US007356680B2

(12) **United States Patent**
Svensson et al.

(10) **Patent No.: US 7,356,680 B2**
(45) **Date of Patent: Apr. 8, 2008**

(54) **METHOD OF LOADING INFORMATION INTO A SLAVE PROCESSOR IN A MULTI-PROCESSOR SYSTEM USING AN OPERATING-SYSTEM-FRIENDLY BOOT LOADER**

(75) Inventors: **Mats Svensson**, Lund (SE); **Peter Aulin**, Malmö (SE); **Niclas Bauer**, Malmö (SE); **Michael Rosenberg**, Södra Sandby (SE)

(73) Assignee: **Telefonaktiebolaget L M Ericsson (publ)**, Stockholm (SE)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 482 days.

(21) Appl. No.: **11/040,798**

(22) Filed: **Jan. 22, 2005**

(65) **Prior Publication Data**

US 2006/0168435 A1 Jul. 27, 2006

(51) **Int. Cl.**
G06F 9/00 (2006.01)

(52) **U.S. Cl.** **713/1**; 713/2; 713/100; 709/208; 709/212; 709/213

(58) **Field of Classification Search** 713/1, 713/2, 100; 709/208, 212, 213
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,943,911	A		7/1990	Kopp et al.
5,068,780	A		11/1991	Bruckert et al.
5,155,833	A	*	10/1992	Cullison et al. 713/2
5,347,514	A	*	9/1994	Davis et al. 370/429
5,652,886	A		7/1997	Tulpule et al.
5,754,863	A	*	5/1998	Reuter 717/173
5,799,186	A		8/1998	Compton
5,835,784	A		11/1998	Gillespie et al.
5,944,820	A		8/1999	Beelitz
6,012,142	A	*	1/2000	Dokic et al. 713/2
6,058,474	A		5/2000	Baltz et al.
6,438,683	B1		8/2002	Endsley
6,490,722	B1		12/2002	Barton et al.
6,601,167	B1		7/2003	Gibson et al.
6,684,397	B1		1/2004	Byer et al.
6,691,216	B2		2/2004	Kelly et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1460539	A2	9/2004

(Continued)

OTHER PUBLICATIONS

PCT International Search Report, mailed Aug. 8, 2006, in connection with International Application No. PCT/EP2006/000351.

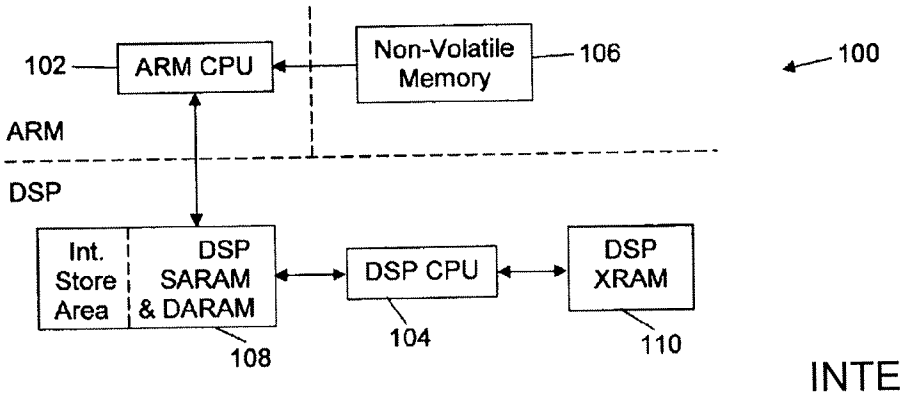
(Continued)

Primary Examiner—Thomas Lee
Assistant Examiner—Malcolm D Cribbs
(74) *Attorney, Agent, or Firm*—Potomac Patent Group PLLC

(57) **ABSTRACT**

A conventional bootloader can conflict with the operating system (OS) of a multi-processor system. An OS-friendly bootloader and methods are described that integrate an OS with a bootloader in any system in which a host processor and a client processor have a communication mechanism that requires the OS for the mechanism to work and the client has two memory systems: one visible to both host and client and one visible only to the client.

23 Claims, 2 Drawing Sheets



INTEL 1210

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

U.S. PATENT DOCUMENTS

6,760,785	B1	7/2004	Powderly et al.	
6,810,478	B1	10/2004	Anand et al.	
2002/0059560	A1 *	5/2002	Phillips	717/124
2002/0138156	A1	9/2002	Wong et al.	
2003/0126424	A1	7/2003	Horanzy et al.	
2004/0059906	A1	3/2004	Park et al.	
2004/0088697	A1	5/2004	Schwartz et al.	
2004/0215952	A1	10/2004	Oguma	

FOREIGN PATENT DOCUMENTS

WO	01/27753	A2	4/2001

OTHER PUBLICATIONS

PCT Written Opinion, mailed Aug. 8, 2006, in connection with International Application No. PCT/EP2006/00351.

Hyde, J., "How to Make Pentium Pros Cooperate", BYTE, McGraw-Hill, Inc. St. Peterborough, US, vol. 21, No. 4, Apr. 1996, pp. 1770178, XP000586039.

Winderweedle, B. et al., "TMS320VC5470/5471 Bootloader Application Report", Texas Instruments, Dallas, TX, SPRA376, Jun. 2002.

"OMAP5910 Dual-Core Processor DSP Subsystems Reference Guide", Texas Instruments, Dallas, TX, SPRU672, Oct. 2003.

* cited by examiner

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

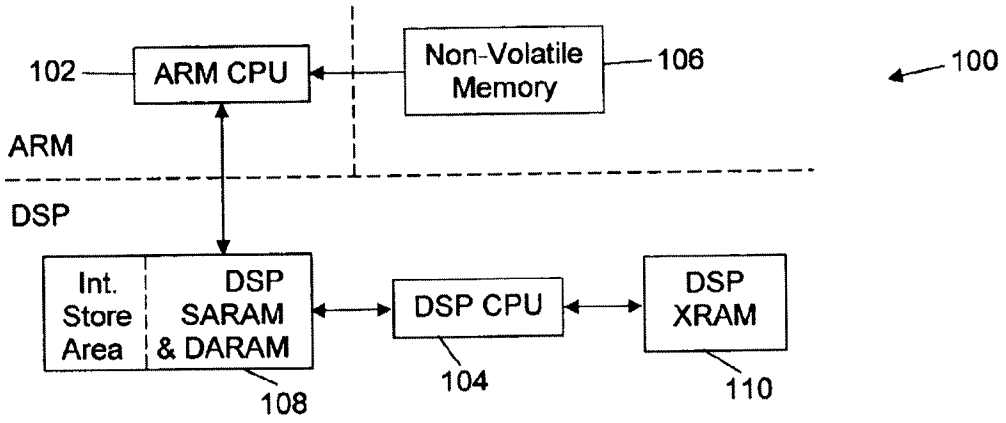


FIG. 1

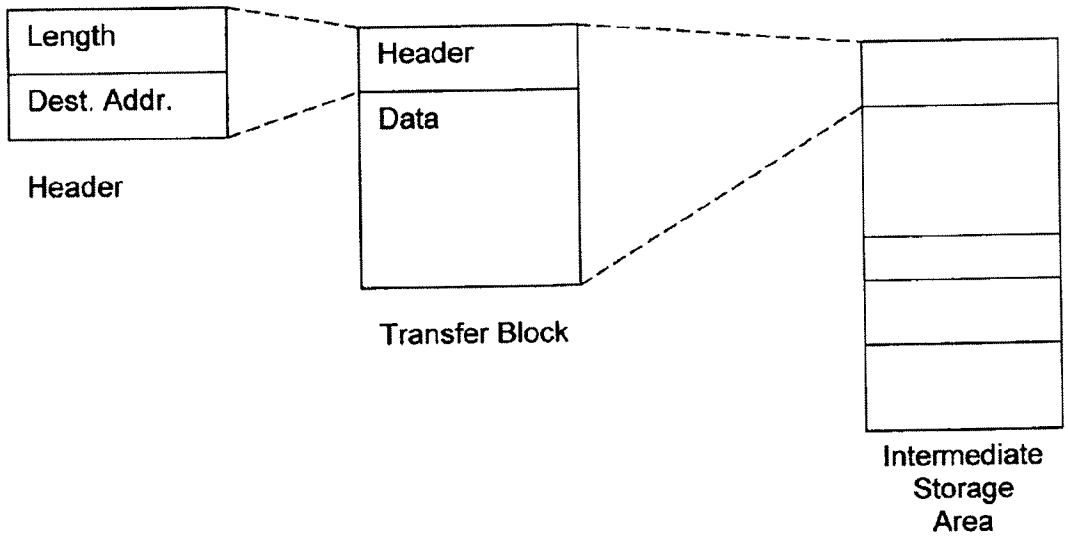


FIG. 3

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

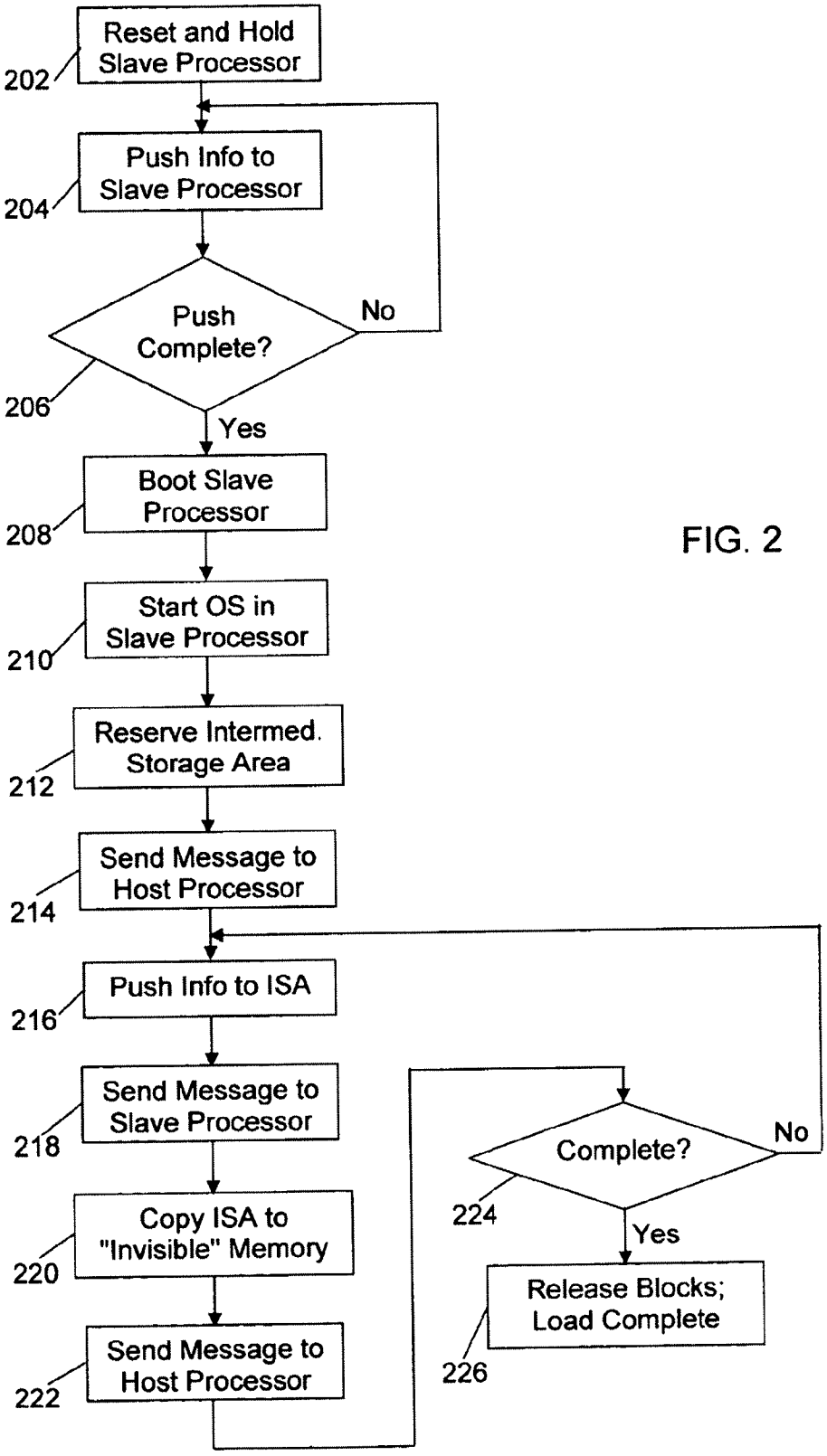


FIG. 2

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

METHOD OF LOADING INFORMATION INTO A SLAVE PROCESSOR IN A MULTI-PROCESSOR SYSTEM USING AN OPERATING-SYSTEM-FRIENDLY BOOT LOADER

BACKGROUND

This invention relates to initialization of electronic systems having multiple programmable processors.

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. An important part of the reason for this is the need for the processor to begin operation in a well-defined state.

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.

Examples of prior mechanisms for starting processor systems, including bootloaders, are U.S. Pat. No. 5,652,886 to Tulpule et al. and U.S. Pat. No. 6,490,722 to Barton et al. and U.S. Patent Application Publication No. US 2002/0138156 A1 to Wong et al. Barton et al., for example, describes a two-stage bootloader in which the second stage finds, verifies, and loads the operating system. In Wong et al., a multiprocessor system uses a master processor coupled to a ROM to transfer boot code to slave processors, with memory controllers in the slave processors denying memory access requests until the boot code has been transferred to their RAMs.

As indicated by Barton et al. and Wong et al., for example, starting up a multi-processor system, which can be generally considered as having a master or host processor, i.e., the system that orders the boot, and one or more slave or client processors, i.e., the system to be booted, is even more complicated than starting up a single-processor system.

Advantages of the "push" method are that it requires no code to execute in the slave during boot and that the only synchronization required is to hold the slave in a reset state and release it when loading is finished. Nevertheless, the "push" method works only when the memory or memories of the slave are visible to the host. This visibility can be implemented in several ways. For example, a memory may be visible on the address and data busses of both the host and the slave processors or direct memory access (DMA) transfers may be allowed from the host's memory or memories to the slave's memory or memories.

When the slave's memory to be loaded is invisible to the host, the "push" method cannot be used. In that situation, some form of bootloading must be used. As noted above, the bootloader technique requires either that boot code can be pushed onto the slave (which in this case is not possible) or that the slave can load code from a non-volatile memory. The bootloader then initiates a transfer of code from the host to the slave and finishes loading the memory.

Multi-processor systems in which some or all of a slave's memory is not visible to a host are possible. In such systems, it can be advantageous to take advantage of well-established

software frameworks for loading and inter-processor communication, which render traditional bootloaders undesirable. Moreover, a bootloader can conflict with the operating system, which can be said to want to have control over the entire system and all of the memory.

Among the problems faced when integrating a bootloader with an operating system (OS) are ensuring that code that is not yet loaded is not executed, efficiently loading code to a memory or memories invisible to the host, and synchronizing with the host the loading and booting of the slave(s). 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. Another issue that can be important is the integration of the bootloader and the OS, as an already established framework for communication between host and slave then can be used during loading. Such a framework typically would include one or more primitives for communication that rely on OS-features.

SUMMARY

This invention provides, in one aspect, a method of loading program code into a slave processor in a multi-processor system that includes a master processor and the slave processor. The method includes the steps of resetting the slave processor and holding the slave processor in a reset state; pushing information into a first memory that is accessible by the master and slave processors; booting the slave processor; starting an operating system in the slave processor, including blocking scheduling of processes having program code located in a second memory that is accessible by the slave processor and inaccessible by the master processor; reserving an intermediate storage area in the first memory; sending to the master processor information about a location and size of the intermediate storage area reserved; based on the sent information, loading the intermediate storage area with information to be loaded into the second memory; sending a first message to the slave processor that indicates the intermediate storage area has been loaded and whether loading is finished or more information is to be loaded; copying information in the intermediate storage area to the second memory; and sending a second message to the master processor that indicates that information in the intermediate storage area has been copied.

In another aspect of the invention, a multi-processor system includes a host processor, at least one client processor, a first random-access memory accessible by the host and client processors, a second random-access memory accessible by the client processor and not accessible by the host processor, and a bootloader. The first memory includes an intermediate storage area, and the bootloader includes a host part and a client part. The host part is loadable into the first random-access memory and has a first stage and a second stage. The first stage resets and holds the client processor in a reset state and pushes information into the first random-access memory. The second stage is initiated by the client part, loads the intermediate storage area with information to be loaded to the second random-access memory, and sends to the client part a first message indicating the intermediate storage area is loaded. The client part is loadable into the first random-access memory, starts an operating system including an idle process and initially blocking scheduling of all processes having program code located in the second random-access memory, copies information loaded into the intermediate storage area to the second random-access

DOCKET ALARM
Find authenticated court documents without watermarks at docketalarm.com.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.