PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 7/00, 9/40, 9/44, 9/46, 9/445, 9/45

A1

(11) International Publication Number: WO 97/24656

(43) International Publication Date: 10 July 1997 (10.07.97)

(21) International Application Number: PCT/US96/20809

(22) International Filing Date: 27 December 1996 (27.12.96)

(30) Priority Data:
08/580,808 29 December 1995 (29.12.95) US

(71) Applicant *(for all designated States except US)*: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: THURLO, Clark, S. [US/US]; 137 Winterstein Drive, Folsom, CA 95630 (US).

(74) Agents: TAYLOR, Edwin, H. et al.; Blakely, Sokoloff, Taylor & Zafman L.L.P., 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

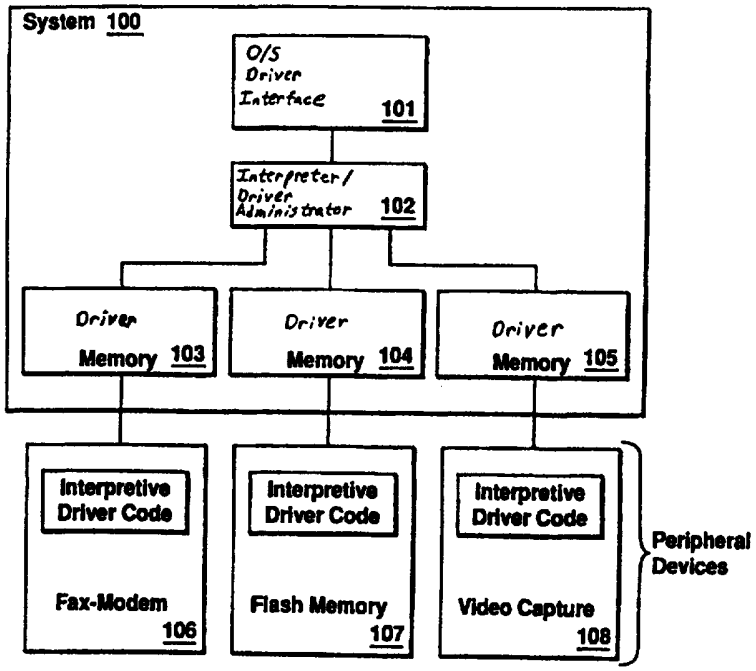
(81) Designated States: AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), EE, EE (Utility model), ES, FI, FI (Utility model), GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR PROVIDING AN INTERFACE BETWEEN A SYSTEM AND A PERIPHERAL DEVICE

(57) Abstract

A peripheral device (106, 107, 108) for use in interfacing with a system (100). The peripheral device (106, 107, 108) contains driver code stored in memory locations within the peripheral device (106, 107, 108). The driver code is uncompiled, and, when read by a system (100) to which the peripheral device (106, 107, 108) is coupled, enables the system (100) to interface with the peripheral device (106, 107, 108).



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	**GB**	United Kingdom	**MW**	Malawi
AT	Austria	**GE**	Georgia	**MX**	Mexico
AU	Australia	**GN**	Guinea	**NE**	Niger
BB	Barbados	**GR**	Greece	**NL**	Netherlands
BE	Belgium	**HU**	Hungary	**NO**	Norway
BF	Burkina Faso	**IE**	Ireland	**NZ**	New Zealand
BG	Bulgaria	**IT**	Italy	**PL**	Poland
BJ	Benin	**JP**	Japan	**PT**	Portugal
BR	Brazil	**KE**	Kenya	**RO**	Romania
BY	Belarus	**KG**	Kyrgystan	**RU**	Russian Federation
CA	Canada	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CF	Central African Republic			**SE**	Sweden
CG	Congo	**KR**	Republic of Korea	**SG**	Singapore
CH	Switzerland	**KZ**	Kazakhstan	**SI**	Slovenia
CI	Côte d'Ivoire	**LI**	Liechtenstein	**SK**	Slovakia
CM	Cameroon	**LK**	Sri Lanka	**SN**	Senegal
CN	China	**LR**	Liberia	**SZ**	Swaziland
CS	Czechoslovakia	**LT**	Lithuania	**TD**	Chad
CZ	Czech Republic	**LU**	Luxembourg	**TG**	Togo
DE	Germany	**LV**	Latvia	**TJ**	Tajikistan
DK	Denmark	**MC**	Monaco	**TT**	Trinidad and Tobago
EE	Estonia	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**UG**	Uganda
FI	Finland	**ML**	Mali	**US**	United States of America
FR	France	**MN**	Mongolia	**UZ**	Uzbekistan
GA	Gabon	**MR**	Mauritania	**VN**	Viet Nam

METHOD AND APPARATUS FOR PROVIDING AN INTERFACE BETWEEN A SYSTEM AND A PERIPHERAL DEVICE

FIELD OF THE INVENTION

The present invention relates to peripheral devices of a computer system and more particularly to a method and apparatus for providing a software interface between a system and a peripheral device.

BACKGROUND OF THE INVENTION

A typical computer system comprises one or more processors, control devices, memory, and input/output devices along with the necessary interconnects that allow these devices to communicate with one another. A processor includes, for example, a microprocessor, microcontroller, or other device that is capable of performing mathematical computations. A processor is commonly considered to be the "brain" of a computer system, while the memory and control devices support the processor by providing information storage and organizing the flow of information, respectively, within the system.

A peripheral device is typically a user-installed, optional device that adds additional performance capability to the basic computer system. For example, a disk drive may be thought of as a peripheral device. A disk drive supplements a basic computer system by providing additional memory and a means for long-term storage of information. A modem is a peripheral device that supplements a basic computer system by providing a means for communicating information to a remote location via telephone lines. Basic input and output devices, such as a

keyboard and a display screen, are examples of other peripheral devices.

Once a peripheral device is physically coupled to a computer system, generally via an electrical socket that accommodates easy coupling and decoupling, the processor, memory, and other devices of the system gain access to the peripheral device through a series of interlinks called buses. Through these buses, various parts of the system are able to communicate with the peripheral device. This communication, also known as interfacing, involves the exchange of data from one device to another.

For a computer system to interface properly with a peripheral device, both the system and the peripheral device must be speaking the same interface language. This requires that the system be aware of certain parameters of the peripheral device. For example, where the peripheral device is a memory device used for data storage, the system needs to know the amount of data that can be stored in the device so that the system can allocate and more effectively manage the flow of data sent to (written) and taken (read) from the device. The system also needs to know the proper protocol for reading or writing data this data. As another example, if the peripheral device is a modem, the system needs to know the speed at which the modem transmits data, as well as the protocol for sending data to the modem for transmission, so that the system can download data to the modem in a manner in which the modem can accept it.

The software that provides the system with the necessary parameters and enables the system to interface with the peripheral

device is called the peripheral device driver, or driver code, or, simply, driver. The source code for a driver is typically written in a high-level language or the processor's assembly language, then compiled into an object code executable file. This executable is stored on a floppy disk that accompanies the peripheral device, and a user installs the compiled driver as an executable in the memory of the computer system to which the peripheral device is to be coupled. The driver may alternatively be stored in the peripheral device itself or come pre-loaded in the computer system in either the operating system or the BIOS. Thereafter, when the peripheral device is accessed, the system uses the executable to enable the interface.

Different drivers are associated with the same type of peripheral device, depending on the make, model, and configuration of the peripheral device, and the make, model, and configuration of the system for which the peripheral device is intended. One reason for multiple drivers is that the executable driver code is compiled in a processor-specific manner. In other words, a peripheral device driver that has been compiled for one type of processor is virtually unreadable by another type of processor. Therefore, a peripheral device having driver code that has been compiled for a type "X" processor can only be used in a computer system comprising a type "X" processor. An essentially identical peripheral device having driver code that has been compiled for a type "Y" processor can only be used in a computer system comprising a type "Y" processor.

As a result, each time a user changes from one type of processor-based system to an incompatible processor-based system, such as

when upgrading or concurrently working on systems from two different companies, a new driver needs to be installed to enable interfacing with the same peripheral device. Similarly, when more advanced peripheral devices become available, along with their associated drivers, users working on older processor-based systems may be unable to take advantage these peripheral devices because the older processors may not be able to read the newer drivers that have been compiled for newer processors.

SUMMARY AND OBJECTS OF THE INVENTION

An object of the present invention is to provide a method for interfacing between a system and a peripheral device.

Another object of the present invention is to provide a method by which an interface between a system a peripheral device is processor-independent.

A peripheral device is described for use in interfacing with a system. The peripheral device contains driver code stored in memory locations within the peripheral device. The driver code is uncompiled, and, when read by a system to which the peripheral device is coupled, enables the system to interface with the peripheral device.

Other features and advantages of the present invention will be apparent from the accompanying drawings and the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings in which like references indicate similar elements and in which:

Figure 1 is a block diagram of a system to which peripheral devices are coupled.

Figure 2 is a sample of interpretive driver code.

DETAILED DESCRIPTION

A peripheral device is described along with a method by which driver code is provided that enables the peripheral device to interface with the system to which it is coupled. The peripheral device contains its associated driver code stored in memory locations within the peripheral device. The driver code is uncompiled, meaning that the driver code can be either source code or interpretive code. For the embodiment in which the driver code is source code, the code is first read by the system to which the peripheral device is coupled, and is then stored in memory locations within the system. The system contains an appropriate compiler for compiling the source code into an executable file, which is then executed by the system, thereby enabling the system to interface with the peripheral device.

For the embodiment in which the driver code is interpretive code, the code is first read by the system to which the peripheral device is coupled, and is then stored in memory locations within the system. The system contains an appropriate interpreter for interpreting the source

code, thereby enabling the system to interface with the peripheral device. A peripheral device and method of providing an interface between the peripheral device and a system, will be described in more detail below.

Figure 1 is a block diagram of a system 100 to which peripheral devices 106-108 have been coupled. System 100 comprises an operating system driver interface 101, which is coupled to an interpreter and driver administrator 102, which is coupled to three memory locations (or access thereto) 103-105. Each memory location 103, 104, and 105 interfaces with each of peripheral devices 106, 107, and 108 respectively.

Operating system driver interface 101 provides the interface between the commands initiated by the system user and the peripheral devices. For example, for an embodiment in which peripheral memory cards, such as Personal Computer Memory Card International Association-compliant (PCMCIA) cards, PC or CardBus cards, are to be accessed, the operating system driver interface comprises a file system/media manager and a card services interface. The file system/media manager partitions and manages the memory media by, for example, providing for directory and subdirectory levels for a user, and associating each of these levels to memory locations. The card services interface, based on the memory location indicated by a read or write command initiated by the file system/media manager, determines which of a plurality of peripheral memory cards is being accessed, and passes the command along to the proper technology driver. The system

then interfaces with the appropriate memory card through the associated driver to execute the command.

Peripheral device 106 is a fax-modem that is capable of converting data from system 100 into telephone signals for transmitting the data to a remote location. Fax-modem 106 is also capable of transmitting data received via a telephone signal back into the system. Therefore, fax-modem 106, in addition to being coupled to system 100, is also coupled to a telephone line (not shown). Peripheral device 107 is flash memory that provides system 100 with non-volatile, flash memory for storage of data. Peripheral device 108 is a video capture device that digitizes incoming images from a camera and provides the data associated with those images to system 100 for manipulation or storage. For one embodiment of the present invention, system 100 is a desktop or mobile computer system, and peripheral devices 106 - 108 are compliant with the PCMCIA or CardBus standards. Many of the terms and operations mentioned herein may be found in a standard PCMCIA driver specification.

Each of peripheral devices 106 - 108 comprises memory locations large enough to store the driver code for the associated peripheral device. As indicated in Figure 1, within the peripheral device memory of each of peripherals 106 - 108 is stored its associated driver as interpretive code, which is capable of being interpreted by an interpreter. Also as indicated, the system to which a peripheral device is coupled contains or has access to driver memory locations within the system. The driver code stored in the memory of a peripheral device is

read into the system by the driver administrator and stored in the appropriate driver memory location within the system.

The interpretive driver code is stored in protected regions of memory of the associated peripheral device. A protected region of memory is a memory location from which the contents of the memory cannot be readily erased or written-over, making it difficult to corrupt the interpretive driver code stored in peripheral device memory through normal use and operation of the peripheral device.

System 100, before transferring the interpretive driver code of a peripheral device into system memory, must first be apprised of the basic information needed to load the driver code into driver memory locations 103 - 105. For example, the driver administrator must know where the driver code resides within the memory of a particular peripheral device. This basic information may be obtained either from predefined memory locations with the peripheral device, or via previously installed set-ups within the system. After the interpretive driver code has been read from the peripheral device and stored in system memory, communication with the peripheral device is primarily conducted in accordance with the software interface established by the driver.

After the proper interpretive driver code, which may be the entire code stored in the peripheral device or some portion thereof, has been transferred from the peripheral device 106 - 108 to driver memory locations 103 - 105 within the system, the interpreter in the interpreter/driver administrator 102 of the system interprets the code as needed to interface with the peripheral device. Interpretive code is code

that need not be compiled before executing it. BASIC is one example of interpretive code. Interpretive languages are usually generic across various processors. So, for example, a driver written in BASIC for a system comprising processor "X" will also run on a system comprising processor "Y," notwithstanding incompatibilities between processors "X" and "Y," as long as a BASIC interpreter is provided in each system. An interpreter, which may be processor-dependent, is the tool through which an interpretive language is executed.

For an alternate embodiment of the present invention, the peripheral devices coupled to a system contain source code rather than interpretive code as the device driver software. Source code, before being executed, is typically compiled into an object code executable. Therefore, for this embodiment, the system contains (or has access to) a compiler. Upon transferring the source code driver from the peripheral device by reading it into driver memory locations of system memory, the source code is compiled into object code by the compiler, and is stored in system memory as an executable file. The system then executes the file as needed to interface with the peripheral device. Source codes, like interpretive codes, are usually generic across various processors. So, for example, a source code driver written for a system comprising processor "X" will also run on a system comprising processor "Y," notwithstanding incompatibilities between processors "X" and "Y," as long as a compiler, which may be processor-dependent, is provided in each system.

As one example of a useful application in accordance with the present invention, a system, comprising a digital camera, uses flash

storage devices, or flash cards, as the film upon which digital images from the digital camera are stored. For one embodiment, each flash card contains its associated driver code stored in an interpretive language in memory within the card. The digital camera comprises an interpreter/driver administrator and memory.

After a picture is taken, the digital camera stores the data associated with the picture into the flash card by calling on the operating system driver interface to write the data into the attached flash card. The driver administrator reads the section of memory within the flash card containing the desired driver code related to write operations, and stores this code in memory within the camera. The camera then proceeds to execute or run this code using the interpreter, thereby enabling an interface between the camera and the flash card, storing the data in the flash card. For an alternate embodiment, the system is an audio recording device which stores digital audio signals in peripheral flash memory. For another embodiment, the system is a personal data assistant (PDA) or cellular phone which uses peripheral devices to perform functions such as storing or displaying data.

As stated above, BASIC is one type of interpretive language. BASIC, however, is a relatively extensive language, resulting in a substantially large interpreter to interpret the language and fairly high computational requirements. In the interest of reducing the size of the interpreter that must be stored in a system, and reducing the load on the processor of the system, a new interpretive language is developed, along with an associated interpreter, which is optimized for the particular application for which the interpretive driver code will be used. In doing

so, the size and the processing requirements of the interpreter and the interpretation, respectively, are reduced.

For example, Figure 2 is a sample of interpretive driver code that has been written in a new interpretive language that has been optimized for flash memory peripherals. Only the lines of code associated parameter settings and the write operation are shown here. A more complete interpretive driver code sequence would additionally include, for example, lines of code associated with the read and erase operations.

The first line of code of Figure 2, "PARAMETER_START" indicates that this is the section of the code that provides parameter information related to the parameters of the flash card. The line "SPEED = 200" indicates that the flash card's access speed is 200ns. The line "BLOCK = 64K" indicates that the block size of the flash card is 64KB. "DEVICE = 1M" indicates that the size of the flash devices in the flash card are 1MB. "SIZE = 2M" indicates that the total memory size of the flash card is 2MB. WIDTH = 08" indicates that this is a x8 card.

The next line of code of Figure 2, "WRITE_START" indicates that this is the section of the code that provides the protocol for executing a write operation to the flash card. A write command is initiated by the operating system driver interface of the system, in response to, for example, a system user command. The write command is directed to the interpreter/driver administrator in the form of a write request packet comprising 1) the request type (write in this case); 2) the number of bytes to be written (called "number_bytes" in Figure 2); 3) the destination address of the data (called "destination_address" in Figure 2

and refers to the flash card in this case); and 4) the source address of the data (called "source_address" in Figure 2 and usually refers to a buffer in system memory). In response to a write command, the interpreter/driver administrator, if it hasn't already done so, will load either the entire interpretive driver code or only the write portion of the code from the peripheral device into the appropriate driver memory location with the system. The interpreter/driver administrator then locates the "WRITE_START" line in the interpretive drive code within the system, and begins to execute what follows.

In the first two lines of code within the write section, "PCMCIA_CALL SET VPP = 12V" and "PCMCIA_CALL SET VCC = 5V" the code instructs the system to set the appropriate voltage levels to the peripheral device to enable a write to the flash memory. Next, a variable "offset" is set equal to 0. This variable will determine the offset from the original destination and source addresses, provided in the original write request packet, into which the associated data will be written. "LABEL 'BEGIN'" labels this line of the code for reference by subsequent JUMP commands, described below.

"OUTPUT destination_address + offset, write_command" instructs that the data or control commands following the "OUTPUT" command be sent to the flash card. In this case, the destination address sent with the write request packet is offset by the "offset" variable (0 in the first iteration) and is sent a "write_command," which is a flash card-specific command that indicates that the next write to that address contains data to be stored. "OUTPUT destination_address + offset, source_address + offset" instructs that the data stored in the source address, offset by the

"offset" variable, is to be stored at the destination address, also offset by the "offset" variable.

Once data has been written in accordance with the two "OUTPUT" commands described above, the system must wait until the flash card is ready to accept another write command before continuing. "counter = 0" sets the variable "counter" equal to 0. The "counter" variable is used to pause the system while the flash card completes the previous write operation. "LABEL 'VERIFY'" labels this line of code for subsequent reference by a JUMP command. "COMPARE card_status, write_valid" is a command indicating that if the value of "card_status" is equal to the value of "write_valid," then the variable "flag" will be set equal to 1. Otherwise "flag" will be set equal to 0. "Card_status" is flash card-dependent value that the flash card provides to the system to indicate whether or not the previous OUTPUT command to write data into the flash card executed properly. "Write_valid" is also a flash card-dependent value that is equal to the value the flash card provides to indicate a successful write operation has taken place.

If "card_status" is equal to "write_valid," then the write to the flash card has been successful, and the variable "flag" will be set to 1. Then, in the next line, the interpreter is instructed to jump to the line of code labeled "VERIFIED". If, on the otherhand, "card_status" is not equal to "write_valid," then the write to the flash card has not been successful, and the variable "flag" will be set to 0. The next line of code will warrant no action on the interpreter's part, and the counter will be incremented in the following line of code "INCREMENT counter". The INCREMENT

command sets the value of the indicated variable equal to the current value of the variable plus one, thereby incrementing its value by one.

Next the interpreter determines if the counter has reached 100, and if so, the interpreter is instructed to jump back up to the line labeled "BEGIN" so that the data can be rewritten. The higher the value of the variable "counter," the longer the delay. Therefore, the counter limitation in this line of code should be set to a reasonable value by which time data can be written into the flash card. If counter hasn't reached 100 yet, the interpreter is instructed to increment the counter in the next line, and loop back to the label "VERIFY" to continue checking the flash card status.

The variable "offset" is incremented in the command line "INCREMENT offset," and its value is then compared to the total number of bytes, "number_bytes" that the original write request data packed indicated would be written to the flash card. If "offset" is not equal to "number_bytes," thereby setting "flag" to 0, the interpreter is instructed to jump back up to the line of code labeled "BEGIN" so that the remaining bytes can be written into the flash card. Once "offset" reaches "number_bytes," "flag" will be set equal to 1, and the line "If flag = 1 THEN JUMP DONE" will instruct the interpreter to skip down two lines to the line labeled "DONE". The following line of code "WRITE_END" indicates to the interpreter that the write to the flash card has been completed. Thus, in this manner, the interpretive driver code, originally resident on the peripheral device, has provided a software interface between the system and peripheral, thereby enabling communication for the execution of a write operation.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A peripheral device comprising:

memory; and

driver code stored in the memory, the driver code being uncompiled code that, when read by a system to which the peripheral device is coupled, enables the system to interface with the peripheral device.

2. The peripheral device of claim 1, wherein the code is stored in the memory as interpretive code capable of being interpreted by an interpreter contained within the system.

3. The peripheral device of claim 1, wherein the code is stored in the memory as source code capable of being compiled and executed by the system.

4. The peripheral device of claim 1, wherein the peripheral device comprises flash memory.

5. The peripheral device of claim 4, wherein the system comprises a digital camera.

6. The peripheral device of claim 1, wherein the peripheral device comprises a fax-modem device.

7. The peripheral device of claim 1, wherein the peripheral device is a video capture device.

8. A flash memory comprising:

a protected region of memory within the flash memory; and

driver code stored in the protected region of memory, the driver code being uncompiled code that, when read by a system to which the flash memory is coupled, enables the system to interface with the flash memory.

9. The flash memory of claim 8, wherein the code is stored as interpretive code.

10. The flash memory of claim 8, wherein the code is stored as source code.

11. The flash memory of claim 8, wherein the flash memory stores digital images from the system, the system comprising a digital camera.

12. The flash memory of claim 8, wherein the flash memory stores digital audio signals from the system.

13. An apparatus wherein a system is coupled to and interfaces with a peripheral device, the apparatus comprising:

peripheral device memory;

system memory coupled to the peripheral device memory; and

driver code stored in the peripheral device memory, the driver code being uncompiled code that, when read by the system, enables the system to interface with the peripheral device.

14. The apparatus of claim 13, wherein the code is stored in the peripheral device memory as interpretive code, which is then transferred to system memory, and is interpreted by an interpreter contained within the system to enable interfacing with the peripheral device.

15. The apparatus of claim 13, wherein the code is stored in the peripheral device memory as source code, which is then transferred to system memory, compiled by a compiler contained within the system, and is executed by the system to enable interfacing with the peripheral device.

16. The apparatus of claim 13, wherein the peripheral device comprises flash memory.

17. The apparatus of claim 16, wherein the system comprises a digital camera and wherein the peripheral device stores digital images from the system.

18. The apparatus of claim 14, wherein the peripheral device comprises flash memory, the system comprises a digital camera, and the peripheral device stores digital images from the system.

19. The apparatus of claim 14, wherein the peripheral device comprises flash memory, the system comprises an audio recording device, and the peripheral device stores digital audio signals from the system.

20. The apparatus of claim 13, wherein the peripheral device comprises a video capture device.

21. A method of interfacing a peripheral device to a system, the method comprising the steps of:

reading memory locations within the peripheral device containing uncompiled driver code;

storing the driver code in memory locations within the system;

using the driver code to enable the system to interface with the peripheral device.

22. The method of claim 21, wherein the code is stored in the peripheral device as interpretive code, which is interpreted by an interpreter contained within the system to enable the system to interface with the peripheral device.

23. The method of claim 21, wherein the code is stored in the peripheral device as source code, which is compiled and executed by the system to enable the system to interface with the peripheral device.

24. The method of claim 21, wherein the peripheral device comprises flash memory.

25. The method of claim 24, further comprising the step of storing digital images in the peripheral device, wherein the system comprises a digital camera.

26. The method of claim 22, further comprising the step of storing digital images in the peripheral device, wherein the system comprises a digital camera and the peripheral device comprises flash memory.

27. The method of claim 21, wherein the peripheral device comprises a fax-modem device.

28. The method of claim 21, wherein the peripheral device comprises a video capture device.

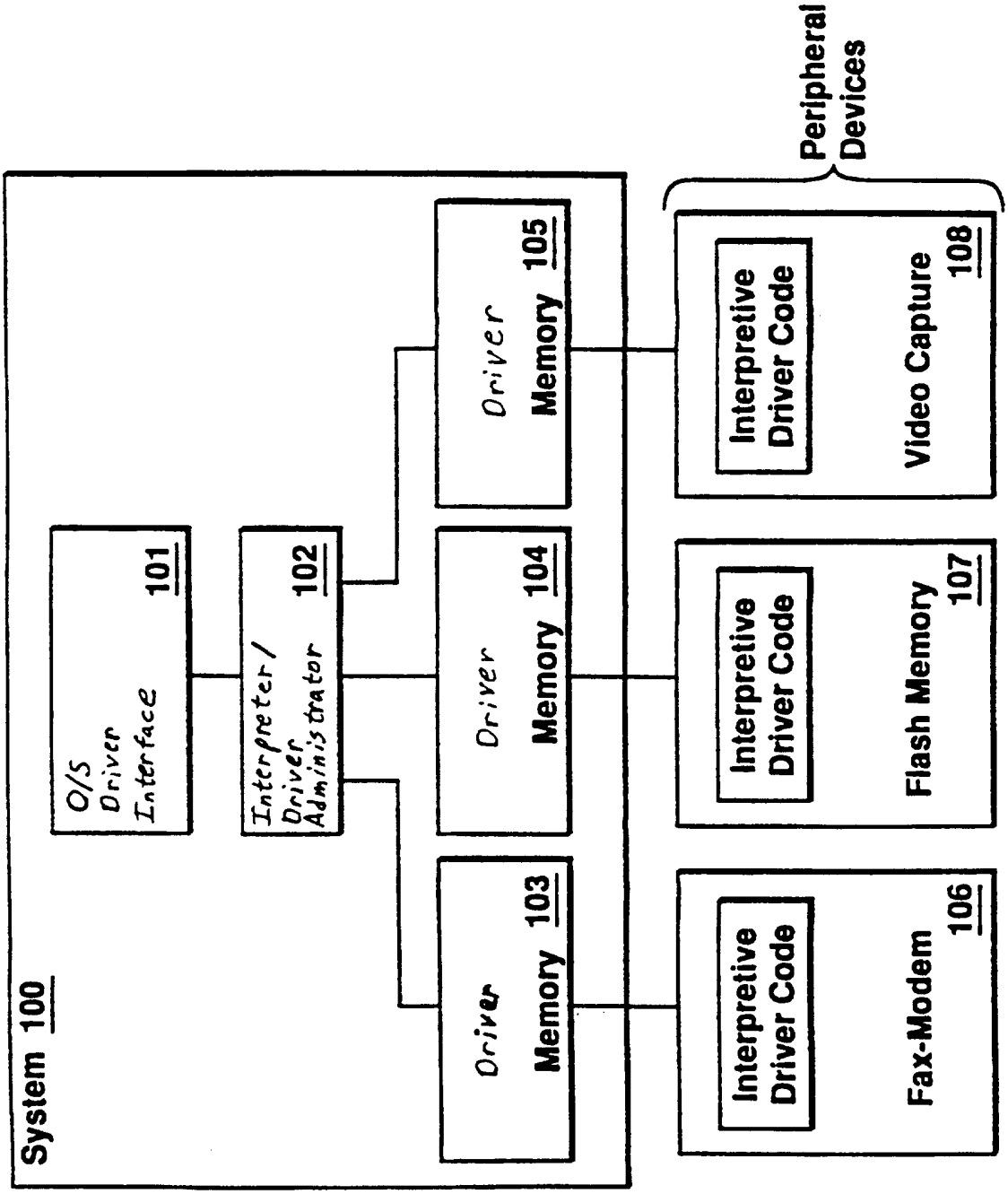

System 100
O/S Driver Interface 101
Interpreter/ Driver Administrator 102
Driver Memory 103
Driver Memory 104
Driver Memory 105
Peripheral Devices
Interpretive Driver Code
Fax-Modem 106
Interpretive Driver Code
Flash Memory 107
Interpretive Driver Code
Video Capture 108

Figure 1

```
        •
        •
        •
PARAMETER_START
  SPEED = 200
  BLOCK = 64K
  DEVICE = 1M
  SIZE = 2M
  WIDTH = 08
PARAMETER_END
        •
        •
        •

WRITE_START
  PCMCIA_CALL SET VPP = 12V
  PCMCIA _CALL SET VCC = 5V
  offset = 0
  LABEL "BEGIN"
     OUTPUT destination_address + offset, write_command
     OUTPUT destination_address + offset, source_address + offset
     counter = 0
     LABEL "VERIFY"
        COMPARE card_status, write_valid
        IF flag = 1 THEN JUMP "VERIFIED"
        IF counter = 100 THEN JUMP "BEGIN"
        INCREMENT counter
        JUMP VERIFY
        LABEL "VERIFIED"
     INCREMENT offset
     COMPARE offset, number_bytes
     IF flag = 1 THEN JUMP "DONE"
     JUMP "BEGIN"
     LABEL "DONE"
WRITE_END
        •
        •
        •
```

Figure 2

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/20809

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 7/00, 9/40, 9/44, 9/46, 9/445, 9/45
US CL :395/651, 652, 712
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/651, 652, 712, 701, 705, 706, 707

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

STN, DIALOG

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US, A, 5,519,851 (BENDER ET AL) 21 MAY 1996, FIG'S 1-13 AND COL 1, LINE 5, TO COL 16, LINE 36.	1-28
Y,P	US, A, 5,504,902 (MCGRATH ET AL) 02 APRIL 1996, FIG'S 1-8 AND COL 1, LINE 5, TO COL 16, LINE 17.	1-28
Y	US, A, 5,404,494 (GARNEY) 04 APRIL 1995, SEE ENTIRE DOCUMENT.	1-28
Y	US, A, 5,319,751 (GARNEY) 07 JUNE 1994, FIG'S 1-13 AND COL 1, LINE 5, TO COL 18, LINE 17.	1-28
Y	US, A, 5,412,798 (GARNEY) 02 MAY 1995, FIG'S 1-17 AND COL 1, LINE 20, TO COL 26, LINE 44.	1-28

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 09 APRIL 1997	Date of mailing of the international search report 23 APR 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 308-5356	Authorized officer PETER J. CORCORAN Telephone No. (703) 308-6685

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US96/20809

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,265,218 (TESTA ET AL) 23 NOVEMBER 1993, FIG'S 1-24 AND COL 1, LINE 10, TO COL 20, LINE 52.	1-28