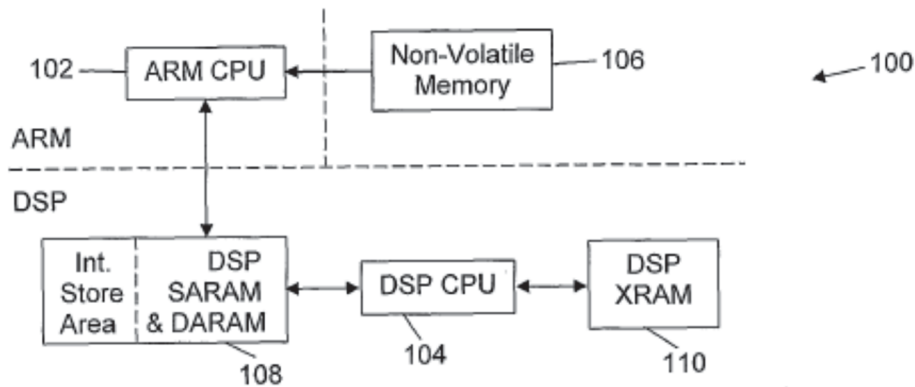
FIG. 1

70. In response, the Applicants did not contest that Svensson PCT anticipated the original claims. Instead, the Applicants amended claim 1 to require that (1) the claimed "hardware buffer" must receive "an image header and at least one data segment" of an executable software image, "the image header and each data segment being received separately"; and (2) the claimed "scatter loader controller" is configured "to load the image header; and to scatter load each received data segment, based at least in part on the loaded image header." Ex-1105, 2-9. Similar amendments were made to independent claims 11, 17, 19, 21, and 23. *Id.*, 4-7. In connection with these amendments, the Applicants admitted that "Svensson [PCT] arguably discloses that the software includes a header and a data segment." *Id.*, 8.

71. In an attempt to distinguish Svensson PCT, the Applicants argued that Svensson PCT "fails to disclose that the image header and each data segment are received separately" (*i.e.*, the requirement added by amendment). Ex-1105, 9. The

Applicants further asserted that "loading *each data segment directly*[4] from the hardware buffer to the system memory," as required by Applicants' amendment, "is patentably distinguishable from concatenating the data blocks and headers in the intermediate storage area and then transferring the concatenated data to the memory, as recited in Svensson [PCT]." *Id.*

72.   Following an additional rejection on lack of enablement grounds, the Examiner allowed the claims. Ex-1106, 5. Thus, the Examiner allowed the application only after the Applicants amended the claims to require that (1) the image header and each data segment be received *separately* at the secondary processor, as well as (2) each data segment be scatter loaded *directly* to the system memory of the secondary processor. *Id.* The '949 patent then issued on Sep. 16, 2014. Ex-1101, cover.[5]

73.   I explain below how Bauer and Kim disclose the same two claim features that the Examiner found allowable over Svensson PCT. These new references—in combination with Svensson, the U.S. counterpart to Svensson

---

[4] All emphasis added unless otherwise noted.

[5] The claims were re-numbered such that original claims 5-24 became issued claims 4-23, respectively.

PCT—present new art and a new combination that the Examiner never had a chance to consider.

VII. LEVEL OF ORDINARY SKILL IN THE ART

74.   A person of ordinary skill in the art of the '949 patent would have had a Master's degree in Electrical Engineering, Computer Engineering or Computer Science plus at least two years of experience in mobile device architecture and multi-processor systems, or a Bachelor's degree in one of those fields plus at least four years of experience in mobile device architecture and multi-processor systems. I understand that in the Related ITC Case, the CALJ held this to be the level of ordinary skill in the art. Ex-1107, 11-13.

VIII. CLAIM CONSTRUCTION

75.   I have applied the "broadest reasonable interpretation" standard for the claim terms of the challenged claims. However, based on my reading of the '949 patent's specification and the ordinary meanings of the claim terms, the prior art teaches each claim limitation under any reasonable interpretation of the claim terms. My analysis is, therefore, not dependent on application of the "broadest reasonable interpretation" standard.

76.   I understand that the Petitioner has set forth its proposed construction of a term of the '949 patent and its support for the construction. Those are copied below. I also understand that the remaining terms of the claims described below

are readily understandable and I have given them their broadest reasonable interpretation in light of the specification as commonly understood by those of ordinary skill in the art.

A. "image header" (claims 10 and 16)

77. As used in the '949 patent, a person of ordinary skill would have understood the term "image header" to mean "a header associated with the entire image that specifies where the data segments are to be placed in the system memory" under either the BRI or *Phillips* standard. This understanding is consistent with the specification of the '949 patent. *See* Ex-1101, 8:18-21 ("The image header includes information used to identify where the modem image executable data is to be eventually placed into the system memory of the secondary processor 305."), 7:50-52 ("The image header also specifies the destination address of the image in target memory."), 9:23-24 ("That image header provides information as to where the data segments are to be located in the system memory 305."), 10:6 ("... the header associated with the entire image."). This understanding is also consistent with the claims of the '949 patent. For example, independent claim 10 recites "processing…the image header to determine at least one location within system memory…to store each data segment." *Id.*, claim 10. In the Related ITC Case, the parties (including the Patent Owner) agreed to this construction for this term. Ex-1108, 3.

B. Means-Plus-Function Terms (Claim 16)

78. The limitations of claim 16—(1) "means for receiving at a secondary processor ... an image header ..."; (2) "means for processing ..."; (3) "means for receiving at the secondary processor ... each data segment ..."; and (4) "means for scatter loading ..."—each uses the term "means" followed by a function without reciting sufficient structure for performing that function. Ex-1101, claim 16. Accordingly, I have interpreted each of these limitations as a means-plus-function limitations.

C. "means for receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor" (claim 16)

79. I understand that this is a means-plus-function limitation, which requires an identification of the claimed function and the structure in the specification that performs the claimed function.

1. Function

80. As recited in claim 16, the function performed by this claim element is "receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor" under either the BRI or *Phillips* standard. This is in accordance with the determination of the CALJ in the Related ITC Case. Ex-1107, 17.

2. Structure

81. The corresponding structure for performing the above-stated function is "a secondary processor (*e.g.,* 110, 210, 302) connected to a primary processor (*e.g.,* 104, 204, 301) via an inter-chip communication bus (*e.g.,* 134, 234, 310) for a USB-based High Speed Inter-Chip (HSIC) bus, a MIPI High Speed Synchronous Interface (HSI) bus, a Secure Digital I/O Interface (SDIO) bus, a Universal Asynchronous Receiver/Transmitter (UART) bus, a Serial Peripheral Interface (SPI) bus, or an Inter-Integrated Circuit (I2C) bus, and equivalents thereof" under either the BRI or *Phillips* standard, as described in the '949 patent at Ex-1101, 5:35-43 and shown in Fig. 3. This is in accordance with the determination of the CALJ in the Related ITC Case. Ex-1107, 17-18.

D. "means for processing, by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment" (claim 16)

1. Function

82. As recited in claim 16, the function associated with this means-plus-function limitation is "processing, by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment" under either the BRI or *Phillips* standard. In the Related ITC Case, the parties (including the Patent Owner)

proposed this construction as the corresponding function for this term.[6] Ex-1107,

4-5.

2.   Structure

83.   The purported "structure" identified in the specification that relates to

the claimed function is "a modem processor coupled to a system memory," as

described in the '949 patent at Ex-1101, 3:9-12, 4:58-5:43, 5:59-6:39, 7:60-10:44,

8:50-56, 9:27-41, and shown in Figs. 1-3.[7] In the Related ITC Case, the Patent

Owner proposed this construction as the corresponding structure for this term.[8]

Ex-1108, 4-5. Therefore, for purposes of the Petition, the structure for this mean-

plus-function term is "a modem processor coupled to a system memory, and

equivalents thereof" under either the BRI or *Phillips* standard.

E.   "means for receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment" (claim 16)

1.   Function

84.   As recited in claim 16, the function associated with this means-plus-

function limitation is "receiving at the secondary processor, from the primary

---

[6] The CALJ did not construe this term.

[7] I do not concede that this structure is sufficient and reserves the right to advance a

different argument regarding this claim term in litigation.

[8] The CALJ did not construe this term.

processor via the inter-chip communication bus, each data segment" under either

the BRI or *Phillips* standard. This is in accordance with the determination of the

CALJ in the Related ITC Case. Ex-1107, 19.

2. Structure

85. The corresponding structure for performing the above-stated function

is "a secondary processor (*e.g.,* 110, 210, 302) connected to a primary processor

(*e.g.,* 104, 204, 301) via an inter-chip communication bus (*e.g.,* 134, 234, 310) for

a USB-based High Speed Inter-Chip (HSIC) bus, a MIPI High Speed Synchronous

Interface (HSI) bus, a Secure Digital I/O Interface (SDIO) bus, a Universal

Asynchronous Receiver/Transmitter (UART) bus, a Serial Peripheral Interface

(SPI) bus, or an Inter-Integrated Circuit (I2C) bus, and equivalents thereof" under

either the BRI or *Phillips* standard, as described in the '949 patent at Ex-1101,

5:35-43 and shown in Fig. 3. This is in accordance with the determination of the

CALJ in the Related ITC Case. Ex-1107, 19.

F. **"means for scatter loading, by the secondary processor, each data segment directly to the determined at least one location within the system memory, and each data segment being scatter loaded based at least in part on the processed image header" (claim 16)**

1. Function

86. As recited in claim 16, the function associated with this means-plus-

function limitation is "scatter loading, by the secondary processor, each data

segment directly to the determined at least one location within the system memory,

and each data segment being scatter loaded based at least in part on the processed image header" under either the BRI or *Phillips* standard. In the Related ITC Case, the parties (including the Patent Owner) proposed this construction as the corresponding function for this term.[9] Ex-1108, 6.

2. Structure

87. The purported "structure" identified in the specification that relates to the claimed function is "a modem processor coupled to a system memory," as described in the '949 patent at Ex. 1101, Abstract, 1:24-33, 4:10-15, 4:58-5:43, 5:59-6:39, 7:60-10:44, 8:21-30, 8:62-67, 9:3-8, 9:16-56, 10:13-18 and 10:27-32, and as shown in Figs. 1-3.[10] In the Related ITC Case, the Patent Owner proposed this construction as the corresponding structure for this term.[11] Ex-1108, 6. Therefore, for purposes of this Petition, the structure for this mean-plus-function term is "a modem processor coupled to a system memory, and equivalents thereof" under either the BRI or *Phillips* standard.

_____

[9] The CALJ did not construe this term.

[10] I do not concede that this structure is sufficient and reserves the right to advance a different argument regarding this claim term in litigation.

[11] The CALJ did not construe this term.

IX. OVERVIEW OF PRINCIPAL PRIOR ART REFERENCES

A. Svensson (Ex-1110)

88. Svensson is generally directed to an "OS-friendly bootloader." Ex-1110, Abstract. In particular, Svensson discloses a technique for loading code and/or data from memory associated with a host processor to memory associated with a client processor in a multi-processor system. *Id.*, 2:24-27.

89. Figure 1 of Svensson below shows a multi-processor system 100. Ex-1110, 3:49-50, Fig. 1.
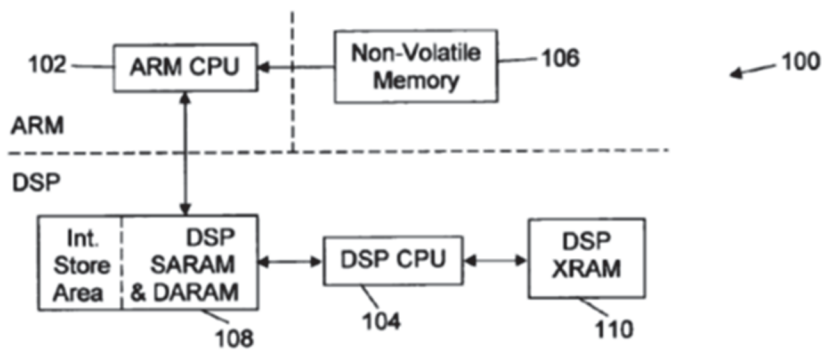
FIG. 1

Multi-processor system 100 has an advanced RISC machine (ARM) device and a digital signal processor (DSP) device. *Id.*, 3:54-58, Fig. 1. The ARM device includes a host processor (ARM CPU) 102 that is coupled to a non-volatile memory 106 and to the DSP device. *Id.*, 3:49-63, 4:3-5, Fig. 1. The DSP device includes a client processor (DSP CPU) 104, a system memory (DSP external RAM (XRAM)) 110, and an internal volatile memory (single-access RAM (SARAM) or dual-access RAM (DARAM)) 108 that can have an intermediate storage area (Int.

Store Area). *Id.*, 3:64-4:3, Fig. 1. The multi-processor system includes access

paths, such as buses and DMA paths, that connect the CPUs and memories. *Id.*,

4:3-5, Fig. 1. The host processor 102 can directly access the non-volatile memory

106 and the shared volatile memory 108 (including the intermediate storage area)

at the client, but not the system memory 110 at the client processor 104. *Id.*, 4:5-8,

Fig. 1. The client processor 104, on the other hand, can directly access the shared

volatile memory 108 as well as the system memory 110, but not the non-volatile

memory 106 coupled to the host. *Id.*

90. Svensson discloses that the host processor 102 loads the code and/or

data from the non-volatile memory 106 to the intermediate storage area in the

shared volatile memory 108 at the client, and the client processor 104 then copies

the code and/or data to end destinations in the system memory 110. Ex-1110,

1:11-15, 2:6-20, 4:9-14, 4:22-26, 6:12-15, Fig. 1; *see also id.*, 4:15-6:11, Fig. 2.

Svensson describes the intermediate storage area as "a block of [reserved] memory

in the slave's heap of memory that is located in the memory visible to the host,

such as 'internal memory 108'" and is "used for intermediate storage of

information (code and/or data) to be transferred to the slave-private memory, i.e.,

the memory that is invisible to the host, such as 'external' XRAM 110." *Id.*, 5:21-

36, Fig. 1.

91.   Figure 3 of Svensson below shows how the code and/or data can be transferred from the host processor into the intermediate storage area using one or more transfer blocks. *Id.*, 6:12-23, Fig. 3.
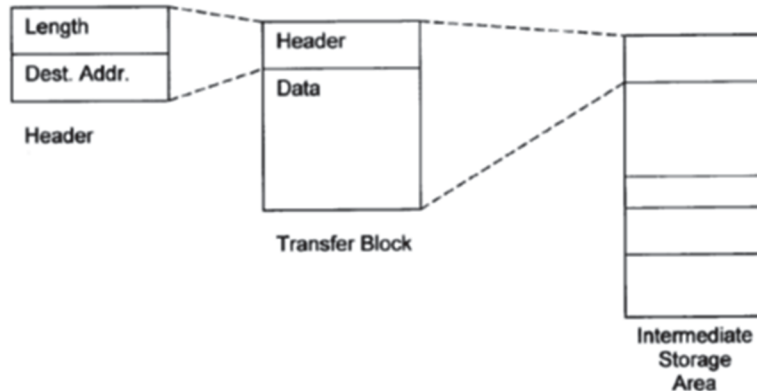
FIG. 3

Each transfer block includes a header that indicates the length (Length) of the block and a destination address (Dest. Addr.) indicating where the block is to be loaded in the system memory 110. *Id.* As shown by the dashed lines in Figure 3, several such blocks may be concatenated in the intermediate storage area. *Id.*, 6:23-25, Fig. 3. Svensson further discloses that the size of the code and/or data can be larger than the intermediate storage area. *Id.*, 6:26-28, 6:37-43.

92.   Svensson discloses that the client receives code and/or data in one or more transfer blocks from the host processor 102. Ex-1110, 6:12-25, 6:60-7:2, Fig. 1. For each transfer block, the client processor 104 first reads the header, which includes the destination address for the data block. *Id.*, 5:65-67, 6:15-23, Fig. 2. The client processor then uses the destination address to load the data block into

the system memory 110. *Id.* Svensson discloses that the client processor 104 can have a communication mechanism such as a DMA unit "to perform the actual transfers of information between memories." *Id.,* 6:48-58, 6:60-7:2, 7:52-60.

B. Bauer (Ex-1109)

93. Bauer is closely interrelated with Svensson. Bauer has the same four inventors and the same assignee as Svensson, and was filed only four months after Svensson. Ex-1109, cover; Ex-1110, cover.

94. Bauer describes a new file format that is an improvement upon the file format described in Svensson. Instead of using a separate block header to store a destination address for each block of an image, the file format in Bauer has that information collected in one place as section information. Ex-1109, ¶¶27, 32-34, 43, Figs. 1A-1C. Bauer describes the benefit of this approach: "[t]his simplifies optimization in a number of circumstances, for instance, if sections are to be loaded into memory" and "makes memory loading efficient as there is no need to search through an image for section headers when loading." *Id.*, ¶43; *see also id.*, ¶16 ("Greater efficiency in loading data can reduce response times in such systems, and space-efficient storage saves valuable memory."), ¶27. This section information is near the beginning of the image—after the header but before all the data sections of an image—so that the section information can be retrieved

separately before the data sections are read and processed. *Id.*, Abstract, ¶¶28-30, 47, Figs. 1A-1C.

95. Bauer teaches that this new file format can be used in the same multi-processor system described in Svensson. Bauer shows and describes the same system as Svensson and even cites to Svensson as an example of a program loader for loading an image with this file format in that same multi-processor system.

96. In particular, Bauer discloses the same circuit architecture as Svensson—Bauer and Svensson disclose loading different image file formats from an ARM device to a DSP device's system memory in the same multi-processor system. *See* Ex-1109, Fig. 2, ¶¶35, 36; *cf.* Ex-1110, Fig. 1, 3:49-4:8.
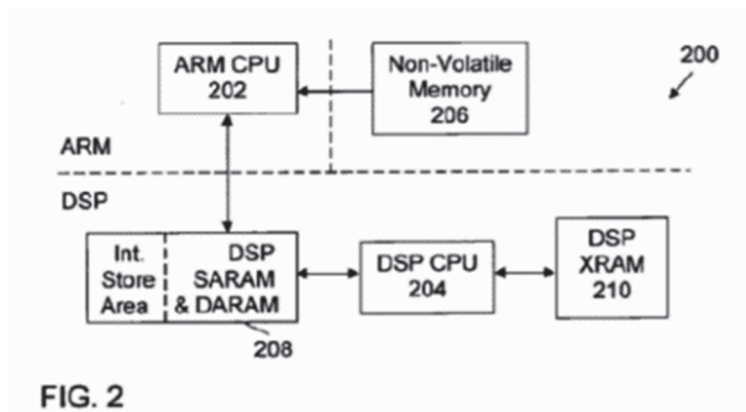
FIG. 2

Bauer, Fig. 2
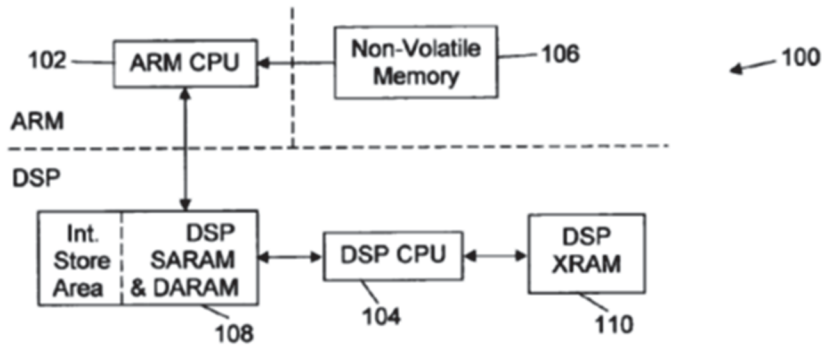
FIG. 1

Svensson, Fig. 1

97. Bauer, in its detailed description of the specification, cites to Svensson as one example of a program loader that can use the invention described in Bauer:

> There are many possible applications of this format and its individually coded sections….Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly. One example of such a program loader is described in U.S. patent application Ser. No. 11/040,798 filed on Jan. 22, 2005, by M. Svensson et al. for "Operating-System-Friendly Bootloader."

Ex-1109, ¶31.

98. Figures 1A-1C of Bauer below show the file format of the binary data image 100:

FIG. 1A	Header 102	Section Information 104	Section Data 106

100

FIG. 1B	Size [32 bits] 102-1	Number of sections [16 bits] 102-2

102

Section 1 Length [16 bits] 108-1	Extra 1 [16 bits] 112-1	Section 2 Length [16 bits] 108-2	Extra 2 [16 bits] 112-2
Load Address 1 [32 bits] 110-1		Load Address 2 [32 bits] 110-2	

104-1   104-2

FIG. 1C
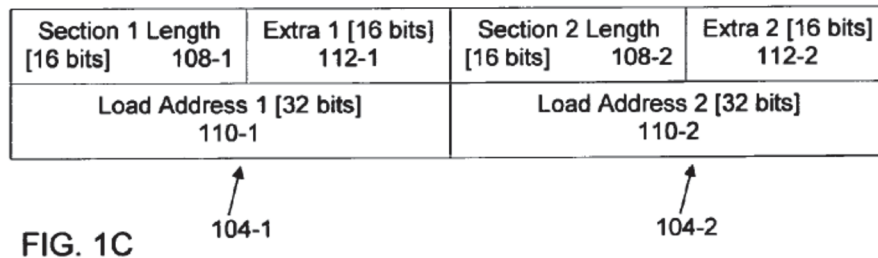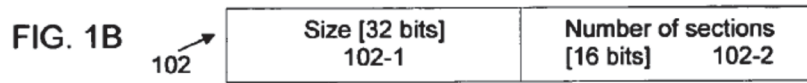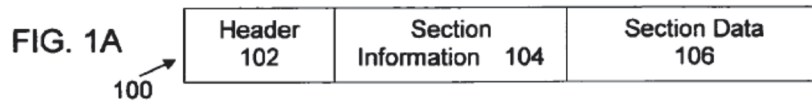
Ex-1109, Figs. 1A-1C.

99.   As shown in Figure 1A, the file format includes a header 102, section

information 104, and section data 106. Ex-1109, ¶32, Fig. 1A. As shown in

Figure 1B, the header 102 indicates the size 102-1 and number of sections 102-2 in

the section data 106. *Id.*, ¶33, Fig. 1B. As shown in Figure 1C, the section

information 104 indicates the length 108 and load (or destination) address 110 of

each section in the section data 106. *Id.*, ¶34, Fig. 1C. Each data section has its

own load address in the section information and can be arranged in the image in

any suitable order (*e.g.,* in order of load address or in an arbitrary order). *Id.*, ¶37.

The section data 106 includes one or more sections containing object code and data

for the image. *Id.*, ¶¶32-34, Figs. 1A-1C.

100. Bauer discloses that the file format shown in Figures 1A-1C can be stored in any of the memories of multi-processor system 200, including the non-volatile memory 206 at the host, the shared volatile memory 208 that has an intermediate storage area (Int. Store Area), and the system memory 210. Ex-1109, ¶35 ("The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored."), ¶36 ("…between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C."), Fig. 2. Bauer also discloses that the client processor can have an operating system memory manager for "load[ing] and unload[ing] sections of memory" or a DMA unit for "DMA transfer[s]." *Id.*, ¶¶31, 38.

C. Kim (Ex-1111) (Including English Translation (Ex-1112))

101. Kim discloses a multi-processor system in which a secondary processor receives program block header information separately from a primary processor before receiving a corresponding program block from the primary processor during a loading procedure. Ex-1112, 5:12-6:5. Figure 1 of Kim below shows a primary processor (system management processor 11) that is coupled to a plurality of secondary processors (main processors 13, switching subsystem processors 14, device processors 15), as well as to a non-volatile memory (hard disk drive 17). *Id.*, 4:7-14, Fig. 1.

FIG. 1
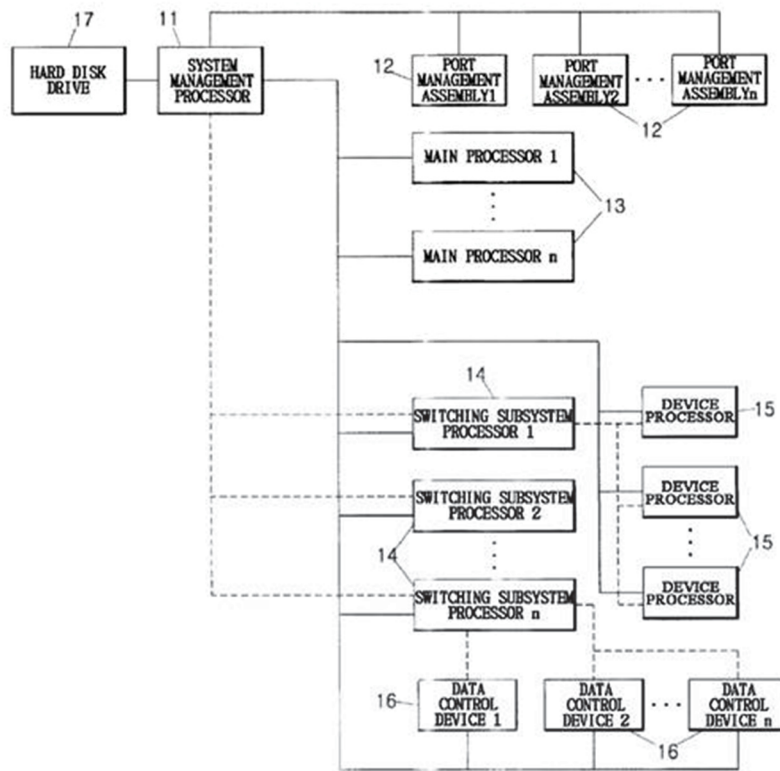
102. Figure 3 of Kim below shows a "conventional loading method" for loading a program stored in the non-volatile memory to one of the plurality of secondary processors. Ex-1112, 5:9-6:7, Fig. 3.

FIG. 3
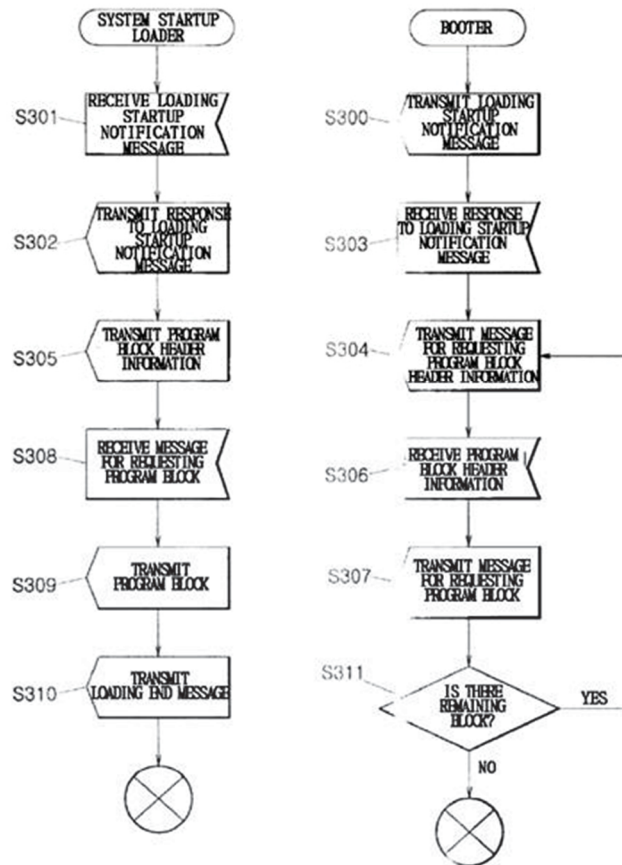
103. As part of the loading procedure, the secondary processor first requests the ***program block header information*** (step 304) from the primary processor. Ex-1112, 5:16-19, Fig. 3. The program block header information is then sent (steps S305 and S306) from the primary processor to the secondary processor. *Id.*, 5:19-21, Fig. 3 Once the secondary processor receives the program block header information, it then requests a ***program block corresponding to actual program content*** (step S307) from the primary processor. *Id.*, 5:21-23, Fig. 3. The program block is then sent from the primary processor to the secondary

processor (step S309). Ex-1112, 5:25-6:2, Fig. 3. In this way, the multi-processor system of Kim provides a system in which the program block header information is transmitted and *received separately* from the program block.

D.   Zhao (Ex-1113)

104.   Zhao discloses a "mobile computing device" that can be, for example, a smartphone or laptop computer. Ex-1113, ¶26. As shown in Figure 3 of Zhao below, the mobile computing device 100 "may comprise a dual processor architecture including a host processor 102 and a radio processor 104" that can "communicate with each other using interfaces 106." *Id.*, ¶32, Fig. 3.
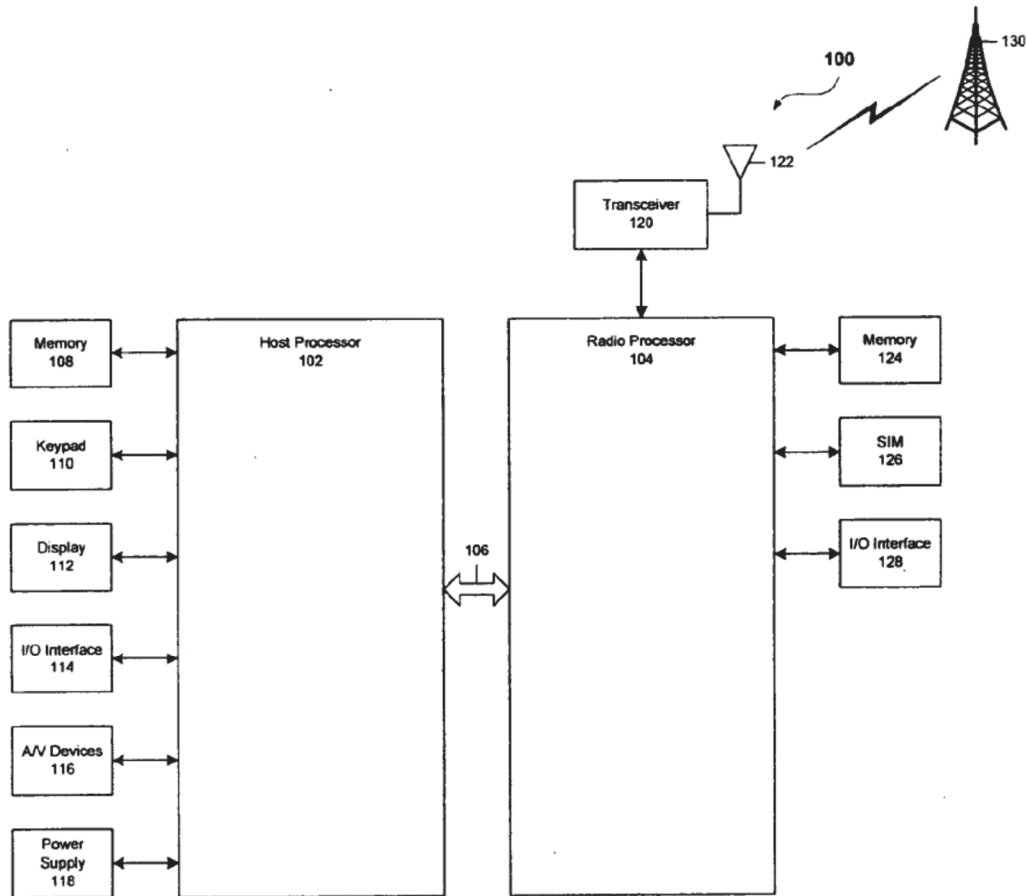
FIG. 3

105.   Radio processor 104 "may be implemented as a communications processor using any suitable processor or logic device, such as a modem processor or baseband processor." *Id.*, ¶44. Interfaces 106 can include "one or more universal serial bus (USB) interfaces, micro-USB interfaces, universal asynchronous receiver-transmitter (UART) interfaces, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth." *Id.*, ¶32, Fig. 5.

X. SPECIFIC GROUNDS FOR CHALLENGE

106. In the following sections I describe in detail how the prior art renders obvious each and every limitation of claims 10-17 of the '949 patent.

A. Ground 1: Claims 10-15 Are Rendered Obvious By The Combination Of Bauer, Svensson, And Kim

1. Reference to "Bauer and Svensson Combined"

107. Bauer is so closely interrelated with Svensson that, for ease of reference, the declaration uses "Bauer and Svensson combined" to illustrate what Bauer alone, or Bauer in combination with Svensson, teaches to a person of ordinary skill in the art.

108. Bauer has the same four inventors and the same assignee as Svensson, and was filed only four months after Svensson. Ex-1109, cover; Ex-1110, cover. Bauer describes a file format where section information containing destination addresses for each data section in an image is collected in one place and precedes all the data sections, and explicitly describes that format as an improvement over using multiple, separate block headers as described in Svensson. Ex-1109, ¶¶27, 32-34, 43, Figs. 1A-1C. *See* Section IX.B;

109. Bauer teaches that its file format can be used in the same multi-processor system as Svensson. Indeed, they have the exact same figures of that system. *See* Ex-1109, Fig. 2 (below), ¶¶35, 36; *cf.* Ex-1110, Fig. 1 (below), 3:49-4:8.
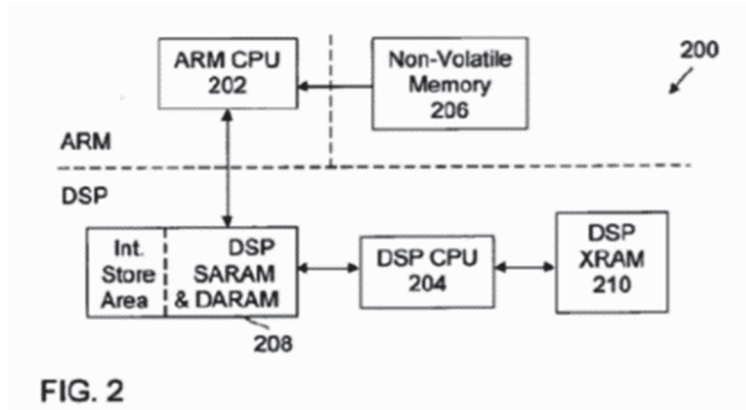
Bauer, Fig. 2
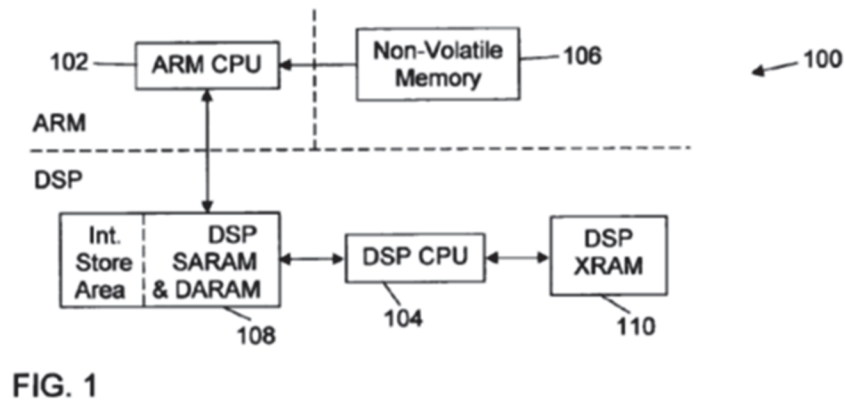
Svensson, Fig. 1

110.   In addition, Bauer explicitly cites to Svensson as an example for loading an image with Bauer's file format in that same multi-processor system. Ex-1109, ¶31. *See* Section IX.B;

111.   For these same reasons, as explained in more detail below, it would have been obvious to a person of ordinary skill in the art to combine Svensson with Bauer.

112.   Although it is readily apparent that the file format of Bauer can be used in the multiprocessor system of Svensson, Bauer does not describe the

multiprocessor system with the same level of detail as Svensson. Because of that,

this declaration refers to "Bauer and Svensson combined," but is clear in

identifying the relevant disclosures from each reference in its citations.

2. Claim 10

113. Claim 10 is rendered obvious by Bauer and Svensson combined;

however, to the extent the Patent Owner contends that Bauer and Svensson

combined does not teach that different parts of an executable software image are

received separately (claim [10b]), this claim is rendered obvious by the

combination of Bauer, Svensson, and Kim.

> **a. [10a] "A method comprising: receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor, the executable software image comprising the image header and at least one data segment;"**

114. Bauer and Svensson combined renders obvious this limitation. More

particularly, Bauer and Svensson combined discloses most of this limitation,

except that it does not explicitly disclose a header that meets the claimed "image

header." However, Bauer and Svensson combined renders obvious the claimed

"image header."

> **(i) "a primary processor," "memory coupled to the primary processor," "a secondary processor," "an inter-chip communication bus"**

115. Bauer and Svensson combined discloses these claim elements.[12] For example, Figure 2 of Bauer below shows a multi-processor system 200 that includes (1) a ***primary processor*** (advanced RISC machine (ARM) device), (2) a ***memory*** (non-volatile memory 206) coupled to the primary processor, (3) a ***secondary processor*** (digital signal processor (DSP) device), and (4) a ***bus*** ("access paths, e.g., buses and direct memory access (DMA) paths" as indicated by the arrows) communicatively coupling the primary processor and the secondary processor. Ex-1109, ¶¶35-36, Fig. 2 (below); *see also* Ex-1110, 3:49-63, 4:3-8, Fig. 1.
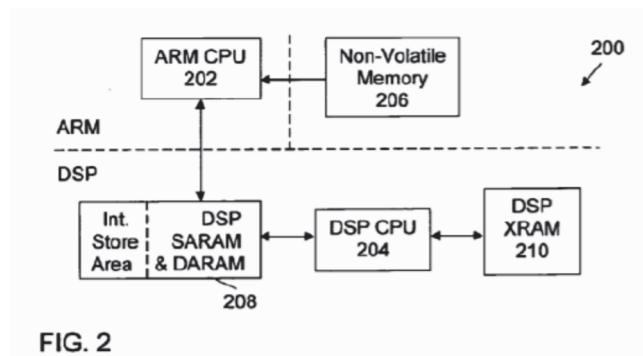
FIG. 2

A person of ordinary skill in the art would understand from Bauer and Svensson combined that the bus allows for the transfer (or communication) of data between

---

[12] The '949 patent admits that a multi-processor system having a primary processor, a non-volatile memory coupled to the primary processor, and a secondary processor, where the processors are on separate chips, is prior art. Ex-1101, 1:41-51, 2:42-45.

the primary processor and the secondary processor, and thus is a "***communication bus***."

116.   Bauer and Svensson combined further teaches that this communication bus is an "inter-chip" communication bus. A person of ordinary skill in the art would understand that "inter-chip" means between two chips, and therefore an "inter-chip communication bus" means a communication bus coupling two chips—which, in the context of claim 10, refers to two "processor" chips.

117.   Bauer and Svensson combined discloses that the primary and secondary processors are located on different chips, and therefore the "communication bus" coupling these two processors is an "inter-chip" communication bus. For example, Bauer and Svensson combined teaches that the two processors are part of two different devices—an ARM device and DSP device—and that there is a "hardware boundary" between the two processors. Ex-1109, ¶35, Fig. 2; Ex-1110, 3:54-60, Fig. 1. Bauer and Svensson combined further notes that the secondary processor is a "commercially available DSP device" that includes "on-chip memories." Ex-1109, ¶36; Ex-1110, 3:64-4:1. A person of ordinary skill in the art would understand that a commercially available DSP device having on-chip memories would comprise a separate chip to the ARM device.

118.   To the extent that the Patent Owner argues that Bauer and Svensson combined does not disclose processors located on different chips, then this feature is rendered obvious by Bauer and Svensson combined. In light of Bauer and Svensson's combined teaching that the primary and secondary processors are devices separated by a hardware boundary and that the secondary processor is a commercially available DSP device having on-chip memories, it would be obvious to a person of ordinary skill in the art to provide the primary and secondary processors on different chips. A person of ordinary skill in the art would be very familiar with multi-processor systems comprising processors on separate chips. *See e.g.,* Ex-1113, ¶¶32, 34, 44, Fig. 3. Indeed, the '949 patent admits that systems in which the primary and secondary processors were on the same or different chips were well known. Ex-1101, 2:42-45.

119.   A person of ordinary skill in the art would have considered different ways to construct the two processors in Bauer and Svensson combined. Such a person would have understood that there are a finite number of ways to do so—on the same chip or on different chips. The person of ordinary skill in the art would have found that this simply involves the combination of prior art elements (processor chips and standard methods of packing those chips in devices) according to known methods to achieve a predictable result with a reasonable expectation of success. Such a person would have understood that this approach

would have been obvious to try, would have been one of a small number of finite ways that provides greater efficiency, and would have involved a simple substitution of one known feature for another.

120. Accordingly, a person of ordinary skill in the art would understand that because Bauer and Svensson combined discloses, or alternatively renders obvious, a communication bus that couples two processors located on separate chips, that communication bus is an "inter-chip" communication bus.

(ii) "an executable software image" having a header and at least one data segment

121. Bauer and Svensson combined discloses an "executable software image."[13] For example, Bauer discloses a "data image" that can include "[o]bject code and data," and can be in the form of an "executable file." Ex-1109, ¶¶30-32; *see also* Ex-1110, 1:11-15, 2:11-15, 6:19-23. A person of ordinary skill in the art would understand from Bauer's teaching that (1) object code and/or data is in the form of software, (2) a data image that includes object code and/or data is a "software image," and further (3) the data image in the form of an executable file is an "executable software image."

---

[13] The '949 patent admits that an executable software image having a header and multiple segments of code is prior art. Ex-1101, 1:37-41, 2:14-16.

122. Bauer and Svensson combined teaches that the executable software image can have a header, section information, and at least one data segment (one or more data sections). Ex-1109, ¶¶27, 31-36, 43, Figs. 1A-1C. In particular, Figures 1A-1C of Bauer below show the file format of a data image 100:
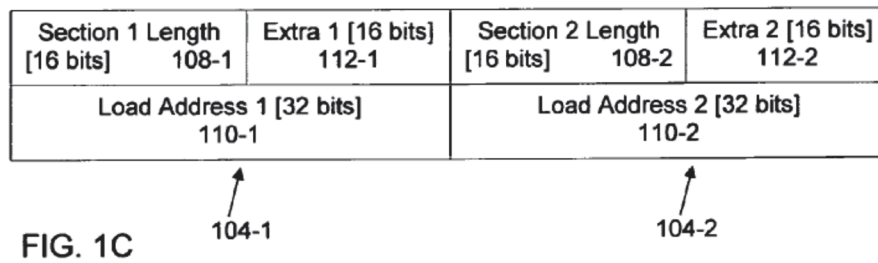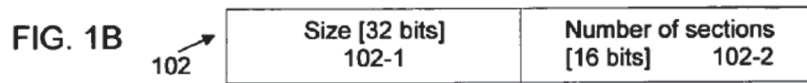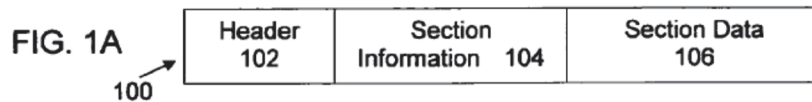
Ex-1109, Figs. 1A-1C. The file format includes a header 102, section information 104, and section data 106. *Id.*, ¶32, Fig. 1A. The header 102 indicates the size 102-1 and number of sections 102-2 in the section data 106. *Id.*, ¶33, Fig. 1B. The section information 104 indicates the length 108 and load address 110 of each section in the section data 106. *Id.*, ¶34, Fig. 1C. The load addresses are the destination addresses in the system memory (DSP XRAM) where the sections are to be transferred. *Id.*, ¶¶32, 34. The section data 106 includes one or more sections containing object code and data for the image. *Id.*, ¶¶32-34, Figs. 1A-1C.

123. From Bauer, it is clear (1) that data image 100 includes only one header 102, one section information 104, and one section data 106 (Ex-1109, ¶32, Fig. 1A); and (2) that the header 102 and section information 104 contains information about all the sections in section data 106 (*id.*, ¶33-34, Figs. 1B-1C). Accordingly, header 102 and section information 104 are associated with the *entire image*.

> ### (iii) an executable software image "for the secondary processor that is stored in memory coupled to the primary processor"

124. Bauer and Svensson combined discloses that the executable software image for the secondary processor (DSP device) can be stored in memory (non-volatile memory) coupled to the primary processor (ARM device).[14] Ex-1109, ¶¶11 (describing how it was well known for a host processor to download a program to a co-processor), 31, 35-36, Fig. 2; Ex-1110, 4:9-14 ("The SARAM and DARAM 108 can be loaded from the non-volatile memory 106 by the trivial "push" method…."); 6:12-15 ("As described above, the host fills the intermediate

---

[14] The '949 patent admits that executable software images for the secondary processor that can be stored in a non-volatile memory coupled to the primary processor is prior art. Ex-1101, 2:3-13.

storage area in the memory 108 with code and data that the slave further copies to

end destinations in the slave-private memory 110."), Figs. 1, 3.

>   **(iv)   "[a] method comprising: receiving" at a
>   secondary processor, from a primary processor
>   via an inter-chip communication bus, a header
>   for an executable software image**

125.   Bauer and Svensson combined discloses a method for receiving at a

secondary processor (DSP device), from a primary processor (ARM device) via an

inter-chip communication bus (buses and/or DMA paths), a header for an

executable software image.[15]

126.   For example, Svensson discloses that the primary processor (ARM

device) loads an image (code and/or data) via the bus (buses and/or DMA paths) to

a hardware buffer (intermediate storage area) at the secondary processor (DSP

device) one part at a time, and the secondary processor transfers each part of the

image to its system memory (DSP XRAM) one at a time. Ex-1110, 4:1-3, 4:9-10,

4:22-26, 5:21-37, 5:53-6:15, Fig. 1 (below).

---

[15] The '949 patent admits that a secondary processor that can receive from a

primary processor a header for an executable software image, where the processors

are on separate chips, is prior art. Ex-1101, 1:45-48, 2:3-16, 2:42-45.
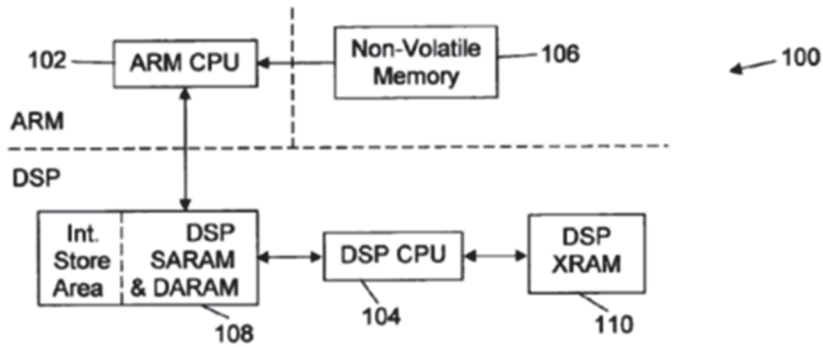
FIG. 1

127.   Figure 3 of Svensson below shows how the image can be transferred from the primary processor into the hardware buffer at the secondary processor using one or more transfer blocks. Ex-1110, 6:15-23, Fig. 3.
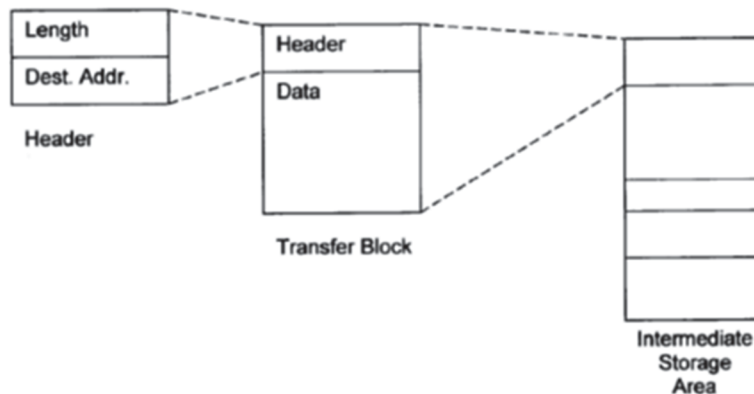
FIG. 3

Each transfer block includes a header that indicates the length (Length) of the block and a destination address (Dest. Addr.) indicating where the block is to be loaded in the system memory (DSP XRAM). *Id*.

128.   For each transfer block, the secondary processor first reads the header, which includes the destination address for the data block. Ex-1110, 5:65-67, 6:15-23, Fig. 2. The secondary processor then uses the destination address to load the

data block from the hardware buffer into the system memory. *Id*. Svensson

discloses that the secondary processor can have a communication mechanism such

as a DMA unit "to perform the actual transfers of information between memories."

Id., 6:48-7:2, 7:52-60.

129. A person of ordinary skill in the art would have found it obvious, and

would have been motivated, to transfer an image from a primary processor to a

secondary processor's system memory using a hardware buffer (intermediate

storage area), as described in Svensson, using Bauer's file format (as described in

section X.A.2.a.ii) rather than Svensson's file format, based on Bauer's express

teachings. Bauer teaches that its file format is an improvement upon the file

format described in Svensson, and can be used in the same multi-processor system

as Svensson. Ex-1109, ¶¶27, 31, 43. Unlike in Svensson in which an image

consists of multiple data segments (blocks) where each data segment has its own

header to store a destination address for that data segment, Bauer teaches an image

that consists of multiple data segments (sections) where there is one header and

one section information (which stores the destination addresses) that describe all

the data segments. Bauer describes the benefit of this approach: "[t]his simplifies

optimization in a number of circumstances, for instance, if sections are to be

loaded into memory" and "makes memory loading efficient as there is no need to

search through an image for section headers when loading." Ex-1109, ¶43; *see*

also id., ¶16 ("Greater efficiency in loading data can reduce response times in such systems, and space-efficient storage saves valuable memory."), ¶27.

130.   Bauer does not explicitly describe the loading process from the primary processor to the secondary processor in much detail, particularly the role of the hardware buffer in the loading process. However, Bauer expressly cites to Svensson as one example of a program loader that can load data using the invention described in Bauer. Ex-1109, ¶31. A person of ordinary skill in the art would understand this to mean that Bauer can load its file format using a similar program loader (and loading process) as that disclosed in Svensson. That is, in Bauer's multi-processor system 200 (shown in Figure 2), the primary processor (ARM device) can load the data image 100 (shown in Figures 1A-1C) to the hardware buffer (intermediate storage area) in the shared volatile memory 208 at the secondary processor (DSP device), and the secondary processor can then transfer the data image 100 to end destinations in the system memory (DSP XRAM) 210 based on the destination addresses stored in the section information. *Id.*, Figs. 1A-1C, 2.

131.   For these reasons, including as described in section X.A.1, the person of ordinary skill in the art would have looked to the teachings of Bauer and Svensson combined. The person of ordinary skill in the art would have found that loading Bauer's file format in the same multi-processor system disclosed in Bauer

and Svensson—but described in more detail in Svensson—would have been obvious to try, would have involved a simple substitution of one known feature (Svensson's file format) for another (Bauer's file format), and would have had a reasonable expectation of success.

<div style="text-align: center">

(v) **Bauer and Svensson combined renders obvious the "image header"**

</div>

132. As I explained in Section VIII.A, an "image header" means "a header associated with the entire image that specifies where the data segments are to be placed in the system memory." Bauer's file format does not meet the claimed "image header"—in Bauer, the header (header 102) is associated with the entire image, but it does not specify where the data segments (sections) are to be placed in the system memory (DSP XRAM). Instead, Bauer teaches that the section information 104, rather than the header 102, specifies the destination addresses. Ex-1109, ¶32-34, Figs. 1A-1C; *see also* section X.A.2.a.ii.

133. A person of ordinary skill in the art would have found it obvious to modify Bauer and Svensson combined to provide, in the header rather than in the separate section information or elsewhere, the destination addresses specifying where the data segments are to be placed in the system memory. The resulting modification would be an "image header" that is associated with the entire image

and that further specifies where the data segments are to be placed in the system memory.

134. A person of ordinary skill in the art would have been motivated to look to the teachings of Bauer and Svensson combined for the reasons described in sections X.A.1 and X.A.2.a.iv. The person of ordinary skill in the art who was familiar with Bauer was indeed aware of Svensson, and would have looked to the teachings of both to understand how different file formats can be loaded in the same multi-processor system. Ex-1109, ¶31, Fig. 2; Ex-1110, 6:12-25, Fig. 1. The person of ordinary skill in the art would have also contemplated other types of file formats that result in greater efficiency in loading data, reduced response times, and space-efficient storage. Ex-1109, ¶16, 27, 43. With the change in file format from having one header per data segment (block) as disclosed in Svensson, to having one header and one section information for multiple data segments (sections) as disclosed in Bauer, the person of ordinary skill in the art would have considered different portions of the executable software image in which to place the destination addresses for the data segments. In fact, Bauer expressly contemplates that "[o]ther arrangements" of this same file information are possible. Ex-1109, ¶29.

135. Bauer and Svensson combined teaches putting the destination addresses before all the code and data of an image. For example, Bauer discloses

that "the sections can be located in sequence *after* the header and the section

information." Ex-1109, ¶28. A person of ordinary skill in the art would therefore

understand there to be a finite number of locations in an image where these

destination addresses can be stored. There are two possible locations—the

destination addresses could be stored in the very beginning of the image (*i.e.,* in the

header) or in a location separate from the header but before the start of the code

and data (such as the separate section information of Bauer).

136. A person of ordinary skill in the art would find that storing the

destination addresses in the header rather than in a different location would have

been obvious to try because of the limited number of locations to put the

destination addresses. The person of ordinary skill in the art would have found this

to involve a simple substitution of one known feature (a header and a separate

section information that includes the destination addresses) for another (a header

that includes the destination addresses). The person of ordinary skill in the art

would have had a reasonable expectation that modifying Bauer's file format to put

the destination addresses in the header rather than in the section information would

have worked as a file format that would be loaded in a similar manner in the same

multi-processor system as taught by Bauer and Svensson combined.

137. For the foregoing reasons, Bauer and Svensson combined renders

obvious a method comprising receiving at a secondary processor, from a primary

processor via an inter-chip communication bus, an image header for an executable

software image for the secondary processor that is stored in memory coupled to the

primary processor, the executable software image comprising the image header and

at least one data segment.

> **b.** **[10b] "the image header and each data segment being received separately;"**

138.   Bauer and Svensson combined renders obvious this limitation;

however, to the extent the Patent Owner contends that Bauer and Svensson

combined does not teach this separate receipt, Kim expressly teaches this feature,

and therefore the combination of Bauer, Svensson, and Kim renders obvious this

limitation.

> **(i)** **Obviousness in View of Bauer and Svensson Combined**

139.   As described in connection with claim [10a], Bauer and Svensson

combined, as modified above to include destination addresses in an image header,

renders obvious that the secondary processor (DSP device) receives an image

header and each data segment. Bauer and Svensson combined teaches that it is

important that the header and section information containing the destination

addresses in Bauer should be "retrieved" before the data segments (sections) are

read and processed. Ex-1109, Abstract ("The header and section information are

arranged in an image having this format such that they are ***readable before*** the

sections are ***processed***."), ¶30 ("…the information about the sections can be

retrieved before the sections are ***read***."); *see also id.*, ¶¶28-29, 47.

140.   A person of ordinary skill in the art would understand from Bauer and

Svensson combined that "retrieving" the header and section information before the

data segments are read and processed means that the secondary processor's

hardware buffer (intermediate storage area) receives the header and section

information ***separately*** from (and before) the data segments are transferred.

141.   Svensson teaches that the primary processor transfers an image (code

and/or data) one part at a time to the hardware buffer at the secondary processor.

Ex-1110, 5:28-36, 5:53-6:25, Fig. 2 (steps 214-226). Svensson also teaches that

the hardware buffer can be of different sizes depending on factors including the

size of the part (block) of the image to be transferred. *Id.*, 6:26-33, 6:67-7:2. A

person of ordinary skill in the art would understand this to suggest transferring

different logical parts of an image (such as the header, section information, and

data segments of Bauer) separately, one part at a time. Bauer describes its section

information as a "block," which would also suggest, to a person of ordinary skill in

the art, transferring it separately from the other blocks (sections). Ex-1109, ¶¶27,

43 ("section information 104 … The block 104"), Figs. 1A-1C.

142.   As discussed in section VI.C, the Patent Owner added this separate

receipt feature during prosecution to distinguish the claims from Svensson PCT,

and it was this claim feature that led the Examiner to allow the application. Bauer, which was not before the Patent Office, teaches this feature.

(ii) Obviousness in View of the Combination of Bauer, Svensson, and Kim

143. To the extent the Patent Owner contends that Bauer and Svensson combined does not teach this claim feature, Kim, which was also not before the Patent Office, expressly teaches this feature. Kim discloses a multi-processor system for the transfer of data files in which program header information is transmitted separately from, and prior to, the associated data. For example, Figure 1 of Kim below shows a multi-processor system that includes a primary processor (system management processor 11) coupled to multiple secondary processors (main processors 13, switching subsystem processors 14, device processors 15). Ex-1112, 4:7-12, Fig. 1.

FIG. 1
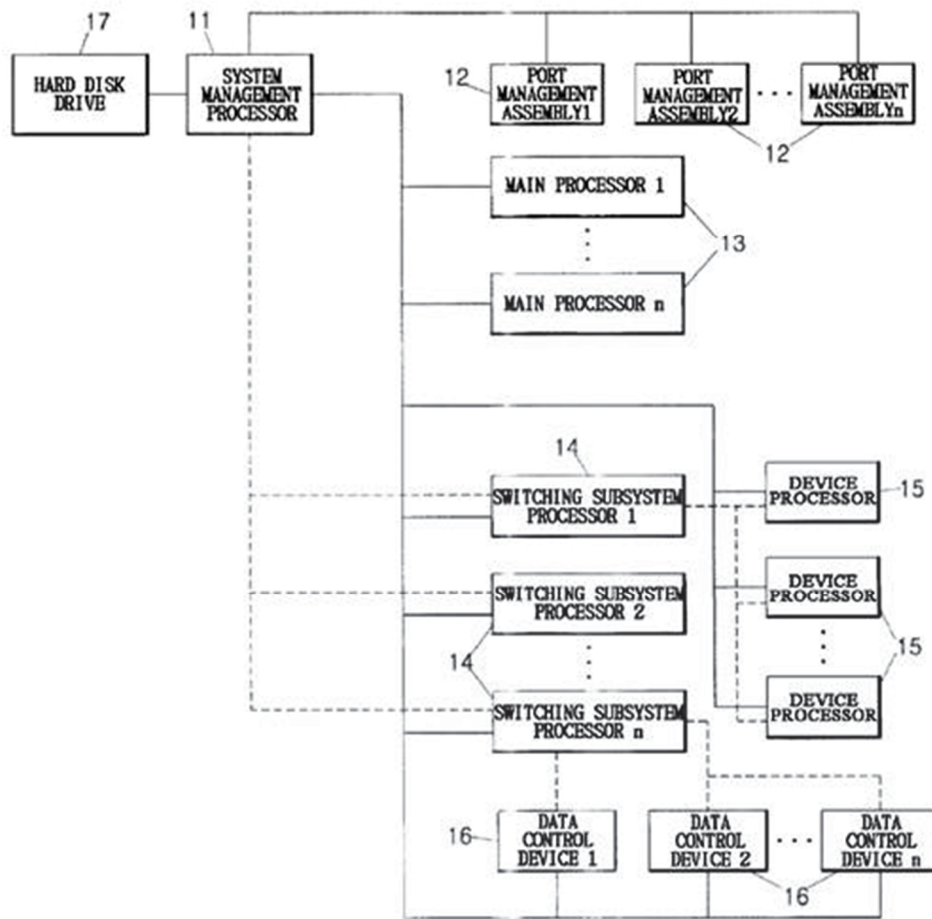
The primary processor is coupled to a non-volatile memory (hard disk drive 17)

that stores the program blocks that need to be loaded to a secondary processor. *Id.*,

4:13-14, Fig. 1.

 144. Figure 3 of Kim below shows a conventional loading method for

loading a program stored in the non-volatile memory to one of the plurality of

secondary processors. Ex-1112, 5:9-6:5, Fig. 3.

FIG. 3
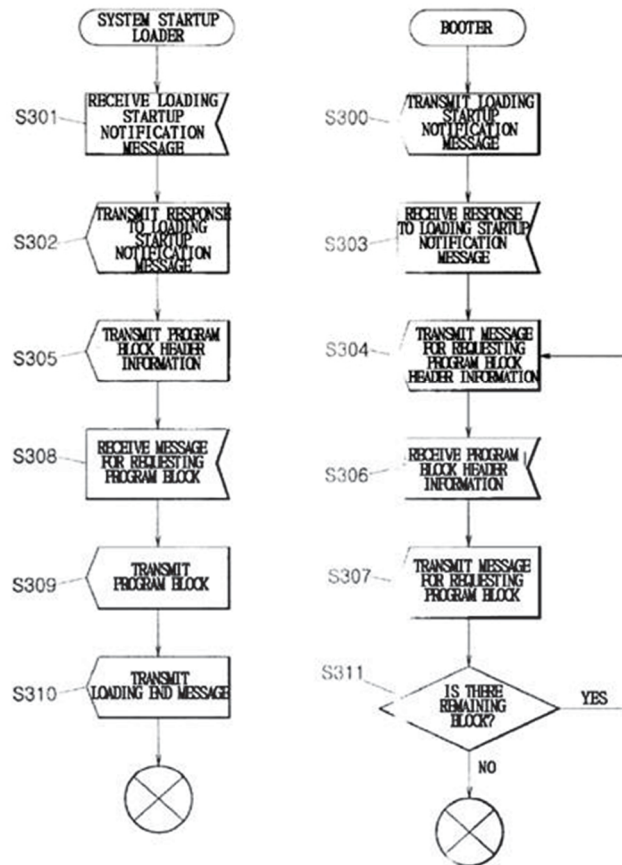
145.   As part of the loading method, the secondary processor first requests

the ***program block header information*** (step 304) from the primary processor. Ex-

1112, 5:16-19, Fig. 3. The program block header information is then sent (steps

S305 and S306) from the primary processor to the secondary processor. *Id.*, 5:19-

21, Fig. 3. Once the secondary processor receives the program block header

information, it then requests a ***program block corresponding to actual program***

content (step S307) from the primary processor. *Id.*, 5:21-23, Fig. 3. The program

block is then sent from the primary processor to the secondary processor (step

S309). *Id.*, 5:25-6:2, Fig. 3. In this way, the multi-processor system of Kim

provides a system in which the program block header information is transmitted

and ***received separately*** from the program block. The loading method used in Kim

to transfer parts of the program separately is similar to that described in Svensson.

Cf. Ex-1110, 5:28-36, 5:53-6:11, Fig. 2 (steps 214-216).

146. A person of ordinary skill in the art would have found it obvious to

use the loading method of Kim in the multi-processor system of Bauer and

Svensson combined. As described in connection with claim [10a], a person of

ordinary skill in the art would have found it obvious to combine Bauer and

Svensson.

147. As I described above, Kim discloses a multi-processor system for the

transfer of data files in which program header information is transmitted separately

from, and prior to, the associated data. The method described in Kim, in which the

program block header information is received first and then the program block is

received separately, would be consistent with, and an obvious way to implement,

the requirement of Bauer and Svensson combined that the header and section

information containing the destination addresses in Bauer should be retrieved

before the data segments (sections) of an image are read and processed.

148. A person of ordinary skill in the art would have been motivated to

combine Kim with Bauer and Svensson combined in order to reduce memory

requirements for the secondary processor. By receiving and processing the header

and section information in Bauer first, and then receiving and processing the data

segments separately, the secondary processor can use the same memory locations

for both tasks, avoiding the need for two sets of memory locations.

149. A reduced memory requirement is important in the context of Bauer

and Svensson combined because the intermediate storage area is of limited size.

Bauer and Svensson combined discloses that the image can be larger than the

intermediate storage area, and further discloses that Bauer's file format can support

data "up to a total of four gigabytes (GB) in size." Ex-1109, ¶¶33, 38; Ex-1110,

6:26-28, 6:37-43. A person of ordinary skill in the art would understand from this

combination that in order to support such a large image in an intermediate storage

area of limited size, the image would require many data segments (sections). All

of the data segments would not fit in the secondary processor's hardware buffer at

the same time, and thus would require that the data segments be separately

transferred. The person of ordinary skill in the art would further understand that

the section information in Bauer would need the same number of section

information entries as the number of data segments to provide information

including the destination address for each data segment. The person of ordinary

skill in the art would understand that the section information, to store all this

information, would be of sufficient size such that it would need to be transferred to

the secondary processor's hardware buffer separately from the data segments—in other words, the section information and data segments would not all fit in the hardware buffer at the same time. Bauer specifically cites the importance of using memory efficiently, which Kim provides. Ex-1109, ¶¶16, 27, 43.

150. A person of ordinary skill in the art would have understood that there are a finite number of ways to receive a file format—to receive the header and section information in Bauer (or the modified image header of Bauer) and data segments together or separately (as suggested by Bauer and Svensson combined, and also taught by Kim). Such a person would have understood that Kim's approach would have been obvious to try because it provides greater efficiency in loading data, reduced response times, and space-efficient storage, and would have involved a simple substitution of one known feature for another. The person of ordinary skill in the art also would have had a reasonable expectation that Kim's method would have worked in the multi-processor system of Bauer and Svensson combined as Kim's method is an implementation of the requirement of Bauer and Svensson combined that the header and section information containing the destination addresses in Bauer should be retrieved before the data sections of an image are read and processed. Ex-1109, Abstract, ¶¶28-30, 47.

      c.   **[10c] "processing, by the secondary processor, the image header to determine at least one location within**

system memory to which the secondary processor is coupled to store each data segment;"

151.   Bauer and Svensson combined, as modified above to include destination addresses in an image header, renders obvious this limitation. [16]

152.   Bauer and Svensson combined discloses a system memory (DSP XRAM) to which the secondary processor (DSP device) is coupled. Ex-1109, ¶¶35-36, Fig. 2 (below); *see also* Ex-1110, 3:49-8, Fig. 1.

FIG. 2
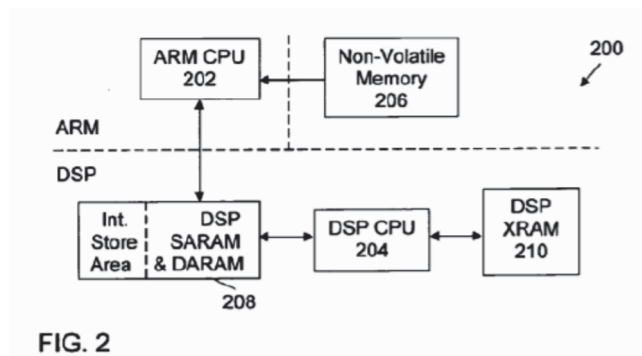
153.   As described in connection with claim [10a], Bauer and Svensson combined teaches that the modified image header of Bauer contains the destination addresses in the system memory (DSP XRAM) where the data segments (sections) are to be transferred. Ex-1109, ¶¶32, 34, Fig. 1C. Accordingly, the destination

---

[16] The '949 patent admits that "some processing," by the secondary processor, of a received packet having a header "is required for that data to be stored where it needs to go (e.g., within the secondary processor's volatile memory)." Ex-1101, 2:14-22, 2:45-54.

addresses specify "at least one location within system memory to which the secondary processor is coupled to store each data segment."

154. Bauer and Svensson further teaches that the secondary processor "processes" the modified image header of Bauer. For example, Bauer and Svensson combined discloses that the secondary processor transfers the image, which includes the header and section information containing destination addresses in Bauer, from the hardware buffer to its system memory—different parts of the image are loaded to the hardware buffer, and then to the system memory, one part at a time. Ex-1109, ¶36 (describing how the image containing a header and section information can be stored in the system memory), Figs. 1A-1C; Ex-1110, 5:21-28, 5:65-67, 6:12-15 (describing how the contents of the intermediate storage area, which contains part of an image that can be the header, are loaded to the system memory one part at a time), Figs. 1, 2 (step 220), 3.

155. In addition, Bauer and Svensson combined teaches that it is important that the header and section information containing the destination addresses in Bauer should be "retrieved" before the data segments (sections) are read and processed. Ex-1109, Abstract; ¶¶28-30, ¶47. A person of ordinary skill in the art would understand that the secondary processor would have to load the header and section information in Bauer from the hardware buffer to the system memory in order to read the destination addresses for later loading of the data segments, as

well as to free up space in the hardware buffer for the data segments. The person

of ordinary skill in the art would understand this to mean that the secondary

processor is configured to "process" the header and section information in Bauer

(or modified image header of Bauer).

156. Accordingly, Bauer and Svensson combined renders obvious

processing, by the secondary processor, the image header to determine at least one

location within system memory to which the secondary processor is coupled to

store each data segment.

> **d.   [10d] "receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment; and"**

157. Bauer and Svensson combined discloses this limitation.[17] As

described in connection with claim limitations [10a]-[10b], Bauer and Svensson

combined discloses receiving at the secondary processor (DSP device), from the

primary processor (ARM device) via the inter-chip communication bus (buses

and/or DMA paths), each data segment (section).

> **e.   [10e] "scatter loading, by the secondary processor, each data segment [directly] to the determined at least**

---

[17] The '949 patent admits that a secondary processor that can receive from a

primary processor each segment of code in an image, where the processors are on

separate chips, is prior art. Ex-1101, 1:45-48, 2:3-22, 2:42-45.

one location within the system memory, and each data segment being scatter loaded based at least in part on the processed image header.”

158. Bauer and Svensson combined, as modified above to include destination addresses in an image header, renders obvious this limitation.[18]

159. In Bauer and Svensson combined, with the modified image header of Bauer, the secondary processor (DSP device) loads the data segments (sections) that separately follow the image header, which it receives from the primary processor (ARM device) in the hardware buffer (intermediate storage area), directly to the system memory (DSP XRAM) using the destination addresses from the modified image header. For example, Bauer and Svensson combined discloses that the image transferred from the hardware buffer to the system memory also includes the data segments (sections). Ex-1109, ¶36 (describing how the image containing the data section having one or more sections can be stored in the system memory), Figs. 1A-1C; Ex-1110, 5:21-28, 5:65-67, 6:12-15 (describing how the contents of the intermediate storage area, which contains part of an image having code and/or data, are loaded to the system memory one part at a time), Figs. 2 (steps 214-226, esp. step 220), 3.

---

[18] The '949 patent admits that scatter loading is prior art. Ex-1101, 2:35-41.

160.   In addition, Bauer discloses that each data segment (section) (1) has

its own load (or destination) address in the section information specifying where

the data segment is to be placed in the system memory and (2) can be arranged in

the image in any suitable order (*e.g.,* "in order of the section load addresses" or in

"an arbitrary order"). Ex-1109, ¶37. A person of ordinary skill in the art would

understand from this disclosure that Bauer teaches that the data segments can be

loaded in contiguous or non-contiguous locations of the system memory, and thus

the data segments are "scatter loaded." *Cf.* Ex-1101, 9:12-15 (With scatter

loading, "the image segments are not necessarily placed into consecutive locations

within the secondary processor's system memory 305. Instead, the segments may

be spread out in different locations of the memory."), 9:21-41. Accordingly, a

person of ordinary skill in the art would find that Bauer and Svensson combined

renders obvious scatter loading, by the secondary processor, each data segment

[directly] to the determined at least one location within the system memory, and

each data segment being scatter loaded based at least in part on the processed

image header.

> **3.   Claim 11: "The method of claim 10 further comprising booting the secondary processor using the executable software image."**

161.   Claim 11 is rendered obvious by the combination of Bauer, Svensson,

and Kim. As I explained above, this combination renders obvious the method of

claim 10. Bauer and Svensson combined discloses the further requirement of

claim 11.[19]

162. Bauer and Svensson combined renders obvious that the executable

software image loaded to the system memory (DSP XRAM) of the secondary

processor (DSP device) can be used to boot the secondary processor. For example,

Svensson describes the need for a bootloader solution to load "boot code" to the

system memory (DSP XRAM) because the primary processor (ARM device)

cannot access the system memory. Ex-1110, 4:9-14. In addition, Figure 2 of

Svensson shown below teaches how its "OS-friendly bootloader" is used to

provide that solution—to load boot code to the system memory. *Id.*, 4:20-6:11,

Fig. 2.

---

[19] The '949 patent admits that booting the secondary processor using the

executable software image is prior art. Ex-1101, 2:1-13.

```
                    ┌──────────────────┐
              202   │ Reset and Hold   │
                    │ Slave Processor  │
                    └──────────────────┘
                             │
                             ▼
                    ┌──────────────────┐◄────────┐
              204   │  Push Info to    │         │
                    │ Slave Processor  │         │
                    └──────────────────┘         │
                             │                   │
                             ▼                   │
                          ╱──────╲      No        │
              206        ╱  Push   ╲──────────────┘
                         ╲Complete?╱
                          ╲──────╱
                             │ Yes
                             ▼
                    ┌──────────────────┐
              208   │   Boot Slave     │
                    │   Processor      │
                    └──────────────────┘
                             │
                             ▼
                    ┌──────────────────┐
              210   │   Start OS in    │
                    │ Slave Processor  │
                    └──────────────────┘
                             │
                             ▼
                    ┌──────────────────┐
              212   │ Reserve Intermed.│
                    │  Storage Area    │
                    └──────────────────┘
                             │
                             ▼
                    ┌──────────────────┐
              214   │ Send Message to  │
                    │ Host Processor   │
                    └──────────────────┘
                             │
                             ▼
                    ┌──────────────────┐◄──────────────────┐
              216   │  Push Info to ISA│                    │
                    └──────────────────┘                    │
                             │                              │
                             ▼                              │
                    ┌──────────────────┐        ╱──────╲    │
              218   │ Send Message to  │       ╱        ╲ No │
                    │ Slave Processor  │──┐   ╱Complete? ╲───┘
                    └──────────────────┘  │   ╲          ╱
                             │            │    ╲──────╱
                             ▼            │   224   │ Yes
                    ┌──────────────────┐  │         ▼
              220   │   Copy ISA to    │  │  ┌──────────────────┐
                    │"Invisible" Memory│  │  │ Release Blocks;  │
                    └──────────────────┘  │  │  Load Complete   │
                             │            │  └──────────────────┘
                             ▼            │   226
                    ┌──────────────────┐  │
              222   │ Send Message to  │  │
                    │ Host Processor   │  │
                    └──────────────────┘  │
                             │            │
                             └────────────┘
```

FIG. 2

163.   Initially, in the first stage of the OS-friendly bootloader (steps 202-210 of Fig. 2), a bootloader is pushed to the shared volatile memory (DSP SARAM & DARAM) to initiate the boot of the secondary processor (step 208) so that, in the second stage (steps 212-226), the secondary processor can assist with loading "boot code" into system memory to complete the boot. Ex-1110, 4:20-6:11. This

two-stage booting process is consistent not only with the '949 patent (Ex-1101,

5:20-55, claim 6) but also other prior art (Ex-1110, 1:20-27).

164. Because Bauer expressly teaches to use the program loader described

by Svensson for loading an executable software image having Bauer's file format

(Ex-1109, ¶31), and in addition to the reasons given in connection with claim

[10a], it would have been obvious to a person of ordinary skill in the art to

combine Svensson with Bauer to use a bootloader to load "boot code" for the

secondary processor (DSP device) in the form of an executable software image

having Bauer's file format with a modified image header. Svensson further

teaches executing code loaded into system memory (DSP XRAM). Ex-1010, 6:6-

11, 8:12-16. A person of ordinary skill in the art would have understood that the

loaded "boot code" would execute as part of the booting process of the secondary

processor.

165. Bauer and Svensson combined therefore renders obvious a method of

booting a secondary processor using an executable software image.

> **4. Claim 12: "The method of claim 10 further comprising loading the executable software image directly from a hardware buffer to the system memory of the secondary processor without copying data between the system memory locations."**

166. Claim 12 is rendered obvious by the combination of Bauer, Svensson,

and Kim. As I explained above, this combination renders obvious the method of

claim 10. Bauer and Svensson combined discloses the further requirement of claim 12.

167. As described in connection with claim limitations [10b]-[10e], Bauer and Svensson combined discloses that the secondary processor loads the executable software image that it receives at its hardware buffer (intermediate storage area) one part at a time directly to its system memory (DSP XRAM) one part at a time—first it loads the modified image header of Bauer and then it scatter loads the data segments (sections).

168. Bauer and Svensson combined also discloses that the hardware buffer (intermediate storage area), which is part of a shared volatile memory (DSP SARAM & DARAM memory), is separate from the system memory (DSP XRAM). Ex-1109, ¶¶35-36, Fig. 2 (below); *see also* Ex-1110, 3:49-8, Fig. 1.

FIG. 2
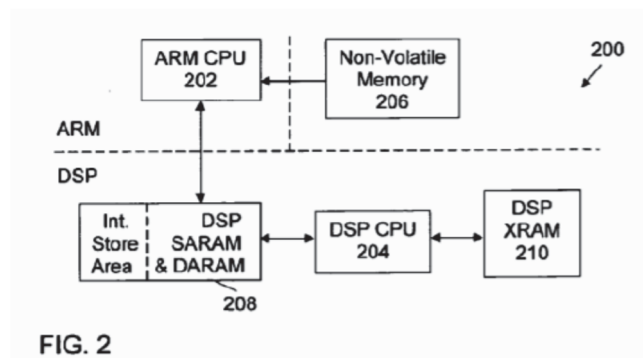
169. In Bauer and Svensson combined, the disclosure of an intermediate storage area that is separate from the system memory (DSP XRAM) is the same alleged inventive feature of the '949 patent. The '949 patent makes a distinction

between prior art systems that used a "temporary buffer" that was part of the system memory (Ex-1101, 2:23-41), and the alleged invention that uses a "hardware buffer" separate from the system memory (*id.*, 2:58-61, 7:20-26, Fig. 3). The intermediate storage area in Bauer and Svensson combined is such a separate hardware buffer.

170. A person of ordinary skill in the art would understand from Bauer and Svensson combined that the executable software image is therefore loaded directly without copying data between system memory locations on the secondary processor. *See, e.g.,* Ex-1109, ¶¶31, 36, Fig. 2; Ex-1110, 4:1-3, 5:21-28, 5:65-67, 6:12-15, Figs. 1, 3. The person of ordinary skill in the art would find no teaching or suggestion in Bauer and Svensson combined that the executable software image, before it is loaded from the hardware buffer to its final destination in the system memory, is copied to another location in the system memory. In fact, Bauer teaches away from this double copy (or "extra memory copy") approach as it teaches the importance of greater efficiency in loading data, reduced response times, and space-efficient storage. Ex-1109, ¶¶16, 27, 43.

171. Thus, Bauer and Svensson combined discloses loading the executable software image directly from a hardware buffer to the system memory of the secondary processor without copying data between the system memory locations.

> **5.** **Claim 13: "The method of claim 10 in which the processing occurs prior to the loading."**

172. Claim 13 is rendered obvious by the combination of Bauer, Svensson, and Kim. As I explained above, this combination renders obvious the method of claim 10, and renders obvious the further requirement of claim 13.

173. As described in connection with claim [10c], Bauer and Svensson combined renders obvious a method in which a secondary processor (DSP device) processes the modified image header of Bauer to determine locations (based on the destination addresses) within a system memory (DSP XRAM) to store each data segment (data section). As described in connection with claim [10b], the combination of Bauer, Svensson, and Kim, renders obvious the image header and each data segment (data section) being received separately.

174. A person of ordinary skill in the art would have found it obvious that the secondary processor would process the image header *before* loading the at least one data segment. As described in connection with claim [10b]: (1) Bauer and Svensson combined emphasizes that the header and section information containing the destination addresses in Bauer should be "retrieved" before the data segments (sections) are read and processed; and (2) the combination of Bauer, Svensson, and Kim renders obvious that the modified image header of Bauer and each data

segment are received separately, with the image header being received before the data segments.

175. A person of ordinary skill in the art would have found it obvious from this combination that when the secondary processor receives the image header—which contains the destination addresses for where to store the data segments in the system memory—separately from and before the data segments, that the secondary processor would process the image header *before* receiving the data segments to determine the destination address for each data segment. The person of ordinary skill in the art would understand that this would allow, for example, the secondary processor to build a look-up table of the destination addresses of the data associated with the image header, before the data segments are received.

176. A person of ordinary skill in the art would have been motivated to process the image header before receiving the data segments because, upon receiving each data segment, the secondary processor would then more efficiently and quickly load each data segment from the hardware buffer to the system memory, as taught by Bauer. *See* claim limitations [10b] and [10e]. The person of ordinary skill in the art also would have understood that the most logical and efficient way is to process the image header *before* receiving the data segments. The person of ordinary skill in the art would have understood that this approach would have been obvious to try, would have been one of a small number of finite

ways that results in greater efficiency in loading data, reduced response times, and

space-efficient storage, and would have involved a simple substitution of one

known feature for another. A person of ordinary skill in the art also would have

had a reasonable expectation of success given that the components involved in the

system disclosed by these references, and the manner in which they operate, were

well known.

6. Claim 14: "The method of claim 10 in which the primary and secondary processors are located on different chips."

177. Claim 14 is rendered obvious by the combination of Bauer, Svensson,

and Kim. As I explained above, this combination renders obvious the method of

claim 10. As described in connection with claim [10a], Bauer and Svensson

combined discloses the further requirement of claim 14 that the primary processor

(ARM device) and secondary processor (DSP device) are located on different

chips.[20]

---

[20] The '949 patent admits that systems in which the primary and secondary

processors are on different chips is prior art. Ex-1101, 1:45-48, 2:42-45.

7. Claim 15: "The method of claim 10 further comprising performing the receiving, processing, and loading, in at least one of a mobile phone … a computer, a handheld personal communication systems (PCS), a portable data unit…."

178. Claim 15 is rendered obvious by the combination of Bauer, Svensson, and Kim. As I explained above, this combination renders obvious the method of claim 10. Bauer and Svensson combined discloses the further limitation of claim 15 that requires the multi-processor system to perform the receiving, processing, and loading in at least "a mobile phone…a computer, a hand-held personal communication systems (PCS) unit, a portable data unit."[21]

179. Bauer and Svensson combined teaches the integration of its multi-processor system in a device—which would make that device a "computer"—such as a mobile phone, PCS unit, and portable data unit. For example, Bauer explicitly teaches that the application of its technology "is particularly useful in embedded systems as well as in other computer environments," and "in embedded systems and other computer systems" ("a computer"). Ex-1109, ¶16, 26. Bauer also teaches that in an embedded computer environment, "[a] lot of software today is sent across wireless communication links (e.g., wireless local area networks

---

[21] The '949 patent also admits that the use of a multi-processor system in a computer such as a smartphone device is prior art. Ex-1101, 1:39-44.

(WLANs), *mobile telephony networks*, etc.)." Ex-1109, ¶15. A person of ordinary skill in the art would understand from Bauer that a "mobile phone," "PCS unit," or "portable data unit" can be used in a "mobile telephony network." Svensson also teaches that its multi-processor system "can be implemented in a wide variety of environments, including for example *mobile communication devices*" or "a *mobile telephone.*" Ex-1110, 7:61-63, 8:26-29. A person of ordinary skill in the art would understand from Svensson that a "mobile communication device" can be a "computer," "mobile phone," "PCS unit," or "portable data unit."

B. Ground 2: Claims 16 And 17 Are Rendered Obvious By The Combination Of Bauer, Svensson, Kim, And Zhao

1. Reference to "Bauer and Svensson Combined"

180. Reference to "Bauer and Svensson Combined" is discussed in section X.A.1.

2. Claim 16

181. Claim 16 is rendered obvious by the combination of Bauer, Svensson, and Zhao; however, to the extent the Patent Owner contends that this combination does not teach that different parts of an executable software image are received separately (claim [16b]), this claim is rendered obvious by the combination of Bauer, Svensson, Kim, and Zhao.

a.   [16a] "An apparatus comprising: means for receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor, the executable software image comprising the image header and at least one data segment,"

182.   The combination of Bauer, Svensson, and Zhao renders obvious this limitation. As discussed in section VIII.B.1, (1) the function performed by this claim element is "receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor"; and (2) the corresponding structure for performing this function is "a secondary processor (*e.g.,* 110, 210, 302) connected to a primary processor (*e.g.,* 104, 204, 301) via an inter-chip communication bus (*e.g.,* 134, 234, 310) such as…a Universal Asynchronous Receiver/Transmitter (UART) bus…and equivalents thereof."

183.   As discussed in connection with claim [10a], Bauer and Svensson combined renders obvious the corresponding function; however, the combination does not expressly disclose all of the corresponding structure. Bauer and Svensson combined teaches an *apparatus* (multi-processor system) having a *secondary processor* (DSP device) connected to a *primary processor* (ARM device) by an

inter-chip communication bus (buses and/or DMA paths). Ex-1109, ¶¶35-36, Fig.

2. However, the combination does not teach the specific inter-chip communication

buses required by the corresponding structure—this is explicitly taught by Zhao.

184. A person of ordinary skill in the art would have been generally aware

of the well-known, standardized inter-chip communication buses, including those

listed in the '949 patent. For example, standardized inter-chip communication

buses such as the UART interfaces (buses) are disclosed in Zhao. Figure 3 of Zhao

below shows a mobile computing device 100 that "may comprise a dual processor

architecture including a host processor 102 and a radio processor 104," which can

be located on separate chips and that can "communicate with each other using

interfaces 106." Ex-1113, ¶¶32, 34, 44, Fig. 3.
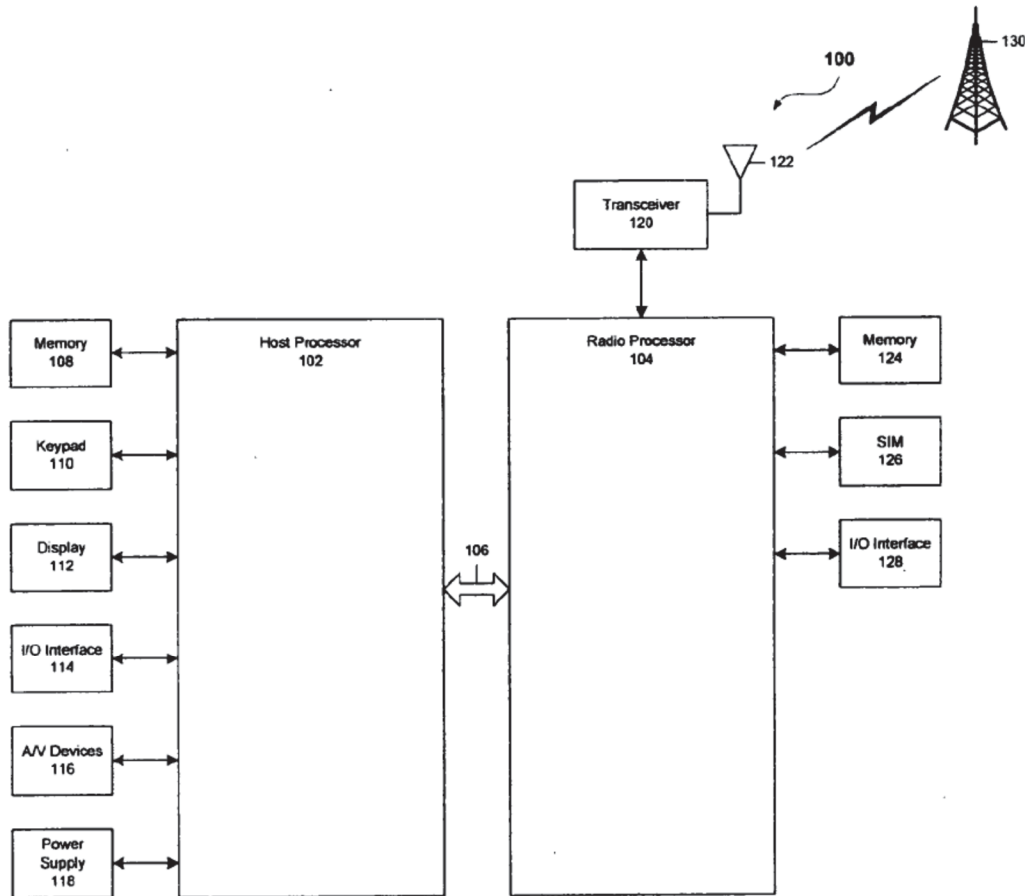
FIG. 3

185. Interfaces 106 can include "one or more universal serial bus (USB) interfaces, micro-USB interfaces, ***universal asynchronous receiver-transmitter (UART) interfaces***, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth." *Id.*, ¶32, Fig. 5.

186. A person of ordinary skill in the art would have been motivated to combine Bauer and Svensson for the reasons described in connection with claim [10a]. In order to connect the separate processor chips of the multi-processor system taught in Bauer and Svensson combined, a person of ordinary skill in the

art would have found it to be a simple known alternative to select an inter-chip communication bus from amongst the standardized inter-chip communication buses that were known in the prior art, such as those taught in Zhao. The use of standardized inter-chip communication buses would have been obvious to a person of ordinary skill in the art as it simply involves the combination of prior art elements (chips and standardized inter-chip communication buses) according to known methods to achieve a predictable result with a reasonable expectation of success.

b.   [16b] "the image header and each data segment being received separately"

187.   As described in connection with claim [10b], Bauer and Svensson combined, as modified above to include destination addresses in an image header, renders obvious this limitation; however, to the extent the Patent Owner contends that Bauer and Svensson combined does not teach this separate receipt, Kim expressly teaches this feature, and therefore the combination of Bauer, Svensson, and Kim renders obvious this limitation. A person of ordinary skill in the art would have been motivated to combine Bauer, Svensson, and Kim for the reasons described in connection with claim [10b]. A person of ordinary skill in the art would have also been motivated to combine Bauer, Svensson, Zhao, and Kim for the reasons described in connection with claim [16a].

    c.   **[16c] "means for processing, by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment;"**

188.   The combination of Bauer, Svensson, and Zhao renders obvious this limitation. As discussed in section VIII.B.2, (1) the function performed by this claim element is "processing, by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment"; and (2) the corresponding structure for performing this function is "a modem processor coupled to a system memory, and equivalents thereof."

189.   As discussed in connection with claim [10c], Bauer and Svensson combined renders obvious the corresponding function; however, the combination does not expressly disclose all of the corresponding structure—it discloses a processor (DSP device) coupled to a system memory (DSP XRAM), but does not expressly describe it as a "modem" processor. Modem processors were well known at the earliest effective filing date of the '949 patent. Indeed, the background section of the '949 patent even admits that this feature is prior art. *See* Ex-1101, 1:45-48 ("A problem exists on a significant number of devices (such as smart phones) that incorporate multiple processors (*e.g.,* a standalone application processor chip integrated with a separate ***modem processor chip***).").

190.   Zhao expressly discloses this feature. Zhao shows a multi-processor system that has an architecture similar to Bauer and Svensson combined. In particular, Figure 3 of Zhao shows a mobile computing device 100 that "may comprise a dual processor architecture including a host processor 102 and a radio processor 104," each having a memory, and that can "communicate with each other using interfaces 106." Ex-1113, ¶32, Fig. 3. The mobile computing device can be a smartphoneor laptop computer, whose radio processor 104 "may be implemented as a communications processor using any suitable processor or logic device, such as a ***modem processor*** or baseband processor." *Id.*, ¶¶26, 44.

191.   A person of ordinary skill in the art would have been motivated to combine Bauer and Svensson for the reasons described in connection with claim [10a]. The person of ordinary skill in the art would have further understood, from his/her knowledge of the field, and specifically from Zhao, that a multi-processor system can include an application processor and a modem processor coupled to a system memory in a mobile computing device such as a smartphone. The person of ordinary skill in the art would have understood that Bauer and Svensson combined teaches a multi-processor system having a primary processor (ARM device) and a secondary processor (DSP device) coupled to system memory (DSP XRAM) that can be used in a device such as a mobile phone (or computer, PCS unit, or portable data unit). Ex-1109, ¶15; Ex-1110, 7:61-63, 8:26-29; *see also*

discussion in connection with claim 15. The person of ordinary skill in the art would have understood that the primary processor (ARM device) could function as the application processor and the secondary processor (DSP device) could function as the modem processor to allow for the transfer of data over a mobile network, including allowing a user to make and receive calls, as taught by Bauer and Svensson combined. Ex-1109, ¶15; Ex-1110, 7:61-63, 8:26-29. Such a person would have understood that this would have been obvious to try and would have involved a simple substitution of one known feature for another.

192. The combination of Bauer, Svensson, and Zhao therefore render obvious a modem processor (secondary processor) coupled to a system memory for processing the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment.

> d. **[16d] "means for receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment, and"**

193. The combination of Bauer, Svensson, and Zhao discloses this limitation. As discussed in section VIII.B.3, (1) the function performed by this claim element is "receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment"; and (2) the corresponding structure for performing this function is "a secondary processor (*e.g.,* 110, 210, 302) connected to a primary processor (*e.g.,* 104, 204, 301) via an

inter-chip communication bus (*e.g.,* 134, 234, 310) such as…a Universal

Asynchronous Receiver/Transmitter (UART) bus…and equivalents thereof."

194.   As discussed in connection with claim [10d], Bauer and Svensson

combined discloses the corresponding function; however, the combination does not

expressly disclose all of the corresponding structure. Bauer and Svensson

combined discloses an ***apparatus*** (multi-processor system) having a ***secondary***

processor (DSP device) connected to a ***primary processor*** (ARM device) by an

inter-chip communication bus (buses and/or DMA paths). Ex-1109, ¶¶35-36, Fig.

2. However, the combination does not teach the specific inter-chip communication

buses required by the corresponding structure—this is explicitly taught by Zhao as

described in connection with claim [16a]. A person of ordinary skill in the art

would have been motivated to combine Bauer, Svensson, and Zhao for the reasons

described in connection with claim [16a].

> e.   **[16e] "means for scatter loading, by the secondary**
> **processor, each data segment directly to the**
> **determined at least one location within the system**
> **memory, and each data segment being scatter loaded**
> **based at least in part on the processed image header."**

195.   The combination of Bauer, Svensson, and Zhao renders obvious this

limitation. As discussed in section VIII.B.4, (1) the function performed by this

claim element is "scatter loading, by the secondary processor, each data segment

directly to the determined at least one location within the system memory, and

each data segment being scatter loaded based at least in part on the processed image header"; and (2) the corresponding structure for performing this function is "modem processor coupled to a system memory, and equivalents thereof."

196. As described in connection with claim [10e], Bauer and Svensson combined renders obvious the corresponding function; however, the combination does not expressly disclose all of the corresponding structure—it discloses a processor (DSP device) coupled to a system memory (DSP XRAM), but does not expressly describe it as a "modem" processor. The "modem" processor is explicitly taught by Zhao as described in connection with claim [16c]. A person of ordinary skill in the art would have been motivated to combine Bauer, Svensson, and Zhao for the reasons described in connection with claim [16c].

197. A person of ordinary skill in the art would have also been motivated to combine Bauer, Svensson, Kim, and Zhao for the reasons described in connection with claim limitations [10a] (describing the combination of Bauer and Svensson), [10b] (describing the combination of Bauer, Svensson, and Kim), and [16a] and [16c] (describing the combination of Bauer, Svensson, and Zhao).

3. **Claim 17: "The apparatus of claim 16 integrated into at least one of a mobile phone … a computer, a hand-held personal communication systems (PCS) unit, a portable data unit…."**
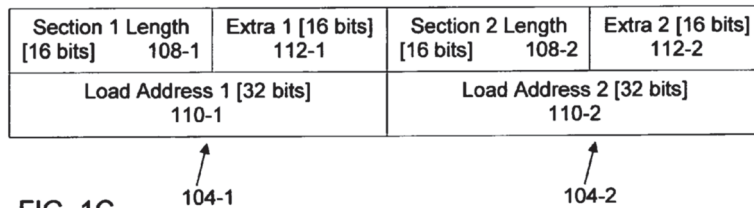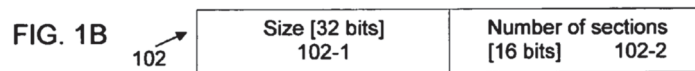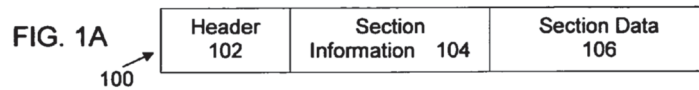
198. Claim 17 is rendered obvious by the combination of Bauer, Svensson, Kim, and Zhao. As I explained above, this combination renders obvious the apparatus of claim 16. As described in connection with claim 15, Bauer and Svensson combined discloses the further requirement of claim 17 that the apparatus of claim 16 is integrated into at least "a mobile phone…a computer, a hand-held personal communication systems (PCS) unit, a portable data unit."

199. The chart below provides a summary of the disclosure of the references in this ground discussed above and presented in the form of a claim chart.

	Bauer + Svensson + Kim
[10a] A method comprising: receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor, the executable software image comprising the image header and at least one data segment,	Bauer Bauer (Ex-1109) at ¶11 ("Also known are program downloading methods for use in data processing systems, integrating non-program information and program information into an executable file that is used by a host processor to download the program to a selected co-processor.") Bauer (Ex-1109) at ¶16 ("Greater efficiency in loading data can reduce response times in such systems, and space-efficient storage saves valuable memory.") Bauer (Ex-1109) at ¶18 ("In another aspect of this invention, there is provided a computer-readable medium containing a data image for loading into a memory in a processor system.") Bauer (Ex-1109) at ¶27 ("The format described here includes a

header, section information, and one or more sections. The section information contains the information for all sections, which is more advantageous than having each section include its own information, i.e., the information is concentrated rather than distributed across the sections. Furthermore, the section information contains information about the encoding of the sections.")

Bauer (Ex-1109) at ¶28 ("Each section includes binary data that is encoded independently of other sections, and the header and section information contains information about the sizes, load addresses, and encoding, e.g., encryption and/or compression, of the sections. The header and section information are arranged in an image having this format such that they are readable before the sections are processed. For example, the sections can be located in sequence after the header and the section information, in an order determined by their load addresses.")

Bauer (Ex-1109) at ¶29 ("Other arrangements are possible, of course. It is important only that the header and section information can be read before the rest of an image. The locations of the header and section information can be anywhere in the image, provided it is possible to access the header and section information before the rest of the image.")

Bauer (Ex-1109) at ¶30 ("Thus, it will be appreciated that the format described here, in contrast to prior data formats, supports individual coding of sections, where a section can contain any type of data, such as executable, binary, text, etc.")

Bauer (Ex-1109) at ¶31 ("There are many possible applications of this format and its individually coded sections. For example, an operating system memory manager can load and unload sections of memory according to images in this format. It can also be used as a file format in which executable files are stored, and linkers and program loaders can be readily adapted to support (read, write, and interpret) the format. Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly. One example of such a program loader is described in U.S. patent application Ser. No. 11/040,798 filed on Jan. 22, 2005, by M. Svensson et al. for "Operating-System-Friendly Bootloader".")

Bauer (Ex-1109) at ¶32 ("FIG. 1A depicts a binary data image

100 in this file format, including a header 102, section information 104, and section data 106. The section data 106 includes the data of the one or more sections included in the image 100.")

Bauer (Ex-1109), Figures 1A-C:

FIG. 1A 100 — Header 102 | Section Information 104 | Section Data 106

FIG. 1B 102 — Size [32 bits] 102-1 | Number of sections [16 bits] 102-2

FIG. 1C — Section 1 Length [16 bits] 108-1 | Extra 1 [16 bits] 112-1 | Section 2 Length [16 bits] 108-2 | Extra 2 [16 bits] 112-2 | Load Address 1 [32 bits] 110-1 | Load Address 2 [32 bits] 110-2 — 104-1 | 104-2

Bauer (Ex-1109) at ¶33 ("As depicted in FIG. 1B, the header 102 contains a size information element 102-1 that indicates the total size of the sections 106 (in bytes, for example)….The header 102 also contains a number-of-sections information element 102-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶34 ("Each section in the section data 106 has a respective "section information" entry in the section information 104, and two such section information entries 104-1, 104-2 are depicted in FIG. 1C….The load addresses of the sections are indicated by address information elements 110-1, 110-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor

204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of a digital signal processor (DSP) device. ***The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory*** 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.")

Bauer (Ex-1109) ¶36 ("***Most commercially available DSP devices include on-chip memories***, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. ***The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C.*** The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Bauer (Ex-1109), Figure 2:

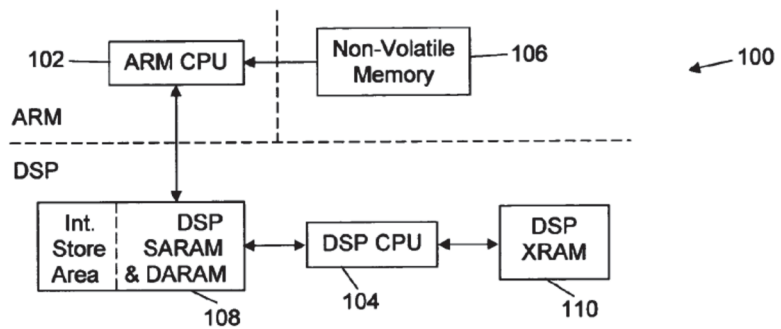
FIG. 2

Bauer (Ex-1109) at ¶38 ("Having all section information entries 104 collected together in the image 100 advantageously simplifies system navigation through the image, and having all section data arranged in a sequence makes it possible to

optimize loading of the sections. For instance, it is simple to split or concatenate sections when they are adjacent in memory. The ability to split sections can be useful, for instance, when a DMA transfer is to be set up. As there is always a small overhead when setting up a DMA transfer, a DMA unit can be used in an efficient way by arranging the size of the data to be transferred to be equal or close to the block sizes used by the DMA unit. As sections can be located sequentially in an image 100, it is simple to split a section into several suitable pieces before downloading it.")

Bauer (Ex-1109) at ¶43 ("Having information about the sections collected in the header 102 and section information 104 simplifies optimization in a number of circumstances, for instance, if sections are to be loaded into memory. The block 104 lists all sections, preferably in order of memory location, and this makes memory loading efficient as there is no need to search through an image for section headers when loading.")

Svensson

Svensson (Ex-1110) at 1:11-15 ("The process of starting, or booting up, an electronic system having a programmable processor connected to one or more memory devices for storing program instructions, or code, and data is not as simple as it might seem at first glance.")

Svensson (Ex-1110) at 2:11-15 ("Moreover, it is necessary to determine which portions of the system must be loaded to memories visible to both host and slave processors and how the binary image to be loaded should be arranged for the bootloader to work together with the OS.")

Svensson (Ex-1110) at 3:49-4:8 ("FIG. 1 depicts such a multi-processor system 100 that includes a host processor 102 and a client processor 104. It will be appreciated that although FIG. 1 shows one client processor 104, more can be provided. It will also be appreciated that the host and client processors may be any programmable electronic processors. In the example depicted in FIG. 1, the processor 102 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 104 is shown as the CPU of a digital signal processor (DSP) device. The dashed line in FIG. 1 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 106. The memory 106 may be a ROM, a flash

memory, or other type of non-volatile memory device.

Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110. An intermediate storage area, indicated by the dashed line, is defined within the memory 108 as described in more detail below. The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths, between the CPUs and the memories. The ARM host CPU 102 can access the non-volatile memory 106 and the SARAM and DARAM 108 of the DSP, but not the DSP's XRAM 110, and the DSP slave CPU 104 can access all of the RAMs 108, 110. ")

Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 4:9-14 ("The SARAM and DARAM 108 can be loaded from the non-volatile memory 106 by the trivial "push" method. When code needs to be loaded to the XRAM 110 during boot, however, a bootloader solution is required because the XRAM 110 is invisible to, i.e., not accessible by, the CPU 102 and so boot code cannot be pushed to the XRAM 110.")

Svensson (Ex-1110) at 4:22-26 ("The first stage resets and holds the slave 104 in the reset state (Step 202) and pushes information (program instructions and/or data) (Step 204) in the usual way from the non-volatile memory 106 into the commonly visible memories 108.")
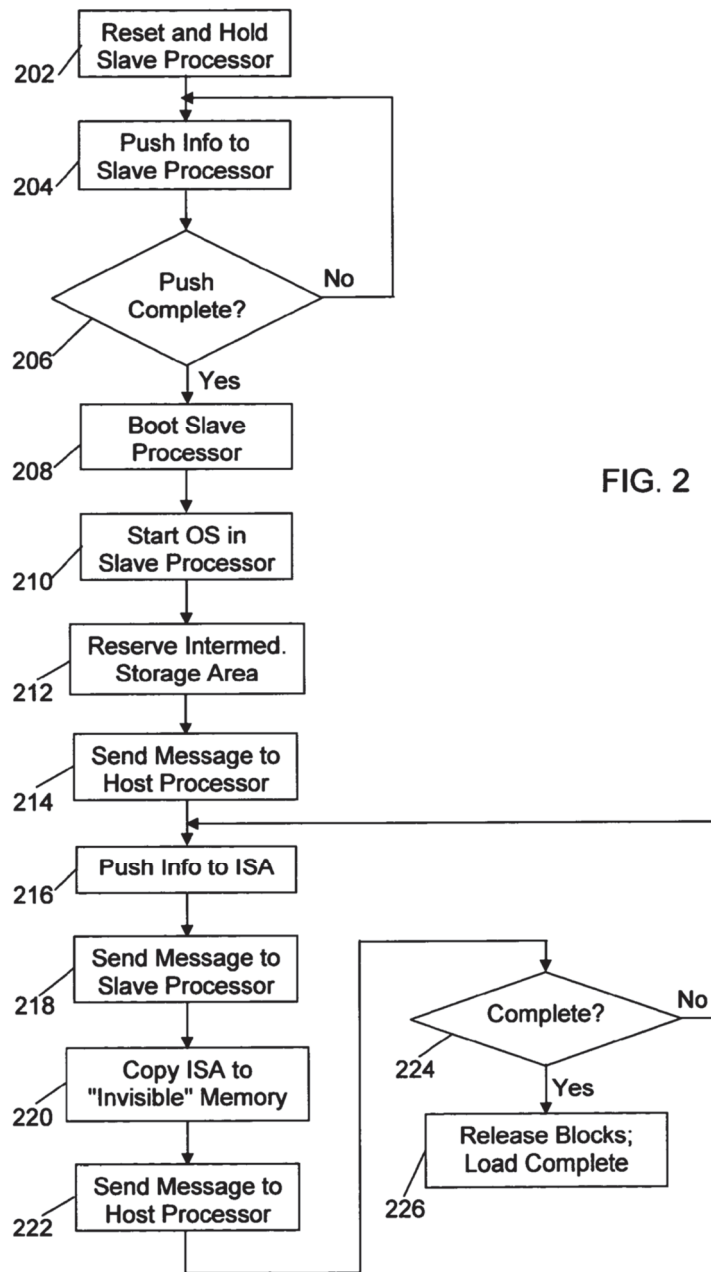
Svensson (Ex-1110), Figure 2:

FIG. 2

Svensson (Ex-1110) at 5:21-37 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110. The slave's idle process advantageously uses the established communication

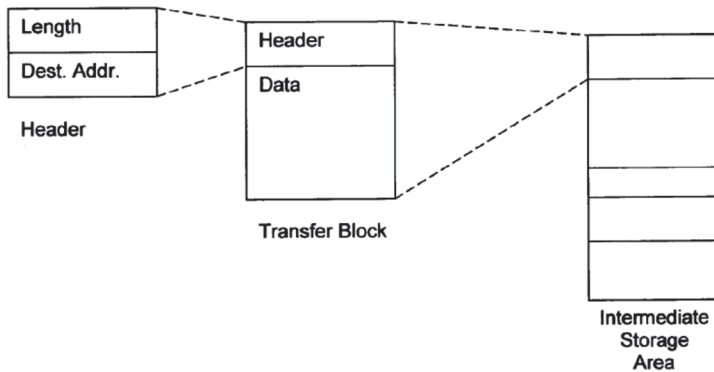|	mechanisms to send to the host (Step 214) information about the address and size or length of the intermediate storage area reserved in the previous step. After sending the information, which may be contained in one or more suitable messages, the slave blocks, awaiting a message from the host. While "blocked", the slave does not conduct any further loading activities until it receives the host's response.") Svensson (Ex-1110) at 5:53-6:25 ("The host now sends a message to the slave (Step 218) that indicates the intermediate storage area has been loaded and whether loading is finished or more code and/or data is available. This is the message the slave is waiting for. The host in turn now blocks, awaiting a message from the slave. The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading. The slave then sends a message to the host (Step 222) that indicates that the slave has copied the contents of the intermediate storage area. If there is more code and/or data to load (Step 224), this cycle of copying and messaging (Steps 216-224) can be repeated as many times as required. When the loading is finished, i.e., when no more information needs to be copied to the slave, the slave releases the blocking of processes that were blocked earlier, thereby allowing scheduling of code in its slave-private memory (Step 226). Loading is now complete. As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage area with a tag that contains the destination address and length of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address. As indicated by the dashed lines in FIG. 3, several such blocks may be concatenated in the intermediate storage area.") Svensson (Ex-1110), Figure 3:

FIG. 3

Svensson (Ex-1110) at 6:48-59 ("The artisan will understand the benefit of this splitting and concatenation of information into transfer blocks. Some kind of communication mechanism is required to perform the actual transfers of information between memories, and whatever the mechanism used, fewer large transfers are typically preferable to more small transfers. A kept-full intermediate storage area can make the most efficient use of the available bandwidth by advantageously minimizing overhead on the communications channel. Each message requires some amount of administration and administrative information, and so fewer messages means less overhead.

Svensson (Ex-1110) at 6:60-7:2 ("A good example of the benefit of block splitting and concatenation effect is DMA as the communication mechanism. DMA typically requires some setup overhead (i.e., it takes some time to set up), but then DMA is very efficient once it has been started because transfers can be carried out in minimal CPU cycles. In order to gain the greatest benefit from the use of DMA, the largest DMA transfer permitted by the hardware should be done every time. Thus, it is currently believed to be advantageous to set the size of the intermediate storage area to the maximum DMA block size.")

Svensson (Ex-1110) at 7:52-60 ("From this description, it will be understood that OS mechanisms are available to the slave part of the OS-friendly bootloader that is executed by the slave processor and that the slave can reuse existing OS-dependent code required for communication. Moreover, the OS-friendly bootloader uses loading resources (e.g., DMA) efficiently, with the host part automatically deciding when to switch from a first stage, or push mode, to a second stage, or bootloader mode.")

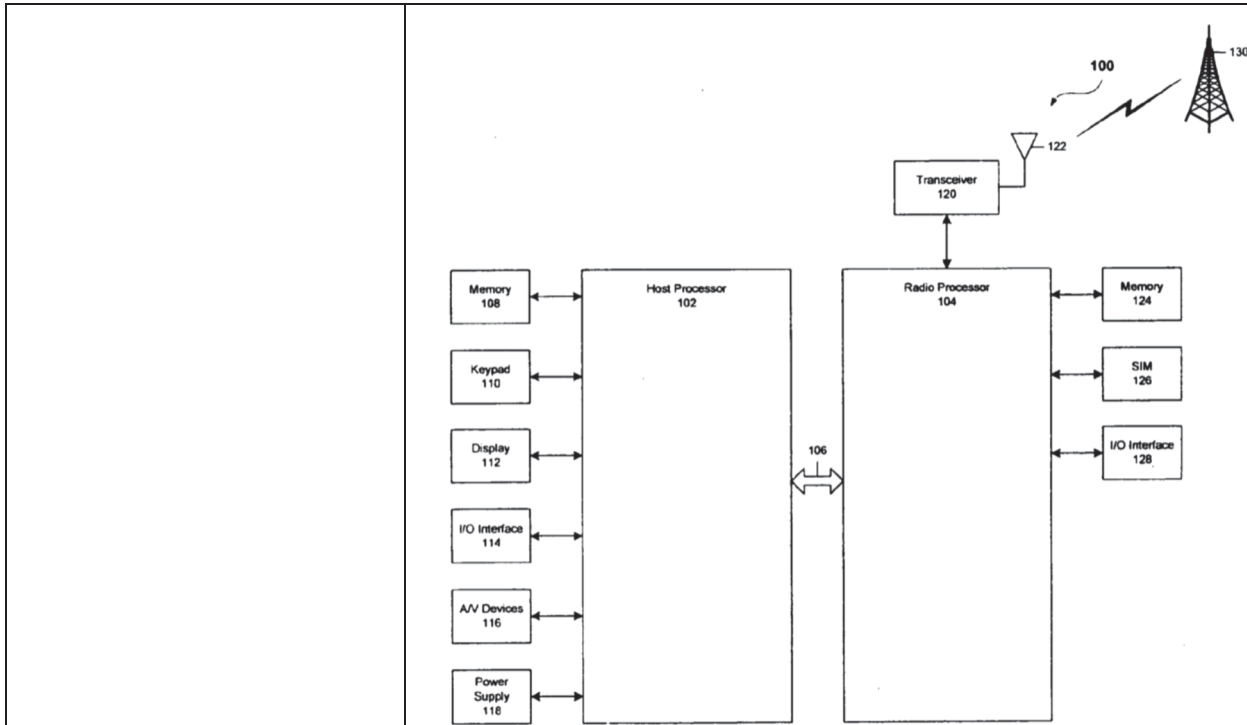|	
	Zhao Zhao (Ex-1113) at ¶32 ("As shown in the embodiment of FIG. 3, mobile computing device 100 may comprise a dual processor architecture including a host processor 102 and a radio processor 104 (e.g., a base band processor). The host processor 102 and the radio processor 104 may be arranged to communicate with each other using interfaces 106 such as one or more universal serial bus (USB) interfaces, micro-USB interfaces, universal asynchronous receiver-transmitter (UART) interfaces, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth.") Zhao (Ex-1113) at ¶33 ("Although embodiments of the dual processor architecture may be described as comprising the host processor 102 and the radio processor 104 for purposes of illustration, it is worthy to note that the dual processor architecture of the mobile computing device 100 may comprise additional processors, may be implemented as a dual- or multi-core chip with both host processor 102 and radio processor 104 on a single chip, etc.") Zhao (Ex-1113), Figure 3:

FIG. 3

Zhao (Ex-1113) at ¶34 ("In various embodiments, the host processor 102 may be implemented as a host central processing unit (CPU) using any suitable processor or logic device, such as a general purpose processor. The host processor 102 may comprise, or be implemented as, a chip multiprocessor (CMP), dedicated processor, embedded processor, media processor, input/output (I/O) processor, co-processor, a field programmable gate array (FPGA), a programmable logic device (PLD), or other processing device in alternative embodiments. In an exemplary embodiment, host processor 102 is an OMAP2, such as an OMAP2431 processor, manufactured by Texas Instruments, Inc.")

Zhao (Ex-1113) at ¶44 ("As mentioned above, the radio processor 104 may perform voice and/or data communication operations for the mobile computing device 100. For example, the radio processor 104 may be arranged to communicate voice information and/or data information over one or more assigned frequency bands of a wireless communication channel. In various embodiments, the radio processor 104 may be implemented as a communications processor using any suitable processor or logic device, such as a modem processor or

baseband processor. Although some embodiments may be described with the radio processor 104 implemented as a modem processor or baseband processor by way of example, it may be appreciated that the embodiments are not limited in this context. For example, the radio processor 104 may comprise, or be implemented as, a digital signal processor (DSP), media access control (MAC) processor, or any other type of communications processor in accordance with the described embodiments. Radio processor 104 may be any of a plurality of modems manufactured by Qualcomm, Inc.")

'949 patent

'949 patent (Ex-1101) at 1:37-41 ("Processors execute software code to perform operations. Processors may require some software code, commonly referred to as boot code, to be executed for hooting up. In a multi-processor system, each processor may require respective boot code for booting up.")

'949 patent (Ex-1101) at 1:41-44 ("As an example, in a smartphone device that includes an application processor and a modem processor, each of the processors may have respective boot code for booting up. ")

'949 patent (Ex-1101) at 1:45-51 ("A problem exists on a significant number of devices (such as smart phones) that incorporate multiple processors (e.g., a standalone application processor chip integrated with a separate modem processor chip). A flash/non-volatile memory component may be used for each of the processors, because each processor has non-volatile memory (e.g., persistent storage) of executable images and file systems.")

'949 patent (Ex-1101) at 2:3-13 ("For example, suppose a first processor in a multi-processor system is responsible for storing to its non-volatile memory boot code for one or more other processors in the system; wherein upon power-up the first processor is tasked with loading the respective boot code to the other processor(s), as opposed to such boot code residing in non-volatile memory of the other processor(s). In this type of system, the software (e.g., boot image) is downloaded from the first processor to the other processor(s) (e.g., to volatile memory of the other processor(s)), and thereafter the receiving processor(s) boots with the downloaded image.:)

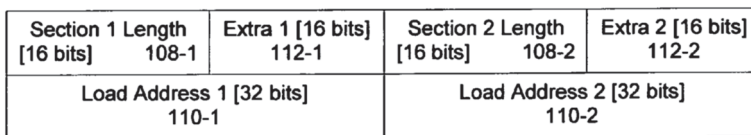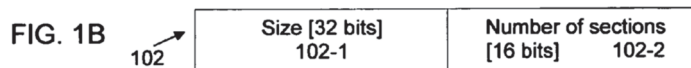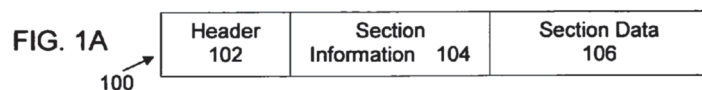'949 patent (Ex-1101) at 2:14-16 ("Often, the software image

	to be loaded is a binary multi-segmented image. For instance, the software image may include a header followed by multiple segments of code.")
	'949 patent (Ex-1101) at 2:42-45 ("The primary processor and its non-volatile memory that stores the boot image for a secondary processor may be implemented on a different chip than a chip on which the secondary processor is implemented.")
[10b] the image header and each data segment being received separately;	*See* claim limitation [10a]. Bauer Bauer (Ex-1109) at Abstract ("A data format includes a header, section information, and one or more sections. Each section includes binary data that is encoded independently of other sections, and the header and section information contains information about the sizes, load addresses, and encoding, e.g., encryption and/or compression, of the sections. The header and section information are arranged in an image having this format such that they are readable before the sections are processed. For example, the sections can be located in sequence after the header and the section information, in an order determined by their load addresses.") Bauer (Ex-1109) at ¶16 ("The new format for binary data described in this application is particularly useful in embedded systems as well as in other computer environments where efficiency is important. Greater efficiency in loading data can reduce response times in such systems, and space-efficient storage saves valuable memory.") Bauer (Ex-1109) at ¶27 ("The format described here includes a header, section information, and one or more sections. The section information contains the information for all sections, which is more advantageous than having each section include its own information, i.e., the information is concentrated rather than distributed across the sections. Furthermore, the section information contains information about the encoding of the sections.") Bauer (Ex-1109) at ¶28 ("The header and section information are arranged in an image having this format such that they are readable before the sections are processed.")

Bauer (Ex-1109) at ¶29 ("Other arrangements are possible, of course. It is important only that the header and section information can be read before the rest of an image. The locations of the header and section information can be anywhere in the image, provided it is possible to access the header and section information before the rest of the image.")

Bauer (Ex-1109) at ¶30 ("Information about the sections is located in a header and section information at, for example, the beginning of the image, and so the information about the sections can be retrieved before the sections are read.")

Bauer (Ex-1109) at ¶33 ("As depicted in FIG. 1B, the header 102 contains a size information element 102-1 that indicates the total size of the sections 106 (in bytes, for example). The size element 102-1 may advantageously be a 32-bit unsigned integer, for example, and such an element is suitable for images having section data up to a total of four gigabytes (GB) in size. The header 102 also contains a number-of-sections information element 102-2, which may advantageously be a 16-bit unsigned integer, for example. It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109), Figures 1A-C:

Bauer (Ex-1109) at ¶38 ("Having all section information entries 104 collected together in the image 100 advantageously simplifies system navigation through the image, and having all section data arranged in a sequence makes it possible to

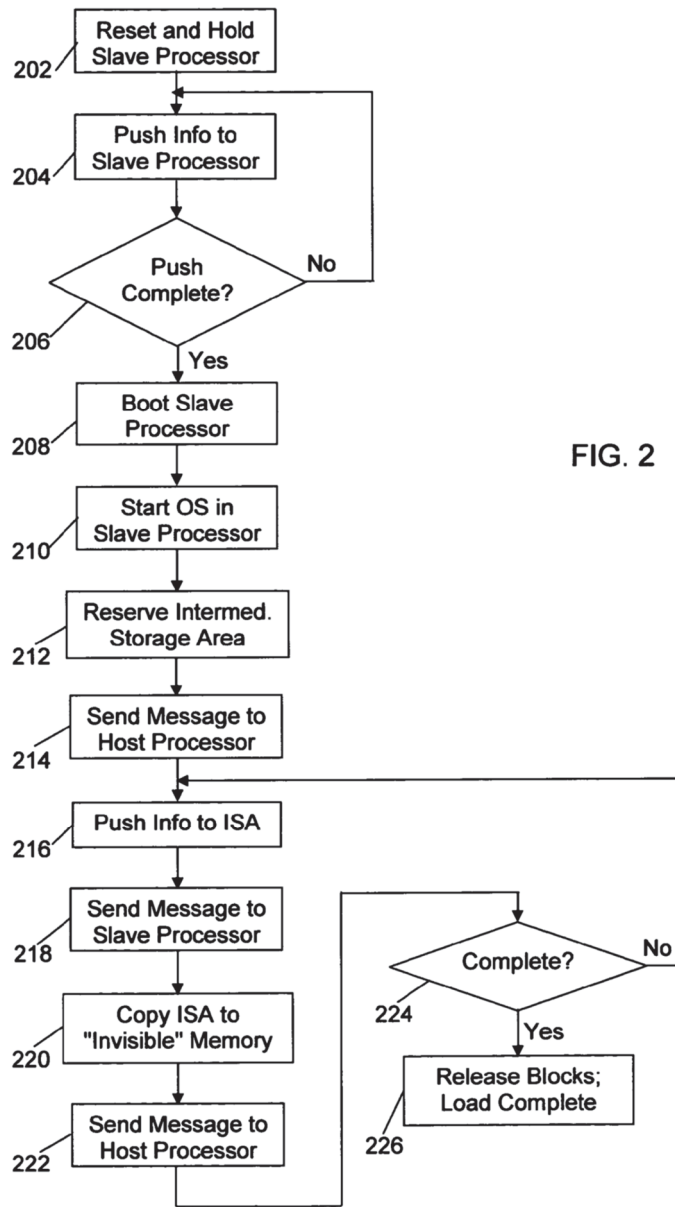|	
	optimize loading of the sections. For instance, it is simple to split or concatenate sections when they are adjacent in memory. The ability to split sections can be useful, for instance, when a DMA transfer is to be set up. As there is always a small overhead when setting up a DMA transfer, a DMA unit can be used in an efficient way by arranging the size of the data to be transferred to be equal or close to the block sizes used by the DMA unit. As sections can be located sequentially in an image 100, it is simple to split a section into several suitable pieces before downloading it.") Bauer (Ex-1109) at ¶43 ("Having information about the sections collected in the header 102 and section information 104 simplifies optimization in a number of circumstances, for instance, if sections are to be loaded into memory. The block 104 lists all sections, preferably in order of memory location, and this makes memory loading efficient as there is no need to search through an image for section headers when loading.") Bauer (Ex-1109) at ¶47 ("Converting a binary image, e.g., an image in COFF/ELF format, into an image having the format described in this application can be carried out in a number of ways, for example by a suitable post-linker conversion tool.") Svensson Svensson (Ex-1110) at 5:28-36 ("The slave's idle process advantageously uses the established communication mechanisms to send to the host (Step 214) information about the address and size or length of the intermediate storage area reserved in the previous step. After sending the information, which may be contained in one or more suitable messages, the slave blocks, awaiting a message from the host. While "blocked", the slave does not conduct any further loading activities until it receives the host's response.") Svensson (Ex-1110), Figure 2:

FIG. 2

Svensson (Ex-1110) at 5:53-59 ("On receipt of the slave's information, the second stage of the host bootloader fills the intermediate storage area with information (code and/or data) to be loaded into the slave's invisible memory (Step 216). Code and data is pushed to the intermediate storage area in the usual way because this area is memory that both processors can access, but the push is activated through the OS communication mechanisms.")

Svensson (Ex-1110) at 5:60-6:3 ("The host now sends a message to the slave (Step 218) that indicates the intermediate

storage area has been loaded and whether loading is finished or more code and/or data is available. This is the message the slave is waiting for. The host in turn now blocks, awaiting a message from the slave. The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading. The slave then sends a message to the host (Step 222) that indicates that the slave has copied the contents of the intermediate storage area.")

Svensson (Ex-1110) at 6:4-11 ("If there is more code and/or data to load (Step 224), this cycle of copying and messaging (Steps 216-224) can be repeated as many times as required. When the loading is finished, i.e., when no more information needs to be copied to the slave, the slave releases the blocking of processes that were blocked earlier, thereby allowing scheduling of code in its slave-private memory (Step 226). Loading is now complete.")

Svensson (Ex-1110) at 6:12-25 ("As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage area with a tag that contains the destination address and length of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address. As indicated by the dashed lines in FIG. 3, several such blocks may be concatenated in the intermediate storage area.")

Svensson (Ex-1110) at 6:26-33 ("The information (code and data) to be loaded can be arranged in many ways in the intermediate storage area and memories. Often the information is arranged as blocks of consecutive information that are to be loaded to different addresses, and thus an arbitrarily chosen size of the intermediate storage area may not match the sizes of all such blocks.")

Svensson (Ex-1110) at 6:37-43 ("This also means that a block should be split if it is larger than the remaining part of the intermediate storage area, and one part transferred to the intermediate storage area with the remaining part transferred in

the next block. Moreover, if a block is several times larger than the intermediate storage area, it may have to be split more than once.")

Svensson (Ex-1110) at 6:67-7:2 ("Thus, it is currently believed to be advantageous to set the size of the intermediate storage area to the maximum DMA block size.")

Kim

Kim (Ex-1112) at 4:7-12 ("FIG. 1 shows a loading system configuration of a conventional exchanger. As shown in FIG. 1, the loading system configuration of the conventional exchanger includes a system management processor 11, a hard disk drive 17, a plurality of port management assemblies 12, a plurality of main processors 13, a plurality of SSPs 14, a plurality of device processors 15, and a plurality of data control devices.")

Kim (Ex-1112), Figure 1:

FIG. 1
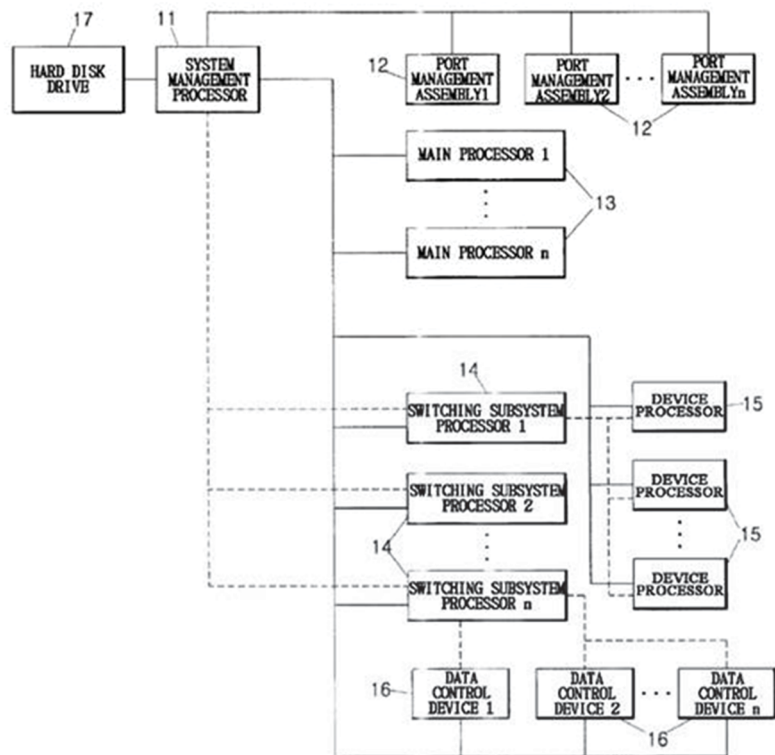
Kim (Ex-1112) at 4:13-14 ("Only the system management processor 11 is directly connected to the hard disk drive 17, which stores blocks needed to be loaded.")

Kim (Ex-1112) at 5:9-11 ("FIG. 3 is a flowchart showing a loading procedure of each of a system startup loader and a booter in a conventional exchange system. The loading procedure will be described in detail below with reference to FIG. 3.")

Kim (Ex-1112), Figure 3:

FIG. 3
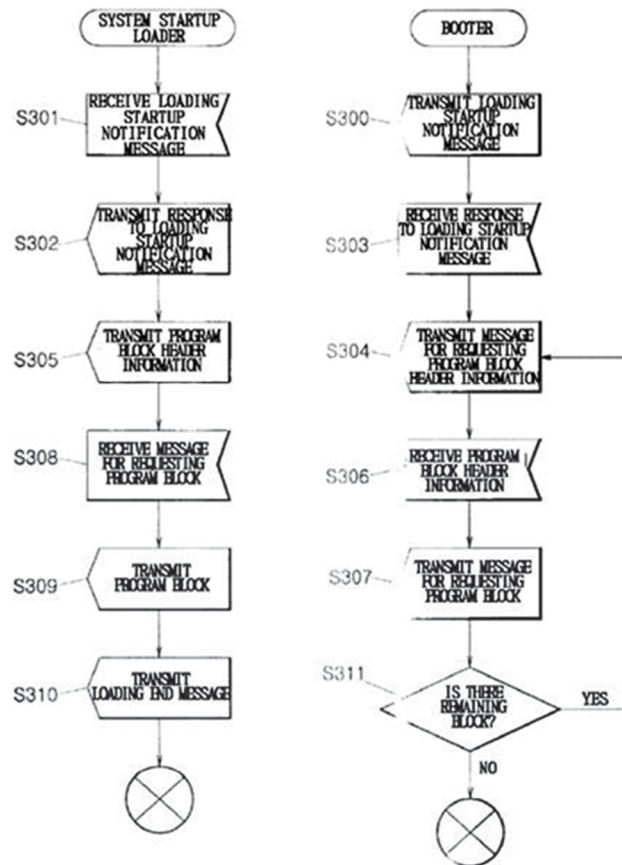
Kim (Ex-1112) at 5:12-6:5 ("A booter in a device transmits a loading startup notification message to the system startup loader in the system management processor (S300). When the loading startup notification message is received (S301), the system startup loader transmits a response message responding to the loading startup notification to the booter that has

	transmitted the loading startup notification (S302). When the response message responding to the loading startup notification is received (S303), the booter transmits a message for requesting program block header information needed for the loading to the system startup loader (S304). When the message is received, the system startup loader transmits the program block header information to the booter (S305). When the header information is received, the booter requests a program block corresponding to actual program content from the system startup loader (S307). The system startup loader receives the request from the booter (S308). The system startup loader transmits the program block to the booter (S309), and transmits a loading end message to the booter after the loading of one block is complete. When there is a block to be received, the booter returns to step S304 in which the booter transmits a message for requesting program block header information needed for the loading to the system startup loader. When there is no longer a block to be received, the booter ends the loading.")
[10c] processing, by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment;	*See* claim limitation [10a]. Bauer Bauer (Ex-1109) at Abstract ("A data format includes a header, section information, and one or more sections. Each section includes binary data that is encoded independently of other sections, and the header and section information contains information about the sizes, load addresses, and encoding, e.g., encryption and/or compression, of the sections. The header and section information are arranged in an image having this format such that they are readable before the sections are processed. For example, the sections can be located in sequence after the header and the section information, in an order determined by their load addresses.") Bauer (Ex-1109) at ¶17 ("The header contains a first information element that indicates a total size of the at least one section and a second information element that indicates a number of the sections. The section information includes a respective entry for each section, each entry including a third information element that indicates a length of the respective section and ***a fourth information element that indicates a load address of the respective section***. The at least one section includes data that is encoded independently of the header, section information, and other sections. The header and the

section information is arranged in the image such that the header and section information are readable before the at least one section.")

Bauer (Ex-1109) at ¶27 ("The format described here includes a header, section information, and one or more sections. The section information contains the information for all sections, which is more advantageous than having each section include its own information, i.e., the information is concentrated rather than distributed across the sections. Furthermore, the section information contains information about the encoding of the sections.")

Bauer (Ex-1109) at ¶28 ("Each section includes binary data that is encoded independently of other sections, and the header and section information contains information about the sizes, load addresses, and encoding, e.g., encryption and/or compression, of the sections. The header and section information are arranged in an image having this format such that they are readable before the sections are processed. For example, the sections can be located in sequence after the header and the section information, in an order determined by their load addresses.")

Bauer (Ex-1109) at ¶29 ("Other arrangements are possible, of course. It is important only that the header and section information can be read before the rest of an image. The locations of the header and section information can be anywhere in the image, provided it is possible to access the header and section information before the rest of the image. Thus, the location of the header must be predetermined, or at least known to the software reading the image, so that the software "knows" where to look for the header. The location of the section information may also be "known" to the software, or the header can indicate the location.")

Bauer (Ex-1109) at ¶30 ("Thus, it will be appreciated that the format described here, in contrast to prior data formats, supports individual coding of sections, where a section can contain any type of data, such as executable, binary, text, etc. Information about the sections is located in a header and section information at, for example, the beginning of the image, and so the information about the sections can be retrieved before the sections are read. Moreover, the format is a representation of a group of sections, coded independently and having minimal overhead, that is traversed sequentially in

reading the image. Images in this format can be optimized in different aspects, depending on the coding scheme or schemes applied in the sections.")

Bauer (Ex-1109) at ¶31 ("There are many possible applications of this format and its individually coded sections. For example, *an operating system memory manager can load and unload sections of memory according to images in this format*. It can also be used as a file format in which executable files are stored, and linkers and program loaders can be readily adapted to support (read, write, and interpret) the format. Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly. *One example of such a program loader is described in U.S. patent application Ser. No. 11/040,798 filed on Jan. 22, 2005, by M. Svensson et al. for "Operating-System-Friendly Bootloader*.")

Bauer (Ex-1109) at ¶32 ("FIG. 1A depicts a binary data image 100 in this file format, including a header 102, section information 104, and section data 106. The section data 106 includes the data of the one or more sections included in the image 100.")
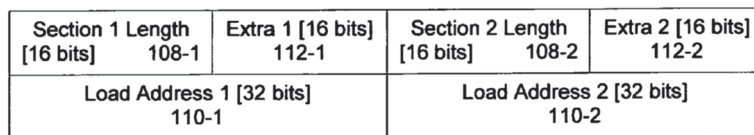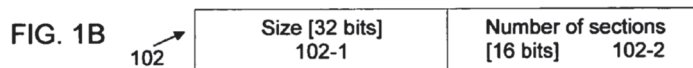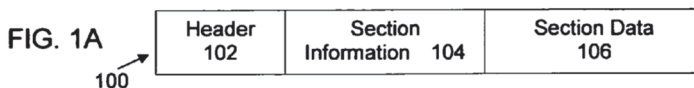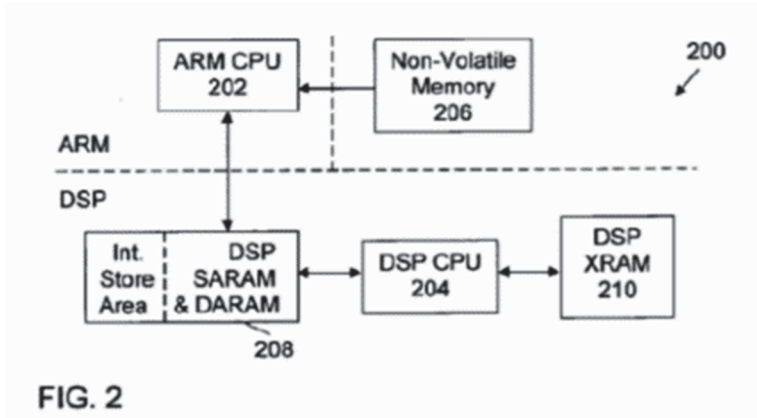
Bauer (Ex-1109), Figures 1A-C:

Bauer (Ex-1109) at ¶33 ("As depicted in FIG. 1B, the header 102 contains a size information element 102-1 that indicates the total size of the sections 106 (in bytes, for example)….The header 102 also contains a number-of-sections information

element 102-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶34 ("Each section in the section data 106 has a respective "section information" entry in the section information 104, and two such section information entries 104-1, 104-2 are depicted in FIG. 1C….The load addresses of the sections are indicated by address information elements 110-1, 110-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor 204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of a digital signal processor (DSP) device. The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.")

Bauer (Ex-1109) at ¶36 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C. The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) at ¶37 ("As depicted in FIG. 1A, the section information entry or entries 104 precede the data 106 of the section(s) in the image 100. The section data 106 is advantageously arranged in the image in a sequence, and it is preferable that the section data 106 as well as the section information entries 104 are arranged in order of the section load addresses 110, starting with the lowest address. It will be understood, however, that other orders are suitable, e.g., starting with the highest address, and that in general it is not necessary to order the section by their load addresses. The sections may be in an arbitrary order. As each section has a respective load address, the sections can appear in any order (e.g., by size, coding type, or whatever is suitable). It is currently believed, however, that the most efficient solution from a loading point of view is probably arranging the sections by load address in either descending or ascending order.")

Bauer (Ex-1109) at ¶38 ("Having all section information entries 104 collected together in the image 100 advantageously simplifies system navigation through the image, and having all section data arranged in a sequence makes it possible to optimize loading of the sections. For instance, it is simple to split or concatenate sections when they are adjacent in memory. The ability to split sections can be useful, for instance, when a DMA transfer is to be set up. As there is always a small overhead when setting up a DMA transfer, a DMA unit can be used in an efficient way by arranging the size of the data to be transferred to be equal or close to the block sizes used by the DMA unit. As sections can be located sequentially in an image 100, it is simple to split a section into

several suitable pieces before downloading it.")

Bauer (Ex-1109) at ¶47 ("Converting a binary image, e.g., an image in COFF/ELF format, into an image having the format described in this application can be carried out in a number of ways, for example by a suitable post-linker conversion tool. An exemplary method is illustrated by the flow chart in FIG. 3, and includes a step of identifying all of the sections of the image to be converted (step 302). Each identified section is individually coded according to a specified coding scheme (step 304). A header having the information described above is formed (step 306), and section information having information about the respective lengths, encodings, and load addresses of the identified sections is formed (step 308). In step 310, the identified sections are arranged in the image according to the section information, e.g., in increasing order of load address, etc., and the header and section information are arranged in the converted image such that they are readable in the converted image before the sections.")

Svensson

Svensson (Ex-1110) at 3:49-4:8 ("FIG. 1 depicts such a multi-processor system 100 that includes a host processor 102 and a client processor 104. It will be appreciated that although FIG. 1 shows one client processor 104, more can be provided. It will also be appreciated that the host and client processors may be any programmable electronic processors. In the example depicted in FIG. 1, the processor 102 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 104 is shown as the CPU of a digital signal processor (DSP) device. The dashed line in FIG. 1 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 106. The memory 106 may be a ROM, a flash memory, or other type of non-volatile memory device.

Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110. An intermediate storage area, indicated by the dashed line, is defined within the memory 108 as described in more detail below. The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths, between the CPUs and the memories. The ARM host CPU 102 can access the non-volatile memory 106

and the SARAM and DARAM 108 of the DSP, but not the DSP's XRAM 110, and the DSP slave CPU 104 can access all of the RAMs 108, 110.")
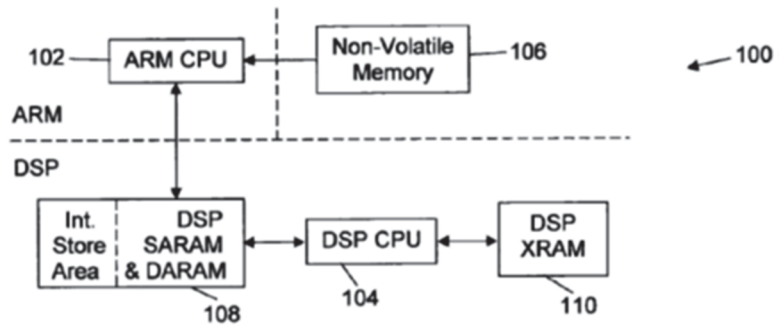
Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 5:21-28 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110.")
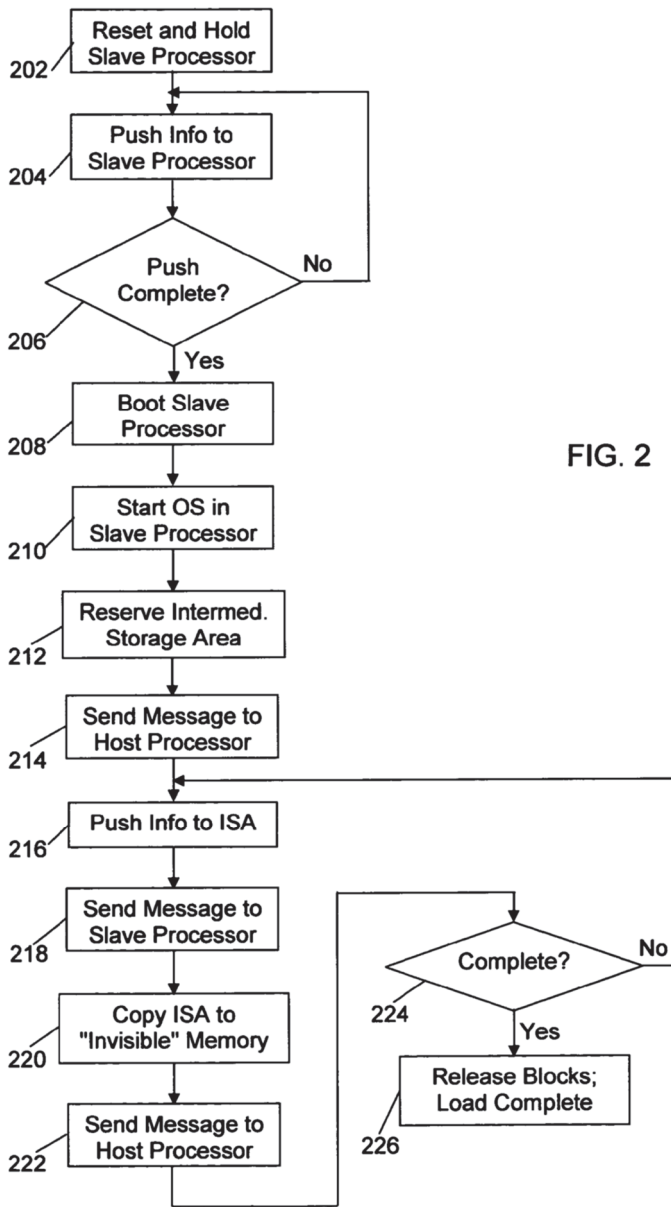
Svensson (Ex-1110), Figure 2:

FIG. 2

Svensson (Ex-1110) at 5:65-67 ("The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading.")

Svensson (Ex-1110) at 6:12-25 ("As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage area with a tag that contains the destination address and length

of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address. As indicated by the dashed lines in FIG. 3, several such blocks may be concatenated in the intermediate storage area.")
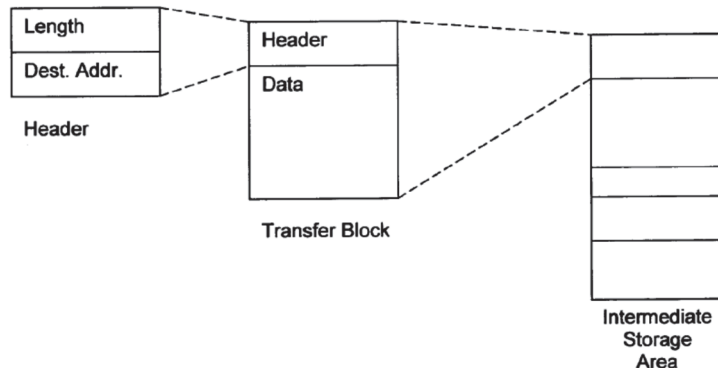
Svensson (Ex-1110), Figure 3:

FIG. 3

Svensson (Ex-1110) at 6:26-48 ("The information (code and data) to be loaded can be arranged in many ways in the intermediate storage area and memories. Often the information is arranged as blocks of consecutive information that are to be loaded to different addresses, and thus an arbitrarily chosen size of the intermediate storage area may not match the sizes of all such blocks. Still, it should be understood that the system will operate more efficiently when the intermediate storage area is always filled. This means that if the blocks to be loaded are smaller than this area, a transfer of several (smaller) blocks should be done at the same time. This also means that a block should be split if it is larger than the remaining part of the intermediate storage area, and one part transferred to the intermediate storage area with the remaining part transferred in the next block. Moreover, if a block is several times larger than the intermediate storage area, it may have to be split more than once. All of this splitting and concatenation is done in the host part of the OS-friendly bootloader in ways that are well known to computer scientists.")

Svensson (Ex-1110) at 6:49-59 ("The artisan will understand the benefit of this splitting and concatenation of information into transfer blocks. Some kind of communication mechanism

is required to perform the actual transfers of information between memories, and whatever the mechanism used, fewer large transfers are typically preferable to more small transfers. A kept-full intermediate storage area can make the most efficient use of the available bandwidth by advantageously minimizing overhead on the communications channel. Each message requires some amount of administration and administrative information, and so fewer messages means less overhead.")

Svensson (Ex-1110) at 6:60-7:2 ("A good example of the benefit of block splitting and concatenation effect is DMA as the communication mechanism. DMA typically requires some setup overhead (i.e., it takes some time to set up), but then DMA is very efficient once it has been started because transfers can be carried out in minimal CPU cycles. In order to gain the greatest benefit from the use of DMA, the largest DMA transfer permitted by the hardware should be done every time. Thus, it is currently believed to be advantageous to set the size of the intermediate storage area to the maximum DMA block size.
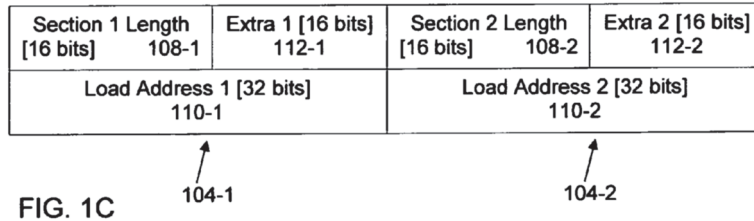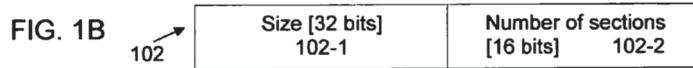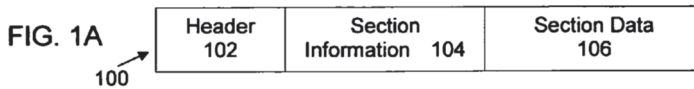
Svensson (Ex-1110) at 7:52-60 ("From this description, it will be understood that OS mechanisms are available to the slave part of the OS-friendly bootloader that is executed by the slave processor and that the slave can reuse existing OS-dependent code required for communication. Moreover, the OS-friendly bootloader uses loading resources (e.g., DMA) efficiently, with the host part automatically deciding when to switch from a first stage, or push mode, to a second stage, or bootloader mode.")

'949 patent

'949 patent (Ex-1101) at 2:14-22 ("Often, the software image to be loaded is a binary multi-segmented image. For instance, the software image may include a header followed by multiple segments of code. When software images are loaded, from an external device (e.g., from another processor) onto a target device (e.g., a target processor) there may be an intermediate step where the binary multi-segmented image is transferred into the system memory and then later transferred into target locations by the boot loader.")

'949 patent (Ex-1101) at 2:45-54 ("Thus, in order to transfer the data from the primary processor's non-volatile memory to the secondary processor (e.g., to the secondary processor's

	volatile memory), a packet-based communication may be employed, wherein a packet header is included in each packet communicated to the secondary processor. The packets are stored in an intermediate buffer, and some processing of the received packets is then required for that data to be stored where it needs to go (e.g., within the secondary processor's volatile memory).")
[10d] receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment; and	*See* claim limitation [10a] regarding the feature "receiving … via the inter-chip communication bus." *See* claim limitation [10b]. <u>Bauer</u> Bauer (Ex-1109) at ¶11 ("Also known are program downloading methods for use in data processing systems, integrating non-program information and program information into an executable file that is used by a host processor to download the program to a selected co-processor.") Bauer (Ex-1109) at ¶18 ("In another aspect of this invention, there is provided a computer-readable medium containing a data image for loading into a memory in a processor system.") Bauer (Ex-1109) at ¶32 ("FIG. 1A depicts a binary data image 100 in this file format, including a header 102, section information 104, and section data 106. The section data 106 includes the data of the one or more sections included in the image 100.") Bauer (Ex-1109), Figures 1A-C:

FIG. 1A

| Header 102 | Section Information 104 | Section Data 106 |

100

FIG. 1B

102

| Size [32 bits] 102-1 | Number of sections [16 bits] 102-2 |

| Section 1 Length [16 bits] 108-1 | Extra 1 [16 bits] 112-1 | Section 2 Length [16 bits] 108-2 | Extra 2 [16 bits] 112-2 |
| Load Address 1 [32 bits] 110-1 | | Load Address 2 [32 bits] 110-2 | |

FIG. 1C

104-1   104-2

Bauer (Ex-1109) at ¶33 ("As depicted in FIG. 1B, the header 102 contains a size information element 102-1 that indicates the total size of the sections 106 (in bytes, for example)….The header 102 also contains a number-of-sections information element 102-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶34 ("Each section in the section data 106 has a respective "section information" entry in the section information 104, and two such section information entries 104-1, 104-2 are depicted in FIG. 1C….The load addresses of the sections are indicated by address information elements 110-1, 110-2,…It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor 204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of

a digital signal processor (DSP) device. ***The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory*** 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.")
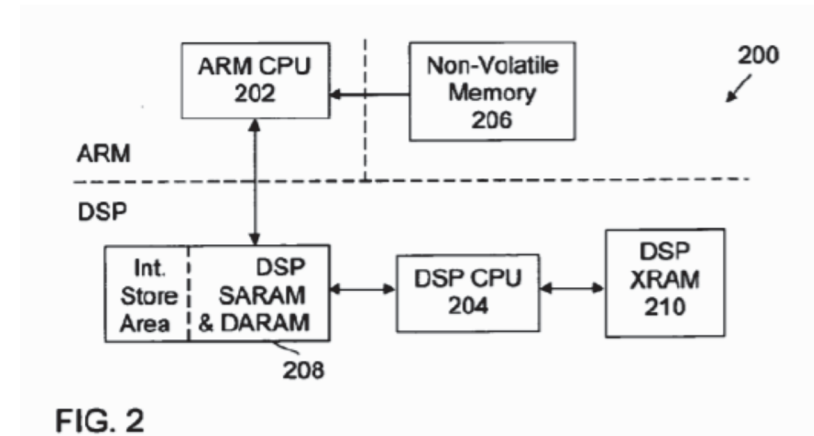
Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) ¶36 ("***Most commercially available DSP devices include on-chip memories***, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. ***The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C.*** The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Svensson

Svensson (Ex-1110) at 3:54-58 ("In the example depicted in FIG. 1, the processor 102 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 104 is shown as the CPU of a digital signal processor (DSP) device.")

Svensson (Ex-1110) at 3:64-4:5 ("Most commercially available

DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110. An intermediate storage area, indicated by the dashed line, is defined within the memory 108 as described in more detail below. The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths, between the CPUs and the memories.")
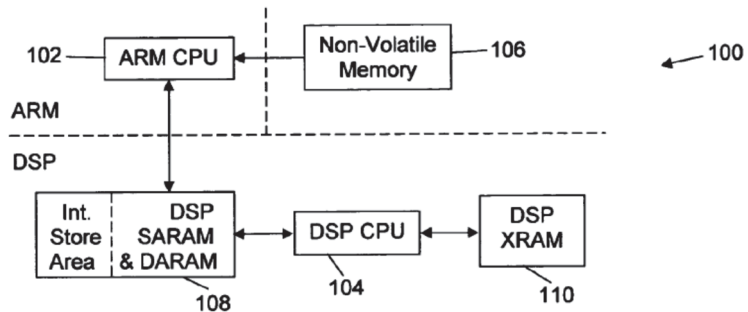
Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 4:9-10 ("The SARAM and DARAM 108 can be loaded from the non-volatile memory 106 by the trivial "push" method.")

Svensson (Ex-1110) at 4:22-26 ("The first stage resets and holds the slave 104 in the reset state (Step 202) and pushes information (program instructions and/or data) (Step 204) in the usual way from the non-volatile memory 106 into the commonly visible memories 108.")

Svensson (Ex-1110) at 5:21-23 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110.")

Svensson (Ex-1110) at 6:12-23 ("As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage

area with a tag that contains the destination address and length of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address.")
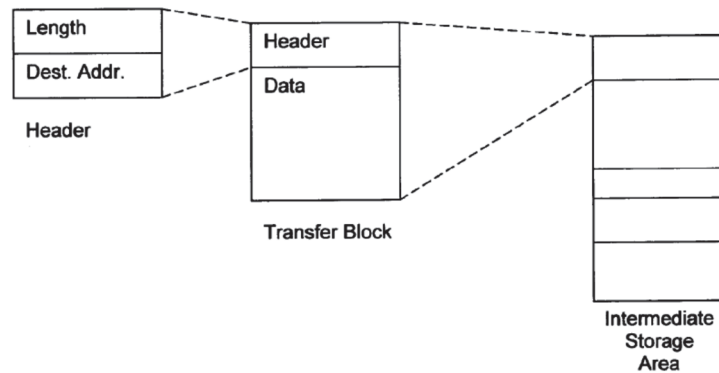
Svensson (Ex-1110), Figure 3:

FIG. 3

'949 patent

'949 patent (Ex-1101) at 1:45-48 ("A problem exists on a significant number of devices (such as smart phones) that incorporate multiple processors (e.g., a standalone application processor chip integrated with a separate modem processor chip).")
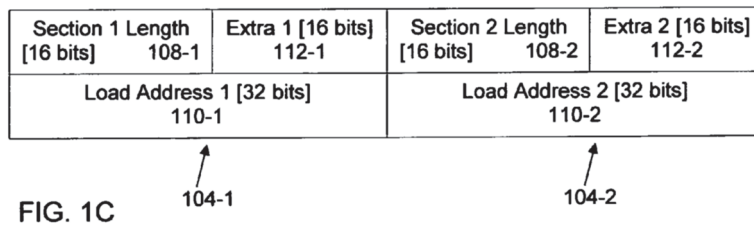
'949 patent (Ex-1101) at 2:3-13 ("In some multi-processor systems, software may be required to be loaded to one processor from another processor. For example, suppose a first processor in a multi-processor system is responsible for storing to its non-volatile memory boot code for one or more other processors in the system; wherein upon power-up the first processor is tasked with loading the respective boot code to the other processor(s), as opposed to such boot code residing in non-volatile memory of the other processor(s). In this type of system, the software (e.g., boot image) is downloaded from the first processor to the other processor(s) (e.g., to volatile memory of the other processor(s)), and thereafter the receiving processor(s) boots with the downloaded image.")

'949 patent (Ex-1101) at 2:14-22 ("Often, the software image to be loaded is a binary multi-segmented image. For instance,

	the software image may include a header followed by multiple segments of code. When software images are loaded, from an external device (e.g., from another processor) onto a target device (e.g., a target processor) there may be an intermediate step where the binary multi-segmented image is transferred into the system memory and then later transferred into target locations by the boot loader.")
	'949 patent (Ex-1101) at 2:42-45 ("The primary processor and its non-volatile memory that stores the boot image for a secondary processor may be implemented on a different chip than a chip on which the secondary processor is implemented.")
[10e] scatter loading, by the secondary processor, each data segment [directly] to the determined at least one location within the system memory, and each data segment being scatter loaded based at least in part on the processed image header.	Bauer Bauer (Ex-1109) at ¶17 ("The header contains a first information element that indicates a total size of the at least one section and a second information element that indicates a number of the sections. The section information includes a respective entry for each section, each entry including a third information element that indicates a length of the respective section and *a fourth information element that indicates a load address of the respective section*. The at least one section includes data that is encoded independently of the header, section information, and other sections. The header and the section information is arranged in the image such that the header and section information are readable before the at least one section.") Bauer (Ex-1109) at ¶31 (" There are many possible applications of this format and its individually coded sections. For example, *an operating system memory manager can load and unload sections of memory according to images in this format*. It can also be used as a file format in which executable files are stored, and linkers and program loaders can be readily adapted to support (read, write, and interpret) the format. Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly. *One example of such a program loader is described in U.S. patent application Ser. No. 11/040,798 filed on Jan. 22, 2005, by M. Svensson et al. for "Operating-System-Friendly Bootloader*.") Bauer (Ex-1109) at ¶32 ("FIG. 1A depicts a binary data image 100 in this file format, including a header 102, section

information 104, and section data 106. The section data 106 includes the data of the one or more sections included in the image 100.")

Bauer (Ex-1109), Figures 1A-C:

Bauer (Ex-1109) at ¶33 ("As depicted in FIG. 1B, the header 102 contains a size information element 102-1 that indicates the total size of the sections 106 (in bytes, for example)....The header 102 also contains a number-of-sections information element 102-2,...It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶34 ("Each section in the section data 106 has a respective "section information" entry in the section information 104, and two such section information entries 104-1, 104-2 are depicted in FIG. 1C....The load addresses of the sections are indicated by address information elements 110-1, 110-2,...It will be understood that other forms of these information elements can be used instead of the examples set forth here.")

Bauer (Ex-1109) at ¶36 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. The arrows in FIG. 2 indicate access paths, e.g., busses and direct

memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C. The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) at ¶37 ("As depicted in FIG. 1A, the section information entry or entries 104 precede the data 106 of the section(s) in the image 100. The section data 106 is advantageously arranged in the image in a sequence, and it is preferable that the section data 106 as well as the section information entries 104 are arranged in order of the section load addresses 110, starting with the lowest address. It will be understood, however, that other orders are suitable, e.g., starting with the highest address, and that in general it is not necessary to order the section by their load addresses. The sections may be in an arbitrary order. As each section has a respective load address, the sections can appear in any order (e.g., by size, coding type, or whatever is suitable). It is currently believed, however, that the most efficient solution from a loading point of view is probably arranging the sections by load address in either descending or ascending order.")

Bauer (Ex-1109) at ¶38 ("Having all section information entries 104 collected together in the image 100 advantageously simplifies system navigation through the image, and having all section data arranged in a sequence makes it possible to optimize loading of the sections. For instance, it is simple to split or concatenate sections when they are adjacent in memory. The ability to split sections can be useful, for

instance, when a DMA transfer is to be set up. As there is always a small overhead when setting up a DMA transfer, a DMA unit can be used in an efficient way by arranging the size of the data to be transferred to be equal or close to the block sizes used by the DMA unit. As sections can be located sequentially in an image 100, it is simple to split a section into several suitable pieces before downloading it.")

Svensson

Svensson (Ex-1110) at 3:64-4:8 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110. An intermediate storage area, indicated by the dashed line, is defined within the memory 108 as described in more detail below. The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths, between the CPUs and the memories. The ARM host CPU 102 can access the non-volatile memory 106 and the SARAM and DARAM 108 of the DSP, but not the DSP's XRAM 110, and the DSP slave CPU 104 can access all of the RAMs 108, 110.")
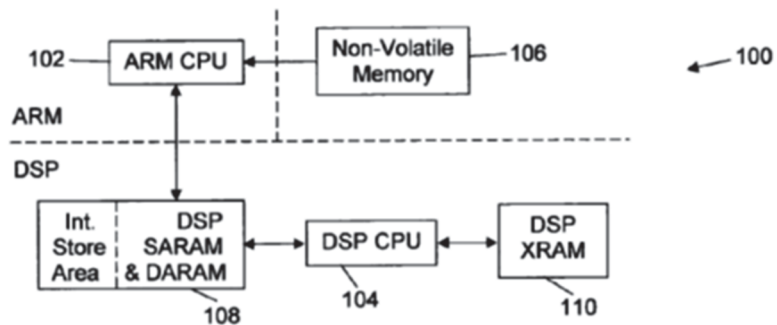
Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 5:21-28 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110.")

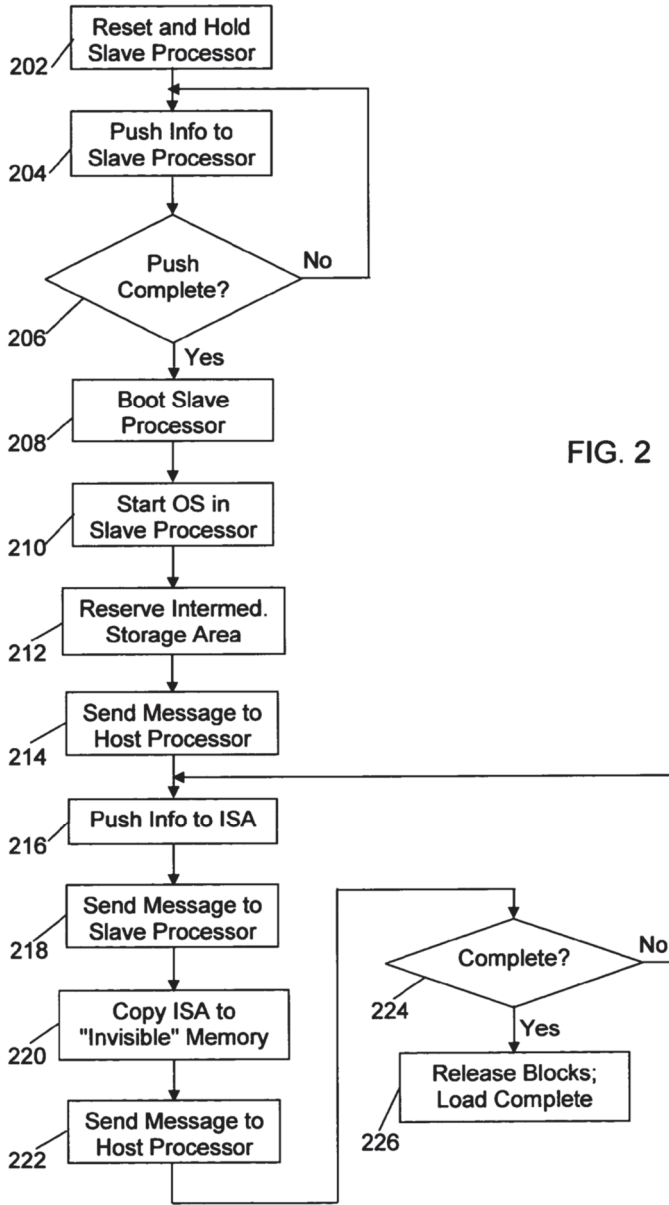Svensson (Ex-1110), Figure 2:

FIG. 2

Svensson (Ex-1110) at 5:65-67 ("The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading.")

Svensson (Ex-1110) at 6:12-25 ("As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage

area with a tag that contains the destination address and length of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address. As indicated by the dashed lines in FIG. 3, several such blocks may be concatenated in the intermediate storage area.")
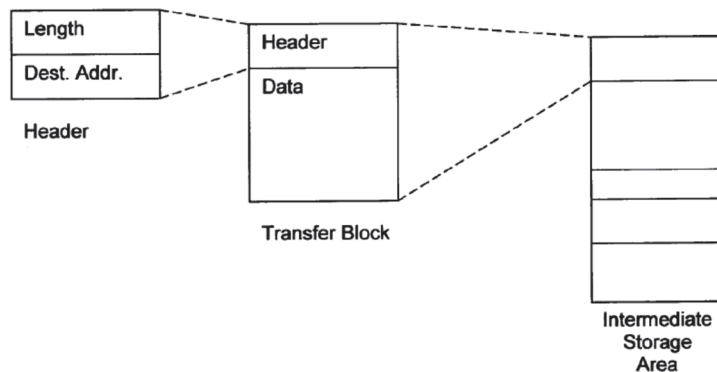
Svensson (Ex-1110), Figure 3:

FIG. 3

Svensson (Ex-1110) at 6:26-48 ("The information (code and data) to be loaded can be arranged in many ways in the intermediate storage area and memories. Often the information is arranged as blocks of consecutive information that are to be loaded to different addresses, and thus an arbitrarily chosen size of the intermediate storage area may not match the sizes of all such blocks. Still, it should be understood that the system will operate more efficiently when the intermediate storage area is always filled. This means that if the blocks to be loaded are smaller than this area, a transfer of several (smaller) blocks should be done at the same time. This also means that a block should be split if it is larger than the remaining part of the intermediate storage area, and one part transferred to the intermediate storage area with the remaining part transferred in the next block. Moreover, if a block is several times larger than the intermediate storage area, it may have to be split more than once. All of this splitting and concatenation is done in the host part of the OS-friendly bootloader in ways that are well known to computer scientists.")

Svensson (Ex-1110) at 6:49-59 ("The artisan will understand the benefit of this splitting and concatenation of information

into transfer blocks. Some kind of communication mechanism is required to perform the actual transfers of information between memories, and whatever the mechanism used, fewer large transfers are typically preferable to more small transfers. A kept-full intermediate storage area can make the most efficient use of the available bandwidth by advantageously minimizing overhead on the communications channel. Each message requires some amount of administration and administrative information, and so fewer messages means less overhead.")

Svensson (Ex-1110) at 6:60-7:2 ("A good example of the benefit of block splitting and concatenation effect is DMA as the communication mechanism. DMA typically requires some setup overhead (i.e., it takes some time to set up), but then DMA is very efficient once it has been started because transfers can be carried out in minimal CPU cycles. In order to gain the greatest benefit from the use of DMA, the largest DMA transfer permitted by the hardware should be done every time. Thus, it is currently believed to be advantageous to set the size of the intermediate storage area to the maximum DMA block size.

Svensson (Ex-1110) at 7:52-60 ("From this description, it will be understood that OS mechanisms are available to the slave part of the OS-friendly bootloader that is executed by the slave processor and that the slave can reuse existing OS-dependent code required for communication. Moreover, the OS-friendly bootloader uses loading resources (e.g., DMA) efficiently, with the host part automatically deciding when to switch from a first stage, or push mode, to a second stage, or bootloader mode.")

'949 patent (Ex-1101) at 2:35-41 ("Thus, where an intermediate buffer is used, the data being downloaded from a primary processor to a secondary processor is copied into the intermediate buffer. In this way, the buffer is used to receive part of the image data from the primary processor, and from the buffer the image data may be scattered into the memory (e.g., volatile memory) of the secondary processor.")

'949 patent

'949 patent (Ex-1101) at 9:12-15 ("As shown in FIG. 3, the image segments are not necessarily placed into consecutive locations within the secondary processor's system memory 305. Instead, the segments may be spread out in different locations

	of the memory.")
	'949 patent (Ex-1101) at 9:21-41 ("The image header is loaded from the primary processor 301 to scatter loader controller 304 of secondary processor 302. That image header provides information as to where the data segments are to be located in the system memory 305. The scatter loader controller 304 accordingly transfers the image segments directly into their respective target locations in the secondary processor's system memory 305. That is, once the secondary processor's CPU processes the image header in its memory 305 and programs the scatter loader controller 304, the scatter loader controller 304 knows exactly where the image segments need to go within the secondary processor's system memory 305, and thus the hardware scatter loader controller 304 is then programmed accordingly to transfer the data segments directly into their target destinations. In the example of FIG. 3, the scatter loader controller 304 receives the image segments and scatters them to different locations in the system memory 305. In one aspect, the executable software image is loaded into the system memory of the secondary processor without an entire executable software image being stored in the hardware buffer of the secondary processor.")
	Bauer + Svensson + Kim
11. The method of claim 10 further comprising booting the secondary processor using the executable software image.	*See* claim 10. *See* claim limitation [10a]. Bauer Bauer (Ex-1109) at ¶12 ("Also known are methods for compressing and recovering binary execution files; image loading program storage media for loaders of operating systems, which load and map executable images into memory based on file formats of images; and executable file protection and execution methods involving incorporating protection descriptors into protected executable files and providing to interpreters for unprotecting and executing protected files.") Bauer (Ex-1109) at ¶31 ("For example, an operating system memory manager can load and unload sections of memory according to images in this format. It can also be used as a file format in which executable files are stored, and linkers and program loaders can be readily adapted to support (read, write,

and interpret) the format. Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly.")

Svensson

Svensson (Ex-1110) at 1:17-27 ("The traditional ways of loading program code and data to a bare system are either by "pushing" the code and data into the system's random-access memory (RAM) directly or by using a bootloader. The bootloader, which is sometimes called a boot loader or a bootstrap loader, is a set of instructions (i.e., program code, sometimes called "boot code") that can be either "pushed" into the system's RAM or loaded into the RAM from a non-volatile memory, such as read-only memory (ROM). In its execution by the processor, the bootloader then "drags" in the rest of the code and data and starts the system.")

Svensson (Ex-1110) at 4:9-14 ("The SARAM and DARAM 108 can be loaded from the non-volatile memory 106 by the trivial "push" method. When code needs to be loaded to the XRAM 110 during boot, however, a bootloader solution is required because the XRAM 110 is invisible to, i.e., not accessible by, the CPU 102 and so boot code cannot be pushed to the XRAM 110.")

Svensson (Ex-1110) at 4:20-37 ("The host part of the OS-friendly bootloader may be considered as including two stages or modes of operation. The first stage resets and holds the slave 104 in the reset state (Step 202) and pushes information (program instructions and/or data) (Step 204) in the usual way from the non-volatile memory 106 into the commonly visible memories 108. The information pushed into these memories is mainly the bootloader, the OS, and any necessary start-up code for the OS. It should be appreciated that an application or applications or parts thereof may also be pushed into these memories at start-up and may start executing during the loading of the "external" memory 110. When this "push" is finished (Step 206), the slave 104 is allowed to boot (Step 208) and to start up the OS (e.g., it is released from the reset state) and its normal communication mechanisms (Step 210). The host part then awaits a message from the slave, which initiates operation of its second stage as described in more detail below.")

Svensson (Ex-1110) at 4:38-5:2 ("The slave part of the OS-friendly bootloader that is loaded ("pushed" by the host part's first stage) into the commonly visible memories 108 starts the operating system, carrying out the following operations (Step 210). First, interrupt handlers are created. The code for the interrupt handlers must be located in the memory that is already loaded because an interrupt may occur at any time. Second, data structures (e.g., process control blocks and stacks) of common processes, i.e., processes that run in both the host and the slave, are created. It should be understood that since these common processes have not yet executed, their code may be loaded at a later time and may very well be located in "external" memory visible only to the slave, e.g., XRAM 110. Third, the system idle process is created. The code for the idle process must be located in the memory that is already loaded because the idle process is the process selected to run by the OS if there is nothing useful to do. Fourth, the scheduling of at least all processes residing in, i.e., having program code or data located in, the "external" memory 110 is blocked. Execution of processes residing in the "internal" memory can thus advantageously start or continue in parallel with the loading of the "external" memory as noted above. It is also possible to stop scheduling all processes except the idle process, but this is not necessary. Making this blocking the last thing done before the OS scheduler switches on ensures that the code in these processes will not run when the scheduler releases. Finally, the OS scheduler is released, which allows the OS to start executing code and scheduling processes. It will be understood that since at least all external-memory-process scheduling was blocked, all that the OS can now do is schedule interrupts and the idle process.")

Svensson (Ex-1110) at 5:3-20 ("At this point, the slave 104 is partly up and running. The slave part of the OS-friendly bootloader has been loaded, and the slave's idle process is executing. The slave's OS can schedule and execute code in response to interrupts and can schedule the idle process and any unblocked processes having code residing in internal memory. OS mechanisms for which all code and data accesses are in memory that has already been loaded (SARAM and DARAM 108, in this example) are available, including the usual communication mechanisms. These OS communication mechanisms, being high-level abstractions of DMA, shared memory, and structured registers, are more capable than simple semaphores and enable the host processor to communicate efficiently with a processor (the slave) that has not completely

| | started, which is to say a processor that is executing mainly only the OS, interrupt services, and processes residing in "internal" RAM.")

Svensson (Ex-1110) at 5:21-37 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110. The slave's idle process advantageously uses the established communication mechanisms to send to the host (Step 214) information about the address and size or length of the intermediate storage area reserved in the previous step. After sending the information, which may be contained in one or more suitable messages, the slave blocks, awaiting a message from the host. While "blocked", the slave does not conduct any further loading activities until it receives the host's response.")

Svensson (Ex-1110) at 5:38-52 ("It will be understood that whether the slave's OS acts on an interrupt at this stage depends on the nature of the interrupt. Since many OS mechanisms (like those used to communicate with the host, for example) rely on interrupts, and it cannot be known in advance when an interrupt will occur, all interrupt code must have been loaded into "internal" memory. In that respect, interrupts are served during the second stage of the bootloading. Nevertheless, if an interrupt is to trigger a chain of events such as processes starting to do some data processing and the code or data for those processes are located or will be located in "external" memory, the interrupt is blocked and the interrupt service puts the request in the "in-queue" of that process so that the request will be served after booting has finished and that process can execute.")

Svensson (Ex-1110) at 5:53-59 ("On receipt of the slave's information, the second stage of the host bootloader fills the intermediate storage area with information (code and/or data) to be loaded into the slave's invisible memory (Step 216). Code and data is pushed to the intermediate storage area in the usual way because this area is memory that both processors can access, but the push is activated through the OS communication mechanisms.") |
|---|---|

Svensson (Ex-1110) at 5:60-6:3 ("The host now sends a message to the slave (Step 218) that indicates the intermediate storage area has been loaded and whether loading is finished or more code and/or data is available. This is the message the slave is waiting for. The host in turn now blocks, awaiting a message from the slave. The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading. The slave then sends a message to the host (Step 222) that indicates that the slave has copied the contents of the intermediate storage area.")

Svensson (Ex-1110) at 6:4-11 ("If there is more code and/or data to load (Step 224), this cycle of copying and messaging (Steps 216-224) can be repeated as many times as required. When the loading is finished, i.e., when no more information needs to be copied to the slave, the slave releases the blocking of processes that were blocked earlier, thereby allowing scheduling of code in its slave-private memory (Step 226). Loading is now complete.")

Svensson (Ex-1110) at 6:6-11 ("When the loading is finished, i.e., when no more information needs to be copied to the slave, the slave releases the blocking of processes that were blocked earlier, thereby allowing scheduling of code in its slave-private memory (Step 226). Loading is now complete.")

Svensson (Ex-1110) at 8:12-16 ("The slave is booted before all code is loaded, but code that is linked to host-inaccessible memory is not run until it is loaded with the help of code that is linked to the slave processors host-accessible memory.")

'949 patent

'949 patent (Ex-1101) at 2:1-13 ("In some multi-processor systems, software may be required to be loaded to one processor from another processor. For example, suppose a first processor in a multi-processor system is responsible for storing to its non-volatile memory boot code for one or more other processors in the system; wherein upon power-up the first processor is tasked with loading the respective boot code to the other processor(s), as opposed to such boot code residing in non-volatile memory of the other processor(s). In this type of system, the software (e.g., boot image) is downloaded from the first processor to the other processor(s) (e.g., to volatile memory of the other processor(s)), and thereafter the receiving

	processor(s) boots with the downloaded image.")
	'949 patent (Ex-1101) at 5:20-55 ("Upon system power-up, the modem processor 110 executes its primary boot loader (PBL) from the hardware boot ROM 126 (small read-only on-chip memory). The modem PBL may be adapted to download the modem executables 120 from the application processor 104. That is, the modem executable image 120 (initially stored in the primary non-volatile memory 106) is requested by the modem processor 110 from the application processor 104. The application processor 104 retrieves the modem executable image 120 and provides it to the modem processor 110 via an inter-processor communication bus 134 (e.g., inter-chip communication bus). The modem processor 110 stores the modem executable image 132 directly into the modem processor RAM (Random Access Memory) 112 to the final destination without copying the data into a temporary buffer in the modem processor RAM 112. The inter-processor communication bus 134 may be, for example, a HSIC bus (USB-based High Speed Inter-Chip), an HSI bus (MIPI High Speed Synchronous Interface), a SDIO bus (Secure Digital I/O interface), a UART bus (Universal Asynchronous Receiver/Transmitter), an SPI bus (Serial Peripheral Interface), an I2C bus (Inter-Integrated Circuit), or any other hardware interface suitable for inter-chip communication available on both the modem processor 110 and the application processor 104.")
	'949 patent (Ex-1101) at claim 6 ("6. The multi-processor system of claim 1, in which the secondary processor further comprises a non-volatile memory storing a boot loader that initiates transfer of the executable software image for the secondary processor.")
Bauer + Svensson + Kim	
12. The method of claim 10 further comprising loading the executable software image directly from a hardware buffer to the system memory of the secondary processor without copying data between system memory locations.	*See* claim 10. See claim limitations [10b]-[10e]. Bauer Bauer (Ex-1109) at ¶16 ("The new format for binary data described in this application is particularly useful in embedded systems as well as in other computer environments where efficiency is important. Greater efficiency in loading data can reduce response times in such systems, and space-efficient

storage saves valuable memory.")

Bauer (Ex-1109) at ¶27 ("The format described here includes a header, section information, and one or more sections. The section information contains the information for all sections, which is more advantageous than having each section include its own information, i.e., the information is concentrated rather than distributed across the sections. Furthermore, the section information contains information about the encoding of the sections.")

Bauer (Ex-1109) at ¶31 ("There are many possible applications of this format and its individually coded sections. For example, an operating system memory manager can load and unload sections of memory according to images in this format. It can also be used as a file format in which executable files are stored, and linkers and program loaders can be readily adapted to support (read, write, and interpret) the format. Object code and data can also be stored in this file format, with a program loader reading the stored information and processing stored sections accordingly. One example of such a program loader is described in U.S. patent application Ser. No. 11/040,798 filed on Jan. 22, 2005, by M. Svensson et al. for "Operating-System-Friendly Bootloader".")

Bauer (Ex-1109) at ¶35 ("The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.")

Bauer (Ex-1109) at ¶36 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C. *The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208,*

210.")
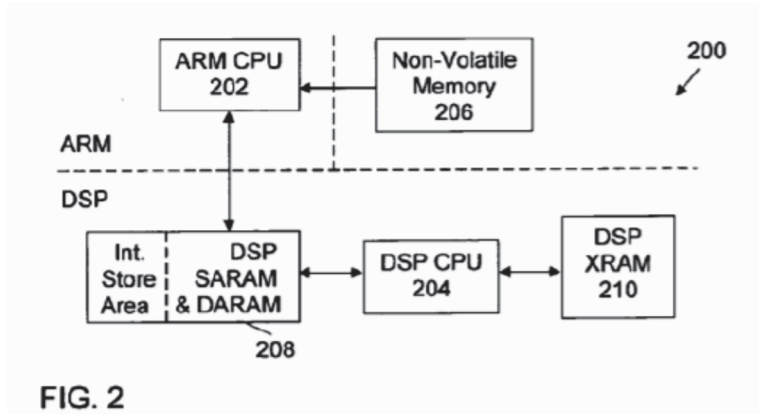
Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) at ¶37 ("As depicted in FIG. 1A, the section information entry or entries 104 precede the data 106 of the section(s) in the image 100. The section data 106 is advantageously arranged in the image in a sequence, and it is preferable that the section data 106 as well as the section information entries 104 are arranged in order of the section load addresses 110, starting with the lowest address. It will be understood, however, that other orders are suitable, e.g., starting with the highest address, and that in general it is not necessary to order the section by their load addresses. The sections may be in an arbitrary order. As each section has a respective load address, the sections can appear in any order (e.g., by size, coding type, or whatever is suitable). It is currently believed, however, that the most efficient solution from a loading point of view is probably arranging the sections by load address in either descending or ascending order.")
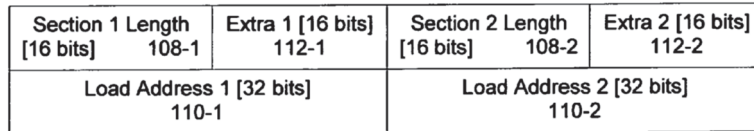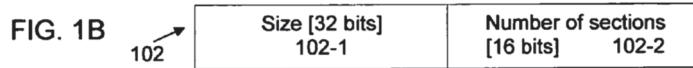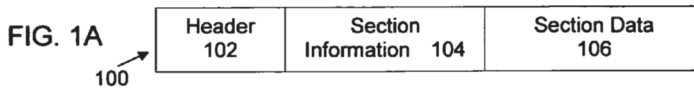
Bauer (Ex-1109), Figures 1A-C:

FIG. 1A

| Header 102 | Section Information 104 | Section Data 106 |

100

FIG. 1B

102

| Size [32 bits] 102-1 | Number of sections [16 bits] 102-2 |

| Section 1 Length [16 bits] 108-1 | Extra 1 [16 bits] 112-1 | Section 2 Length [16 bits] 108-2 | Extra 2 [16 bits] 112-2 |
| Load Address 1 [32 bits] 110-1 | | Load Address 2 [32 bits] 110-2 | |

FIG. 1C

104-1

104-2

Bauer (Ex-1109) at ¶43 ("Having information about the sections collected in the header 102 and section information 104 simplifies optimization in a number of circumstances, for instance, if sections are to be loaded into memory. The block 104 lists all sections, preferably in order of memory location, and this makes memory loading efficient as there is no need to search through an image for section headers when loading.")

Svensson

Svensson (Ex-1110) at 3:49-63 ("FIG. 1 depicts such a multi-processor system 100 that includes a host processor 102 and a client processor 104. It will be appreciated that although FIG. 1 shows one client processor 104, more can be provided. It will also be appreciated that the host and client processors may be any programmable electronic processors. In the example depicted in FIG. 1, the processor 102 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 104 is shown as the CPU of a digital signal processor (DSP) device. The dashed line in FIG. 1 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 106. The memory 106 may be a ROM, a flash memory, or other type of non-volatile memory device.")

Svensson (Ex-1110) at 3:64-4:3 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110. An intermediate storage area,

indicated by the dashed line, is defined within the memory 108 as described in more detail below.")

Svensson (Ex-1110) at 4:3-8 ("The arrows in FIG. 1 indicate access paths, e.g., busses and DMA paths, between the CPUs and the memories. The ARM host CPU 102 can access the non-volatile memory 106 and the SARAM and DARAM 108 of the DSP, but not the DSP's XRAM 110, and the DSP slave CPU 104 can access all of the RAMs 108, 110.")
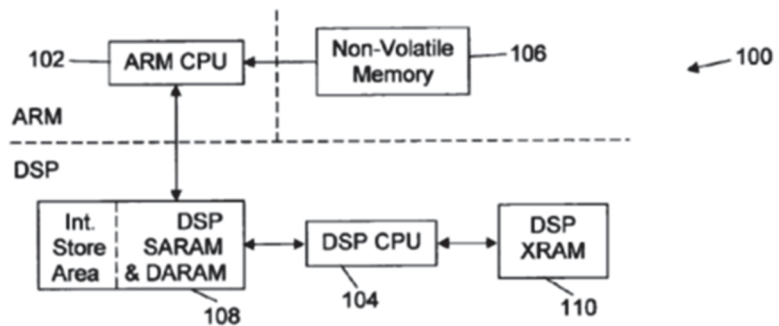
Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 5:21-28 ("The idle process reserves a block of memory in the slave's heap of memory that is located in the memory visible to the host, such as "internal" memory 108 (Step 212). As described in more detail below, this reserved block of memory is used for intermediate storage of information (code and/or data) to be transferred to the slave-private memory, i.e., the memory that is invisible to the host, such as "external" XRAM 110.")

Svensson (Ex-1110) at 5:65-67 ("The slave copies the contents of the intermediate storage area to appropriate locations in its slave-private memory (Step 220), thereby implementing its actual loading.")

Svensson (Ex-1110) at 6:12-23 ("As described above, the host fills the intermediate storage area in the memory 108 with code and data that the slave further copies to end destinations in the slave-private memory 110. Perhaps the simplest way of doing this is to precede all code and data in the intermediate storage area with a tag that contains the destination address and length of the block to be loaded. FIG. 3 depicts one example of such an organization of the intermediate storage area. A block of code and/or data to be transferred into the intermediate storage

area includes a header that indicates the length of the block and where it is to be loaded in the slave memory, i.e., the destination address.")

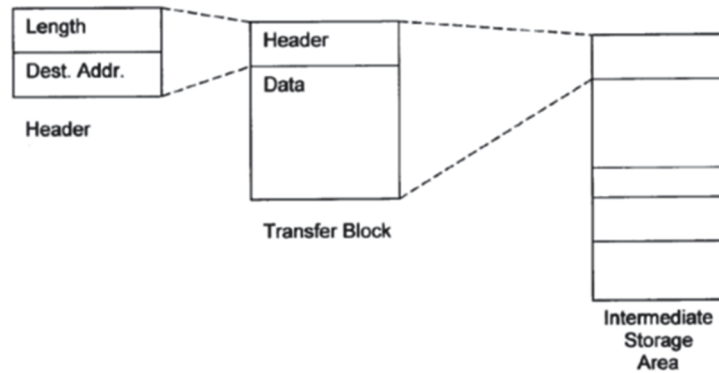Svensson (Ex-1110), Figure 3:

FIG. 3

'949 patent

'949 patent (Ex-1101) at 2:23-34 ("In a system in Which the software image is loaded onto a target "secondary" processor from a first "primary" processor, one way of performing such loading is to allocate a temporary buffer into which each packet is received, and each packet would have an associated packet header information along with the payload. The payload in this case would be the actual image data. From the temporary buffer, some of the processing may be done over the payload, and then the payload would get copied over to the final destination. The temporary buffer would be some place in system memory, such as in internal random-access-memory (RAM) or double data rate (DDR) memory, for example.

'949 patent (Ex-1101) at 2:35-41 ("Thus, where an intermediate buffer is used, the data being downloaded from a primary processor to a secondary processor is copied into the intermediate buffer. In this way, the buffer is used to receive part of the image data from the primary processor, and from the buffer the image data may be scattered into the memory (e.g., volatile memory) of the secondary processor.")

'949 patent (Ex-1101) at 2:58-61 ("A multi-processor system is offered. The system includes a secondary processor having a system memory and a hardware buffer for receiving at a least a portion of an executable software image.")

'949 patent (Ex-1101) at 7:20-26 ("As mentioned above, traditional loading processes require an intermediate step where the binary multi-segmented image is buffered (e.g., transferred into the system memory) and then later scattered into target locations (e.g., by a boot loader). Aspects of the present disclosure provide techniques that alleviate the intermediate step of buffering required in traditional loading processes."
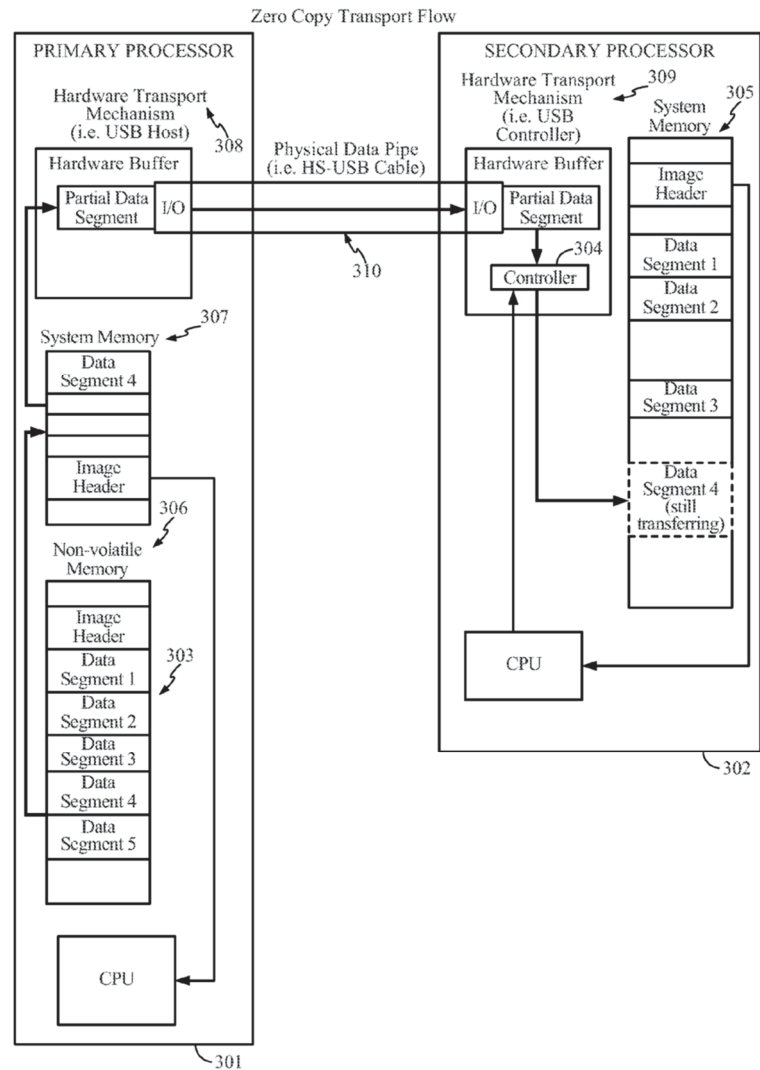
'949 patent (Ex-1101), Figure 3:

FIG. 3

Bauer + Svensson + Kim

| 13. The method of claim 10 in which the processing | *See* claim 10. |

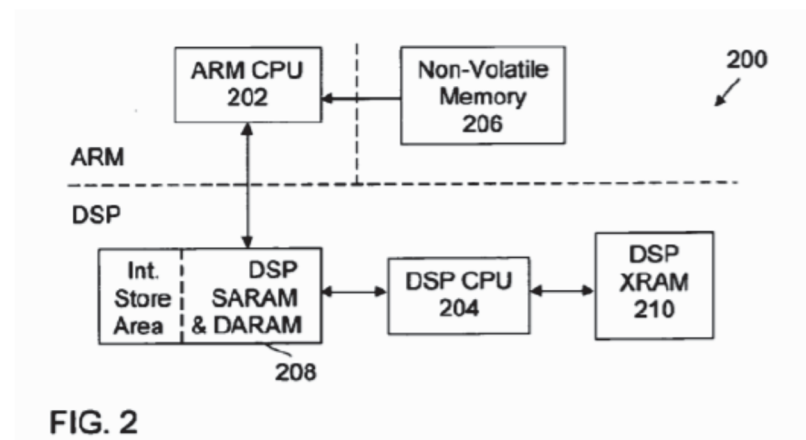| occurs prior to the loading.	*See* claim limitations [10b], [10c], and [10e].
	Bauer + Svensson + Kim
14. The method of claim 10 in which the primary and secondary processors are located on different chips.	*See* claim limitation [10a]. Bauer Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor 204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of a digital signal processor (DSP) device. ***The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 206.*** The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.") Bauer (Ex-1109), Figure 2: FIG. 2 Bauer (Ex-1109) ¶36 ("***Most commercially available DSP devices include on-chip memories***, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external"

RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C. The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Svensson

Svensson (Ex-1110) at 3:54-63 ("In the example depicted in FIG. 1, the processor 102 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 104 is shown as the CPU of a digital signal processor (DSP) device. The dashed line in FIG. 1 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory 106. The memory 106 may be a ROM, a flash memory, or other type of non-volatile memory device.")
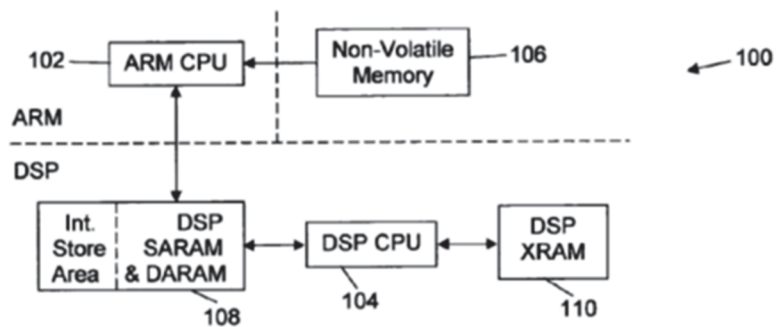
Svensson (Ex-1110), Figure 1:

FIG. 1

Svensson (Ex-1110) at 3:64-4:1 ("Most commercially available DSP devices include on-chip memories, and as indicated in FIG. 1, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 108, as well as an "external" RAM (XRAM) 110.")

Zhao

Zhao (Ex-1113) at ¶33 ("Although embodiments of the dual processor architecture may be described as comprising the host processor 102 and the radio processor 104 for purposes of

illustration, it is worthy to note that the dual processor architecture of the mobile computing device 100 may comprise additional processors, may be implemented as a dual- or multi-core chip with both host processor 102 and radio processor 104 on a single chip, etc.")
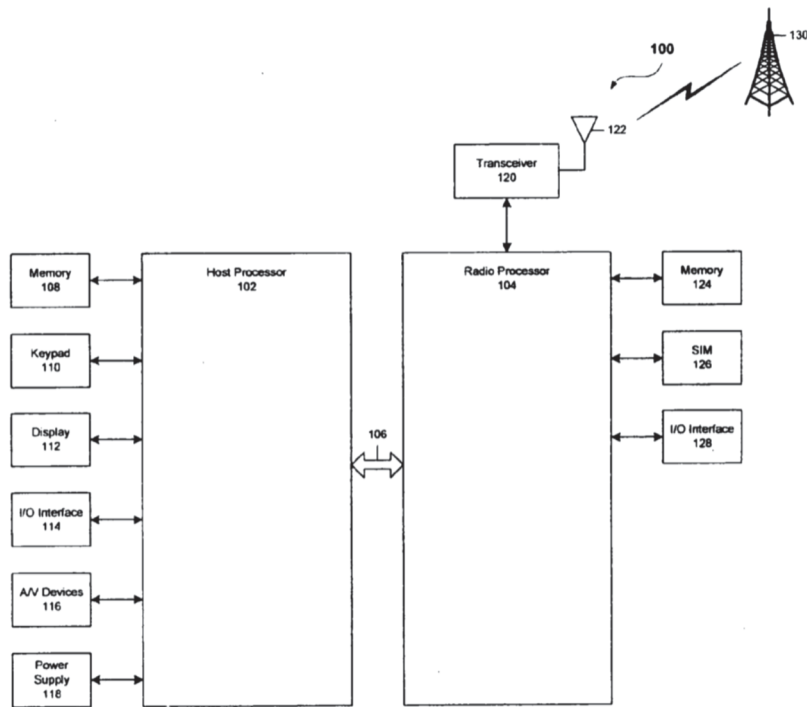
Zhao (Ex-1113), Figure 3:

FIG. 3

Zhao (Ex-1113) at ¶150 ("Host processor 102 and radio processor 104 each contain a universal serial bus controller, which may be on-chip or a separate integrated circuit associated with the respective processor. The host USB controller is coupled to host processor 102 and is configured to provide USB communication over a universal serial bus (USB) with radio processor 104. The radio USB controller is coupled to radio processor 104 and is configured to [provide] USB communication over the USB with host processor 102.")
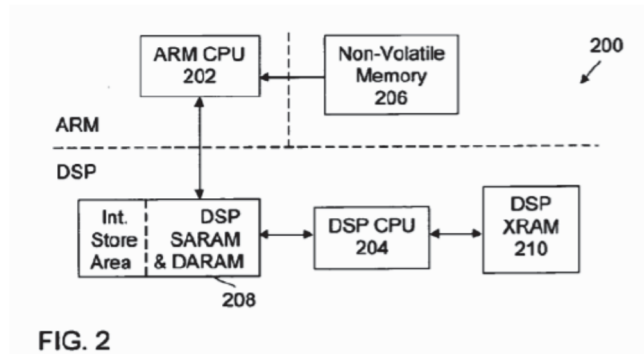
'949 patent

'949 patent (Ex-1101) at 1:45-48 ("A problem exists on a significant number of devices (such as smart phones) that

	incorporate multiple processors (e.g., a standalone application processor chip integrated with a separate modem processor chip).")
	'949 patent (Ex-1101) at 2:42-45 ("The primary processor and its non-volatile memory that stores the boot image for a secondary processor may be implemented on a different chip than a chip on which the secondary processor is implemented.")
Bauer + Svensson + Kim	
15. The method of claim 10 further comprising performing the receiving, processing, and loading, in at least one of a mobile phone, a set top box, a music player, a video player, an entertainment unit, a navigation device, a computer, a hand-held personal communication systems (PCS) unit, a portable data unit, and a fixed location data unit.	*See* claim 10. Bauer Bauer (Ex-1109) at ¶15 ("A lot of software today is sent across wireless communication links (e.g., wireless local area networks (WLANs), mobile telephony networks, etc.), and it is important that software can be sent in a secure manner.") Bauer (Ex-1109) at ¶16 ("Greater efficiency in loading data can reduce response times in such systems, and space-efficient storage saves valuable memory.") Bauer (Ex-1109) at ¶26 ("As described above, the binary data format described in this application is useful in processor systems, such as embedded systems, in which efficiency is important. Greater efficiency when loading software can lower response times in embedded systems and other computer systems, and space-efficient storage saves valuable memory.") Svensson Svensson (Ex-1110) at 7:61-67 ("It is expected that this invention can be implemented in a wide variety of environments, including for example mobile communication devices. Newer ones of such devices can employ the OS-friendly bootloader described here to boot their DSPs, which may be provided to handle multimedia tasks, in cooperation with their main-processor software systems.") Svensson (Ex-1110) at 8:26-32 ("This capability can be important in many devices and many use cases. In a mobile telephone, for example, such use cases include making a call, receiving a call, compressing/decompressing speech, playing music files, etc. With the OS-friendly bootloader described

	here, one can load and execute new software in the slave processor virtually anytime the host processor is running.")
	’949 patent
	’949 patent (Ex-1101) at 1:39-44 ("In a multi-processor system, each processor may require respective boot code for booting up. As an example, in a smartphone device that includes an application processor and a modem processor, each of the processors may have respective boot code for booting up.")

	Bauer + Svensson + Kim + Zhao
[16] An apparatus comprising:	*See* below
[16a] means for receiving at a secondary processor, from a primary processor via an inter-chip communication bus, an image header for an executable software image for the secondary processor that is stored in memory coupled to the primary processor, the executable software image comprising the image header and at least one data segment,	*See* claim limitation [10a]. Bauer Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor 204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of a digital signal processor (DSP) device. ***The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory*** 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.") Bauer (Ex-1109) ¶36 ("***Most commercially available DSP devices include on-chip memories***, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. ***The***

arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C. The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) at ¶38 ("Having all section information entries 104 collected together in the image 100 advantageously simplifies system navigation through the image, and having all section data arranged in a sequence makes it possible to optimize loading of the sections. For instance, it is simple to split or concatenate sections when they are adjacent in memory. The ability to split sections can be useful, for instance, when a DMA transfer is to be set up. As there is always a small overhead when setting up a DMA transfer, a DMA unit can be used in an efficient way by arranging the size of the data to be transferred to be equal or close to the block sizes used by the DMA unit. As sections can be located sequentially in an image 100, it is simple to split a section into several suitable pieces before downloading it.")

Zhao

Zhao (Ex-1113) at ¶32 ("As shown in the embodiment of FIG. 3, mobile computing device 100 may comprise a dual processor architecture including a host processor 102 and a radio processor 104 (e.g., a base band processor). The host processor 102 and the radio processor 104 may be arranged to communicate with each other using interfaces 106 such as one or more universal serial bus (USB) interfaces, micro-USB interfaces, universal asynchronous receiver-transmitter

(UART) interfaces, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth.")

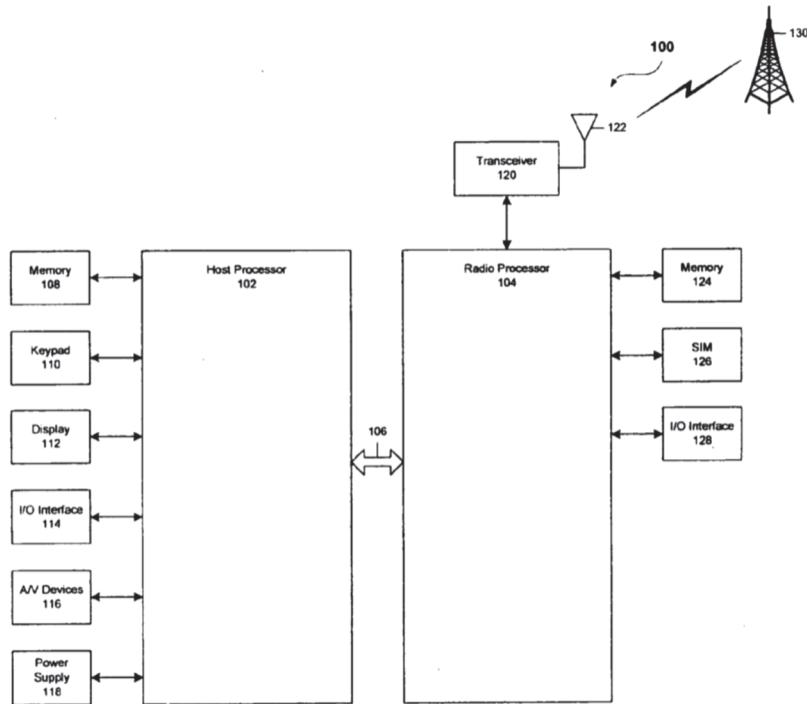Zhao (Ex-1113), Figure 3:

FIG. 3

Zhao (Ex-1113) at ¶33 ("Although embodiments of the dual processor architecture may be described as comprising the host processor 102 and the radio processor 104 for purposes of illustration, it is worthy to note that the dual processor architecture of the mobile computing device 100 may comprise additional processors, may be implemented as a dual- or multi-core chip with both host processor 102 and radio processor 104 on a single chip, etc.")

Zhao (Ex-1113) at ¶34 ("In various embodiments, the host processor 102 may be implemented as a host central processing unit (CPU) using any suitable processor or logic device, such as a general purpose processor. The host processor 102 may comprise, or be implemented as, a chip multiprocessor (CMP), dedicated processor, embedded processor, media processor, input/output (I/O) processor, co-processor, a field

<table>
<tr>
<td></td>
<td>

programmable gate array (FPGA), a programmable logic device (PLD), or other processing device in alternative embodiments. In an exemplary embodiment, host processor 102 is an OMAP2, such as an OMAP2431 processor, manufactured by Texas Instruments, Inc.")

Zhao (Ex-1113) at ¶44 ("As mentioned above, the radio processor 104 may perform voice and/or data communication operations for the mobile computing device 100. For example, the radio processor 104 may be arranged to communicate voice information and/or data information over one or more assigned frequency bands of a wireless communication channel. In various embodiments, the radio processor 104 may be implemented as a communications processor using any suitable processor or logic device, such as a modem processor or baseband processor. Although some embodiments may be described with the radio processor 104 implemented as a modem processor or baseband processor by way of example, it may be appreciated that the embodiments are not limited in this context. For example, the radio processor 104 may comprise, or be implemented as, a digital signal processor (DSP), media access control (MAC) processor, or any other type of communications processor in accordance with the described embodiments. Radio processor 104 may be any of a plurality of modems manufactured by Qualcomm, Inc.")

Zhao (Ex-1113) at ¶52 ("Each processor is provided with the ability to wake the other when communication between the two is needed (e.g., via UART, USB or shared memory).")

Zhao (Ex-1113) at ¶60 ("Referring now to FIG. 5, another exemplary system and method for limiting power consumption will be described. FIG. 5 is a schematic diagram illustrating communication lines between a plurality of processors, according to an exemplary embodiment. As can be seen, a plurality of digital serial ports and control signals 500 of radio processor 104 are coupled to host processor 102. Analog audio signals 502 are coupled to audio CODEC 504 for routing to microphones, speakers, or host processor 102. In one exemplary embodiment, UART1 506 may be used for multiplexed control and data. UART2 508 may be used for debug information during development. In an alternative embodiment, UART1 506 may be used for command and diagnostics information and UART2 508 may be used for data calls.")

</td>
</tr>
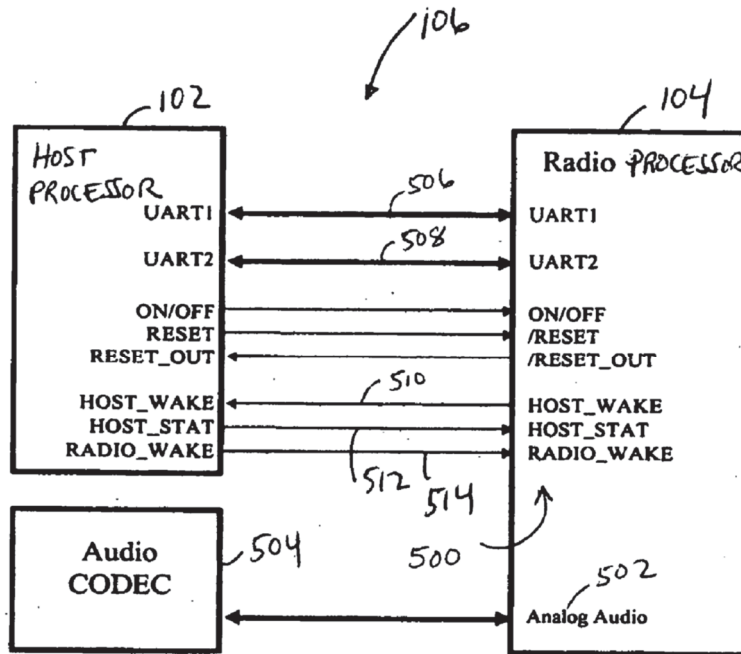</table>

Zhao (Ex-1113), Figure 5:

FIG. 5

Zhao (Ex-1113) at ¶150 ("Host processor 102 and radio processor 104 each contain a universal serial bus controller, which may be on-chip or a separate integrated circuit associated with the respective processor. The host USB controller is coupled to host processor 102 and is configured to provide USB communication over a universal serial bus (USB) with radio processor 104. The radio USB controller is coupled to radio processor 104 and is configured to [provide] USB communication over the USB with host processor 102.")

'949 patent

'949 patent (Ex-1101) at 2:42-45 ("The primary processor and its non-volatile memory that stores the boot image for a secondary processor may be implemented on a different chip than a chip on which the secondary processor is implemented.")

[16b] the image header and each data segment being received separately;	*See* claim limitation [10b]. *See* claim limitation [16a].
[16c] means for processing,	*See* claim limitation [10a] and [10c].

by the secondary processor, the image header to determine at least one location within system memory to which the secondary processor is coupled to store each data segment;	*See* claim 15. Bauer Bauer (Ex-1109) at ¶15 ("An embedded computer environment has many features that are not present in a desktop- or server-computer environment. For example, it is usually important that the sizes of binary images are kept low, as an embedded system usually has limited storage capacity. Thus, binary files should contain as little overhead as possible. It is also important that object code and data can be loaded efficiently, as processing power may be limited, especially in an embedded system. A lot of software today is sent across wireless communication links (e.g., wireless local area networks (WLANs), mobile telephony networks, etc.), and it is important that software can be sent in a secure manner. If a binary file format supports encryption, a higher level of safety can be achieved.") Svensson Svensson (Ex-1110) at 7:61-63 ("It is expected that this invention can be implemented in a wide variety of environments, including for example mobile communication devices.") Svensson (Ex-1110) at 8:26-29 ("This capability can be important in many devices and many use cases. In a mobile telephone, for example, such use cases include making a call, receiving a call, compressing/decompressing speech, playing music files, etc.") Zhao Zhao (Ex-1113) at ¶26 ("Referring first to FIG. 1, a mobile computing device 100 is shown. Device 100 is a smart phone, which is a combination mobile telephone and handheld computer having personal digital assistant functionality. The teachings herein can be applied to other mobile computing devices (e.g., a laptop computer) or other electronic devices (e.g., a desktop personal computer, home or car audio system, etc.). Personal digital assistant functionality can comprise one or more of personal information management, database functions, word processing, spreadsheets, voice memo recording, etc. and is configured to synchronize personal

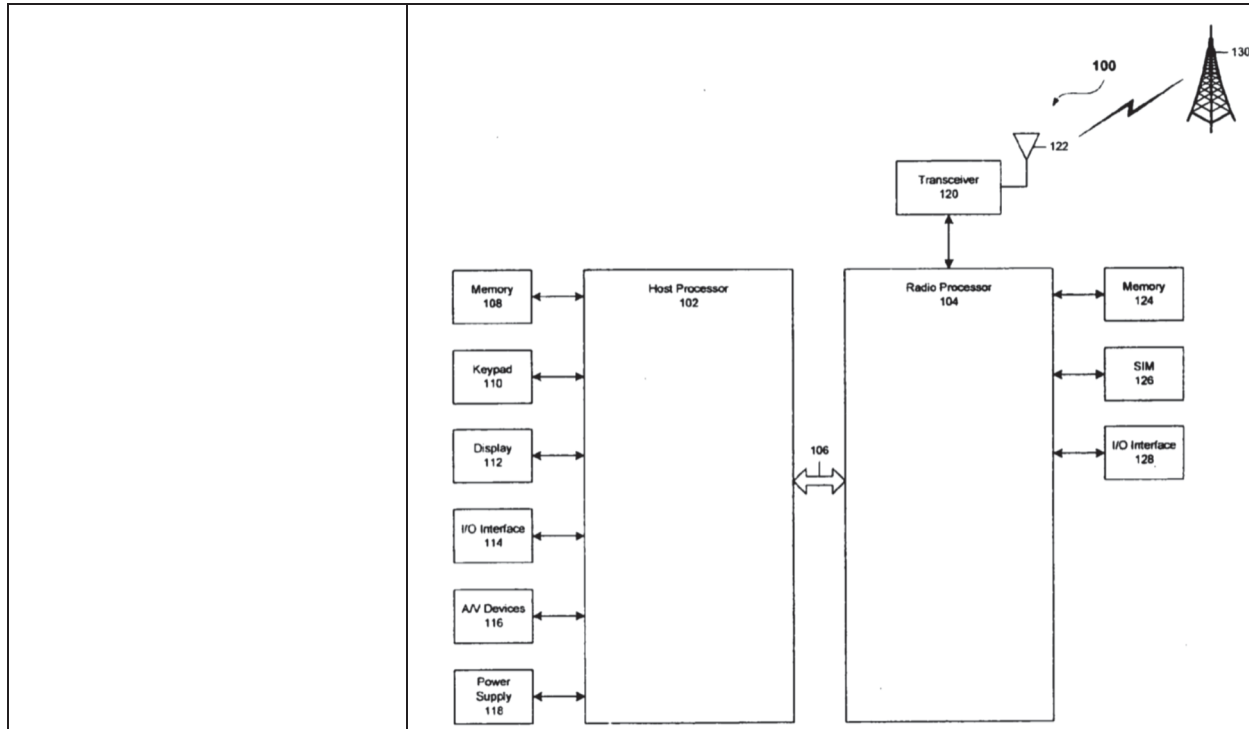|	information from one or more applications with a computer (e.g., desktop, laptop, server, etc.). Device 100 is further configured to receive and operate additional applications provided to device 100 after manufacture, e.g., via wired or wireless download, SecureDigital card, etc.") Zhao (Ex-1113) at ¶32 ("As shown in the embodiment of FIG. 3, mobile computing device 100 may comprise a dual processor architecture including a host processor 102 and a radio processor 104 (e.g., a base band processor). The host processor 102 and the radio processor 104 may be arranged to communicate with each other using interfaces 106 such as one or more universal serial bus (USB) interfaces, micro-USB interfaces, universal asynchronous receiver-transmitter (UART) interfaces, general purpose input/output (GPIO) interfaces, control/status lines, control/data lines, shared memory, and so forth.") Zhao (Ex-1113) at ¶44 ("As mentioned above, the radio processor 104 may perform voice and/or data communication operations for the mobile computing device 100. For example, the radio processor 104 may be arranged to communicate voice information and/or data information over one or more assigned frequency bands of a wireless communication channel. In various embodiments, the radio processor 104 may be implemented as a communications processor using any suitable processor or logic device, such as a modem processor or baseband processor. Although some embodiments may be described with the radio processor 104 implemented as a modem processor or baseband processor by way of example, it may be appreciated that the embodiments are not limited in this context. For example, the radio processor 104 may comprise, or be implemented as, a digital signal processor (DSP), media access control (MAC) processor, or any other type of communications processor in accordance with the described embodiments. Radio processor 104 may be any of a plurality of modems manufactured by Qualcomm, Inc.") Zhao (Ex-1113), Figure 3:

FIG. 3

<table>
<tr>
<td></td>
<td>

'949 patent

'949 patent (Ex-1101) at 1:45-48 ("A problem exists on a significant number of devices (such as smart phones) that incorporate multiple processors (e.g., a standalone application processor chip integrated with a separate modem processor chip).")

</td>
</tr>
<tr>
<td>

[16d] means for receiving at the secondary processor, from the primary processor via the inter-chip communication bus, each data segment; and

</td>
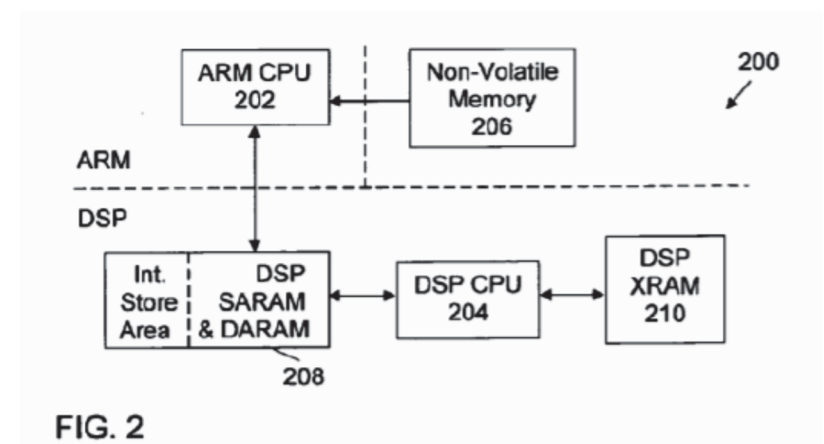<td>

See claim limitation [10d].

See claim limitation [16a] regarding the feature "means for receiving … via the inter-chip communication bus".

Bauer

Bauer (Ex-1109) at ¶35 ("FIG. 2 depicts a multi-processor system 200 that includes a host processor 202 and a client processor 204 and that can advantageously use a binary image 100 having the format depicted in FIGS. 1A, 1B, 1C. It will be appreciated that although FIG. 2 shows one client processor 204, more can be provided, and it will further be appreciated that although FIG. 2 shows a multi-processor system, even

</td>
</tr>
</table>

only a single processor 202 can be provided. Moreover, the processor(s) may be any programmable electronic processor(s). In the example depicted in FIG. 2, the processor 202 is shown as the central processing unit (CPU) of an advanced RISC machine (ARM), and the processor 204 is shown as the CPU of a digital signal processor (DSP) device. ***The dashed line in FIG. 2 depicts the hardware boundary between the host and slave devices, in this example, the ARM and the DSP, and also a non-volatile memory*** 206. The memory 206 may be a ROM, a flash memory, or other type of non-volatile memory device, within which an image in the format depicted in FIGS. 1A-1C can be stored.")

Bauer (Ex-1109), Figure 2:

FIG. 2

Bauer (Ex-1109) ¶36 ("***Most commercially available DSP devices include on-chip memories***, and as indicated in FIG. 2, the DSP includes "internal" single-access RAM (SARAM) and dual-access RAM (DARAM) 208, as well as an "external" RAM (XRAM) 210. An intermediate storage area, indicated by the dashed line, may be defined within the memory 208. ***The arrows in FIG. 2 indicate access paths, e.g., busses and direct memory access (DMA) paths, between the CPUs and the memories, any one or more of which may store an image in the format depicted in FIGS. 1A-1C.*** The ARM host CPU 202 can access the non-volatile memory 206 and the SARAM and DARAM 208 of the DSP, but not the DSP's XRAM 210, and the DSP slave CPU 204 can access all of the RAMs 208, 210.")

[16e] means for scatter loading, by the secondary processor, each data segment directly to the	*See* claim limitations [10a], [10b], and [10e]. *See* claim limitations [16a] and [16c].

determined at least one location within the system memory, and each data segment being scatter loaded based at least in part on the processed image header.	
Bauer + Svensson + Kim + Zhao	
17. The apparatus of claim 16 integrated into at least one of a mobile phone, a set top box, a music player, a video player, an entertainment unit, a navigation device, a computer, a hand-held personal communication system (PCS) unit, a portable data unit, and a fixed location data unit.	*See* claim 16. *See* claim 15.

XI.   AVAILABILITY FOR CROSS-EXAMINATION

200.   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.
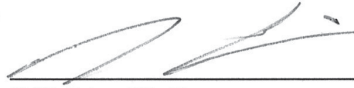
XII. RIGHT TO SUPPLEMENT

201.   I reserve the right to supplement my opinions in the future to respond

to any arguments that the Patent Owner raises and to take into account new

information as it becomes available to me.

XIII. JURAT

202.   I declare that all statements made herein of my own knowledge are

true and that all statements made on information and belief are believed to be true;

and further that these statements were made with the full knowledge that willful

false statements and the like so made are punishable by fine or imprisonment, or

both, under Section 1101 of Title 18 of the United States code.

Dated: 7/2/08

Bill Lin, Ph.D.

APPENDIX A

BILL LIN
Curriculum Vitae
Tel: 858.531.2441
Email: billmbox@gmail.com

EDUCATION

- **Ph.D**, Electrical Engineering and Computer Sciences
  University of California, Berkeley, May 1991
- **Masters of Science**, Electrical Engineering and Computer Sciences
  University of California, Berkeley, May 1988
- **Bachelor of Science**, Electrical Engineering and Computer Sciences
  University of California, Berkeley, May 1985

PROFESSIONAL EXPERIENCE

1/1997 – present	**University of California, San Diego** Professor, Electrical and Computer Engineering Adjunct Professor, Computer Science and Engineering
2/1992 – 12/1996	**IMEC, Leuven, Belgium** Group Head, Systems Control and Communications Group

AREAS OF EXPERTISE

- All aspects of computer architecture and computer network problems, including the design of heterogeneous multicore processors, systems-on-chips, data networks, wireless communications and mobile computing systems, and multimedia and graphics systems.

PROFESSIONAL ACTIVITIES

- Served or serving as Editor on 3 ACM or IEEE journals, as General Chair on 4 ACM or IEEE conferences, on the Organizing or Steering Committees for 5 ACM or IEEE conferences, and on the Technical Program Committees of 44 ACM or IEEE conferences.

SELECTED PROFESSIONAL ACTIVITIES

- Associate Editor, ACM Transactions on Design Automation and Electronics Systems (TODAES), 2010-2015
- Editorial Board, International Journal of Embedded Systems, 2003-present
- General Chair, ACM/IEEE Symposium on Networks-on-Chips (NOCS), 2009
- General Chair, ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2010
- Chair of Steering Committee, ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2012-2017
- Guest Editor, IEEE Transactions on Network and Service Management (TNSM), 2012-2013
- General Co-Chair, ACM SIGMETRICS, 2015

PUBLICATIONS

- Published over 170 journal articles and conference papers in top-tier venues and publications.

PATENTS

- Mitigating Low-Rate Denial-of-Service Attacks in Packet-Switched Networks. United States Patent 8,443,444. Issued in May 14, 2013.
- Systems and Methods for Proactive Surge Protection. United States Patent 7,860,004. Issued in December 28, 2010.
- Method and Apparatus for Scan Block Caching. United States Patent 7,672,005. Issued in March 2, 2010.
- Design Environment and a Design Method for Hardware/Software Co-Design. United States Patent 5,870,588. Issued in February 9, 1999.
- System and Method for Generating a Hazard-Free Asynchronous Circuit. United States Patent 5,748,487. Issued in May 5, 1998.

EXPERT WITNESS HISTORY

I have served as an expert witness on 21 cases (most cases involved multiple patents in suit), of which I wrote expert reports or declarations for 14 cases, analyzed source code in 10 cases, was deposed 11 times, and testified at trial 2 times. The specific cases and services provided are as follows:

1. Vizio, Inc. v. LG Electronics, Inc. and LG Electronics USA Inc., Civil Action No. 09-cv-1481-BEL
   - Expert witness on behalf of Vizio, Inc.
   - Law firm: Jones Day.
   - Services provided: Provided expert report, analyzed source code.
   - Technologies: Digital TV.

2. Vizio Inc. v. LG Electronics, Inc. and LG Electronics USA Inc., In the Matter of CERTAIN FLAT PANEL DIGITAL TELEVISIONS AND COMPONENTS THEREOF, U.S. International Trade Commission Inv. No. 337-TA-733
   - Expert witness on behalf of Vizio, Inc.
   - Law firm: Jones Day.
   - Services provided: Provided expert report, analyzed source code.
   - Technologies: Digital TV.

3. Vizio Inc. v. Coby Electronics Corp., Curtis International Ltd., E&S International Enterprises, Inc., MStar Semiconductor, Inc., ON Corp US, Inc., Renesas Electronics Corporation, Japan, Renesas Electronics America, Inc., Sceptre, Inc., and Westinghouse Digital, LLC, In the Matter of CERTAIN DIGITAL TELEVISIONS AND COMPONENTS THEREOF, U.S. International Trade Commission Inv. No. 337-TA-789
   - Expert witness on behalf of Vizio, Inc.
   - Law firm: Jones Day.
   - Services provided: Provided expert reports, was deposed, analyzed source code.
   - Technologies: Digital TV.

4. Linex Technologies Inc. v. Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless, In the Matter of CERTAIN WIRELESS COMMUNICATION DEVICES AND SYSTEMS, COMPONENTS THEREOF, AND PRODUCTS CONTAINING SAME, U.S. International Trade Commission Inv. No. 337-TA-775
   - Expert witness on behalf of Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless
   - Law firms: Covington & Burling, K&L Gates, Morrison & Foerster, Lewis Roca & Rothgerber.
   - Services provided: Provided expert reports, was deposed, analyzed source code.
   - Technologies: WiFi networks.

5. MOSAID Technologies, Inc. v. Dell, Inc., Qualcomm Atheros, Inc., et al., Case No. 2:11-cv-00179-TJW
   - Expert witness on behalf of Qualcomm Atheros, Inc.
   - Law Firm: McDermott Will & Emory.
   - Services provided: Provided expert analysis.
   - Technologies: WiFi networks.

6. Hitachi Consumer Electronics Co. Ltd., et al. v. Top Victory Electronics (Taiwan) Co. Ltd., et al., Civil Action No. 2:10-cv-260-JRG.
   - Expert witness on behalf of Vizio, Inc.
   - Law Firm: Akin Gump Strauss Hauer & Feld
   - Services provided: Provided expert reports, analyzed source code.
   - Technologies: Digital TV.

172

7. United States International Trade Commission (USITC) Investigation Number: 337-TA-853 and Technology Properties Limited, LLC v. Kyocera Corporation and Kyocera Communciations, Inc. in the Northern District of California, Case No. 3:12-cv-03860-JSC
   - Expert witness on behalf of Kyocera Corporation and Kyocera Communciations.
   - Law firm: Morrison & Foerster.
   - Services provided: Provided expert analysis.
   - Technologies: Cell phones, microprocessors.

8. Netgear, Inc. v. Ruckus Wireless, Inc., Case No. 1:10-cv-00999-SLR / D. Del.
   - Expert witness on behalf of Ruckus Wireless, Inc.
   - Law firms: Lewis Roca & Rothgerber, Orrick.
   - Services provided: Provided expert report, analyzed source code, was deposed, and testified at trial.
   - Technologies: WiFi networks.

9. Sonics, Inc. v. Arteris, Inc., Case No. 11 CV-05311 SBA, and Arteris S.A.S. v. Sonics, Inc., Case No. C 12-00434 (WHA)
   - Expert witness on behalf of Arteris, Inc. and Arteris S.A.S.
   - Law firm: DLA Piper.
   - Services provided: Retained as an expert.
   - Technologies: Systems-on-Chips (SoCs).

10. Linex Technologies Inc. v. Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless, U.S. District Court, Northern District of California, C 13-00159 CW
    - Expert witness on behalf of Hewlett-Packard Company, Apple Inc., Aruba Networks, Inc., Meru Networks, and Ruckus Wireless
    - Law firms: Covington & Burling, Milbank, K&L Gates, Morrison & Foerster, Lewis Roca & Rothgerber.
    - Services provided: Provided expert reports, analyzed source code.
    - Technologies: WiFi networks.

11. Frontier Communications Corp. v. Google, C.A. No. 10-545 (D. Del)
    - Expert witness on behalf of Frontier Communications Corp.
    - Law firm: Steese, Evans & Frankel
    - Services provided: Provided expert analysis, wrote declarations, analyzed source code.
    - Technologies: Voice over IP.

12. U.S. Ethernet Innovations LLC v. Acer, Inc. et al., Case No. 10-cv-3724 (N.D. Cal.)
    - Expert witness on behalf of Qualcomm Atheros, Inc., Sigma Designs, Inc., and AT&T Services, Inc.
    - Law firms: Reed Smith LLP, Pillsbury Winthrop Shaw Pittman LLP, and Vinson & Elkins LLP.
    - Services provided: Provided expert reports, was deposed, analyzed source code.
    - Technologies: Intelligent network interfaces, data center networks.

13. Inter Partes Review of a U.S. Patent.
    - Expert witness on behalf of Netgear, Inc. and Belkin International, Inc.
    - Law firm: Reed Smith LLP.
    - Services provided: Provided declaration for IPR petition.
    - Technologies: Wireless networks.

14. Inter Partes Review of U.S. Patents.
    - Expert witness on behalf of Arista Networks.
    - Law firm: Fish & Richardson.
    - Services provided: Provided declarations for IPR petitions, deposed 3 times.
    - Technologies: Network routers, data center networks.

15. Invalidity Expert Witness.
    - Expert witness on behalf of Intel Corp.
    - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
    - Services provided: Provided expert analysis.
    - Technologies: WiFi networks, bluetooth networks.

16. Parthenon Unified Memory Architecture LLC v. HTC Corporation et al., Case No. 2:14-CV-690-JRG-RSP (E. D. Texas).
    - Expert witness on behalf of HTC Corporation and HTC America, Inc.
    - Law firm: Sidley Austin LLP.
    - Services provided: Retained as an expert.
    - Technologies: Memory architectures.

17. System Architecture Information Technology dba SAI Technology, Inc. v. Qualcomm Inc.
    - Expert witness on behalf of Qualcomm Inc.
    - Law firm: McDermott Will & Emory.
    - Services provided: Retained as an expert, analyzed source code, was deposed, and testified at trial.
    - Technologies: WiFi networks.

18. Invalidity Expert Witness.
    - Expert witness on behalf of Sandvine Corp.
    - Law firm: Erise IP.
    - Services provided: Provided declarations.
    - Technologies: Network monitoring and measurements.

19. Inter Partes Review of U.S. Patents.
    - Expert witness on behalf of Intel Corp., Cavium Inc., and Dell Inc.
    - Law firms: Weil, Gotshal & Manges LLP, Duane Morris LLP, Alston & Bird LLP.
    - Services provided: Provided declarations for IPR petitions, deposed 2 times.
    - Technologies: Intelligent network interfaces, data center networks.

20. Expert Witness.
    - Law firm: Covington & Burling.
    - Services provided: Retained as an expert.
    - Technologies: WiFi networks.

21. Qualcomm Inc. v. Apple Inc., Inv. No. 337-TA-1065 (ITC), and Civ. Action No. 3:17-cv-01375-JAH-MDD (S.D. Cal.)
    - Expert witness on behalf of Apple Inc. and Intel Corp.
    - Law firm: Wilmer Cutler Pickering Hale and Dorr LLP.
    - Services provided: Provided expert reports/declarations, was deposed.
    - Technologies: Wireless systems, computer architecture.