EXHIBIT

49

11-14-00 A/Prov

JC965 U.S. PTO 11/10/00

JC804 U.S. PTO 60/247844 11/10/00

Practitioner's Docket No. 03797.00072 *PATENT*

Preliminary Classification Proposed Class: Subclass:

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Keely, Leroy B.

For: Simulating Gestures of a Mouse Using a Stylus and Providing Feedback Thereto

Box Provisional Patent Application
Assistant Commissioner for Patents
Washington, D.C. 20231

COVER SHEET FOR FILING PROVISIONAL APPLICATION
(37 C.F.R. SECTION 1.51(c)(1))

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 C.F.R. section 1.51(c)(1)(i). The following comprises the information required by 37 C.F.R. Section 1.51(c)(1):

CERTIFICATION UNDER 37 C.F.R. SECTION 1.10*
*(Express Mail label number is **mandatory**.)*
(Express Mail certification is optional.)

I hereby certify that this correspondence and the documents referred to as attached therein are being deposited with the United States Postal Service on November 10, 2000, in an envelope as "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 C.F.R. Section 1.10, Mailing Label Number EL566709730US Addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Jordan N. Bodner
(type or print name of person mailing paper)

[illegible]
Signature of person mailing paper

WARNING: *Certificate of mailing (first class) or facsimile transmission procedures of 37 C.F.R. Section 1.8 cannot be used to obtain a date of mailing or transmission for this correspondence.*

WARNING:* *Each paper or fee filed by "Express Mail" **must have the number of the "Express Mail" mailing label placed thereon prior to mailing. 37 C.F.R. Section 1.10(b).*
*"Since the filing of correspondence under [Section] 1.10 without the Express Mail mailing label thereon is an oversight that can be avoided by the exercise of reasonable care, requests for waiver of this requirement will **not** be granted on petition." Notice of Oct. 24, 1996, 60 Fed. Reg. 56,439, at 56,442.*

Practitioner's Docket No. 03797.00072

1. The name of the inventor is (37 C.F.R. Section 1.51(c)(1)(ii)):

1. Leroy B. Keely

2. Residence address of the inventor, as numbered above (37 C.F.R. Section 1.51(c)(1)(iii)):

1. 210 Gabarda Way
Portola Valley, CA 94028

3. The title of the invention is (37 C.F.R. Section 1.51(c)(1)(iv)):

Simulating Gestures of a Mouse Using a Stylus and Providing Feedback Thereto

4. The name, registration, customer and telephone numbers of the practitioner are (37 C.F.R. Section 1.51(c)(1)(v)):

Name of practitioner: Pamela I. Banner
Reg. No. 33,644
Tel. No. 202-508-9100

5. The docket number used to identify this application is (37 C.F.R. Section 1.51(c)(1)(vi)):

Docket No. 03797.00072

6. The correspondence address for this application is (37 C.F.R. Section 1.51(c)(1)(vii)):

Pamela I. Banner
1001 G Street, N.W.
Washington, DC 20001

7. Statement as to whether invention was made by an agency of the U.S. Government or under contract with an agency of the U.S. Government. (37 C.F.R. Section 1.51(c)(1)(viii)).

This invention was NOT made by an agency of the United States Government, or under contract with an agency of the United States Government.

8. Identification of documents accompanying this cover sheet:

A. Documents required by 37 C.F.R. Section 1.51(c)(2)-(3):

Specification:	No. of pages	16 Pages (including claims)
Drawings:	No. of sheets	3 (Figs. 1, 2, and 3)

B. Additional documents:

Claims:	No. of claims	3

9. Fee

The filing fee for this provisional application, as set in 37 C.F.R. Section 1.16(k), is $150.00 for other than a small entity.

10. Fee payment

Fee payment in the amount of $150.00 is being made at this time.

11. Method of fee payment

Charge Account No. 19-0733, in the amount of $150.00.

A duplicate of this Cover Sheet is attached.

Please charge Account No. 19-0733 for any fee deficiency.

Date: November 10, 2000

[illegible] Reg. No. 42,338
Signature of Practitioner

Reg. No.: 33,644
Tel. No.: 202-508-9100

Pamela I. Banner
Banner & Witcoff, Ltd.
1001 G Street, N.W.
Washington, DC 20001

Express Mail Certificate EL566709730US

SIMULATING GESTURES OF A MOUSE USING A STYLUS AND PROVIDING FEEDBACK THERETO

PROVISIONAL PATENT APPLICATION

Cross-Reference to Related Applications

The present application is related to application Serial No. (Atty docket 3797.00066), entitled Method and Apparatus For Improving the Appearance of Digitally Represented Handwriting, filed concurrently with the present application; to application Serial No. (Atty docket 3797.00067), entitled Highlevel Active Pen Matrix, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00069), entitled Selection Handles in Editing Electronic Documents, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00070), entitled Insertion Point Bungee Space Tool, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00074), entitled System and Method for Accepting Disparate Types of User Input, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00075), entitled In Air Gestures, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00076), entitled Mouse Input Panel Windows Class List, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00077), entitled Mouse Input Panel and User Interface, and filed concurrently with the present application; to application Serial No. (Atty docket 3797.00079), entitled System and Method for Inserting Implicit Page

Breaks, and filed concurrently with the present application; each of which is incorporated by reference herein as to their entireties.

Field of the Invention

The present invention is directed generally to apparatus and methods for simulating a various gestures of a mouse in a computer system and for providing feedback thereto, and more specifically to simulating gestures such as the left click of a mouse, the right click of a mouse, and mouse dragging by manipulation of a stylus in conjunction with a touch-sensitive computer display, as well as generating appropriate visual or other feedback in response to certain gestures.

Background of the Invention

Typical computer systems, especially computer systems using graphical user interface (GUI) systems such as Microsoft WINDOWS, are optimized for accepting user input from two discrete input devices: a keyboard and for entering text, and a pointing device such as a mouse with two or more buttons for driving the user interface. Virtually all software applications designed to run on Microsoft WINDOWS are optimized to accept user input in the same manner. For instance, many applications make extensive use of the right mouse button (a "right click") to display context-sensitive command menus. It is noted that other operating systems incorporate right click operability as well. The user may generate other gestures using the mouse such as by clicking the left button of the mouse (a "left click"), or by clicking the left or right button of the mouse and moving the mouse while the button is depressed (either a "left click drag" or a "right click drag").

While such mouse gestures are easily done with a mouse, it is not always convenient for a user to utilize a mouse with a computer even though the computer may have a GUI system. Once such type of system may be that which utilizes a touch-sensitive display screen and a stylus. The stylus tip is placed upon the display screen at certain locations to control the objects on the display screen. The stylus is limited in the types of actions that may be easily performed as compared with a mouse. For instance, the typical stylus can thus perform only three actions: placing the stylus tip onto the screen, moving the stylus tip across the screen, and removing the stylus tip from the screen. While a user may operate a remote toolbar to select a different tool, the excessive movement renders the excess movement laborious, inconvenient, and slow. There is presently no convenient way to simulate, for example, a right click as opposed to a left click using a stylus-type input device.

In order to make the full range of interface features accessible to users of such a stylus-based computer, there is a need for an intuitive way of simulating two-or-more button mouse gestures with the stylus. Moreover, it is preferable that any new way of simulating mouse gestures be compatible with existing software applications that conventionally are used with a mouse.

There is also a need for providing helpful feedback to the user of the stylus to indicate whether gestures made with the stylus are those that are intended by the user.

Summary of the Invention

As discussed in the various copending patent applications incorporated herein by reference, aspects of the present invention are directed to a tablet-like computer that may be used for directly writing on a touch-sensitive display surface using a stylus. The computer may allow the user to write and to edit, manipulate, and create objects through the use of the stylus. Many of the features discussed in these copending applications are more easily performed by use of the various aspects of the present invention discussed herein.

An aspect of the present invention is directed to methods and apparatus for simulating gestures of a mouse by use of a stylus on a touch-sensitive display surface. For example, the left click of a mouse may be simulated where the user holds the stylus down on the touch-sensitive display surface without substantial movement and then removes the stylus from the display surface before the expiration of a threshold amount of time. Also, the right click of a mouse may be simulated where the user instead waits until at least the threshold amount of time before removing the stylus. Where the stylus is moved along the display surface during these above stylus gestures, left or right click dragging may thereby be performed.

Another aspect of the present invention is directed to methods and apparatus for providing feedback to a user in a stylus-based touch-sensitive display surface computer system. The user may accordingly be given some indication that they have performed a stylus gesture correctly or incorrectly by, e.g., displaying something on the display surface to indicate a particular status. Such feedback may be responsive to certain stylus gestures such as pressing and holding the stylus against the display surface for at least a certain minimum amount of time. Also, the feedback may be in the form of a "feedback indicator," which may be visual on the display surface such as an icon, bitmap, or other visual indicator, and/or auditory such as a sound emitted from the computer's speakers. The feedback indicator may change with stylus hold time and/or may be animated. Significant advantages of this aspect of the invention are realized especially in conjunction with the various features discussed in the copending patent applications incorporated herein by reference.

These and other features of the invention will be apparent upon consideration of the following detailed description of preferred embodiments. Although the invention has been defined using the appended claims, these claims are exemplary in that the invention is intended to include the elements and steps described herein in any combination or subcombination. Accordingly, there are any number of alternative combinations for defining the invention, which incorporate one or more elements from the specification, including the description, claims, and drawings, in various combinations or subcombinations. It will be apparent to those skilled in the relevant technology, in light of the present specification, that alternate combinations of aspects of the invention, either alone or in combination with one or more elements or steps defined herein, may be utilized as modifications or alterations of the invention or as part of the invention. It is intended that the written description of the invention contained herein covers all such modifications and alterations.

Brief Description of the Drawings

The foregoing summary of the invention, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the accompanying drawings, which are included by way of example, and not by way of limitation with regard to the claimed invention. In the accompanying drawings, elements are labeled with three-digit reference numbers, wherein the first digit of a reference number indicates the drawing number in which the element is first illustrated. The same reference number in different drawings refers to the same element.

Fig. 1 is a schematic diagram of a general-purpose digital computing environment that can be used to implement various aspects of the invention.

Fig. 2 is a plan view of a tablet computer and stylus that can be used in accordance with various aspects of the present invention.

Fig. 3 is a flow chart showing an exemplary set of steps that may be performed in order to simulate a right click of a mouse according to aspects of the present invention.

Detailed Description of Preferred Embodiments

FIG. 1 shows an exemplary embodiment of a general-purpose digital computing environment that can be used to implement various aspects of the invention. In this embodiment, computer 100 includes a processing unit 110, a system memory 120, and a system bus 130 that couples various system components including the system memory to the processing unit 110. The system bus 130 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 140 and random access memory (RAM) 150.

A basic input/output system 160 (BIOS), containing the basic routines that help to transfer information between elements within the computer 100, such as during start-up, is stored in ROM 140. Computer 100 also includes a hard disk drive 170 for reading from and writing to a hard disk (not shown), a magnetic disk drive 180 for reading from or writing to a removable magnetic disk 190, and an optical disk drive 191 for reading from or writing to a removable optical disk 192 such as a CD ROM or other optical media. The hard disk drive 170, magnetic disk drive 180, and optical disk drive 191 are connected to the system bus 130 by a hard disk drive interface 192, a magnetic disk drive interface 193, and an optical disk drive interface 194, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 100. It will be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the example operating environment.

A number of program modules can be stored on the hard disk, magnetic disk 190, optical disk 192, ROM 140 or RAM 150, including an operating system 195, one or more application programs 196, other program modules 197, and program data 198. A user can enter commands and information into the computer 100 through input devices such as a keyboard 101 and pointing device 102. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 110 through a serial port interface 106 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 107 or other type of display device is also connected to the system bus 130 via an interface, such as a video adapter 108. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The computer 100 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 109. Remote computer 109 can be a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 100, although only a memory storage device 111 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 112 and a wide area network (WAN) 113. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 100 is connected to the local network 112 through a network interface or adapter 114. When used in a WAN networking environment, the personal computer 100 typically includes a modem 115 or other means for establishing a communications over the wide area network 113, such as the Internet. The modem 115, which may be internal or external, is connected to the system bus 130 via the serial port interface 106. In a networked environment, program modules depicted relative to the personal computer 100, or portions thereof, may be stored in the remote memory storage device.

It will be appreciated that the network connections shown are example and other means of establishing a communications link between the computers can be used. The existence of any of various well-known protocols such as TCP/IP, Ethernet, FTP, HTTP and the like is presumed, and the system can be operated in a client-server configuration to permit a user to retrieve web pages from a web-based server. Any of various conventional web browsers can be used to display and manipulate data on web pages.

Figure 2 illustrates a tablet PC 201 that can be used in accordance with various aspects of the present invention. Any or all of the features, subsystems, and functions in the system of Figure 1 can be included in the computer of Figure 2. Tablet PC 201 includes a large display surface 202, e.g., a digitizing flat panel display, preferably, a liquid crystal display (LCD) screen, on which a plurality of windows 203 is displayed. Using stylus 204, a user can select, highlight, and write on the digitizing display area. Examples of suitable digitizing display panels include electromagnetic pen digitizers, such as the Mutoh or Wacom pen digitizers. Other types of pen digitizers, e.g., optical digitizers, may also be used. Tablet PC 201 interprets marks made using stylus 204 in order to manipulate data, enter text, and execute conventional computer application tasks such as spreadsheets, word processing programs, and the like.

A stylus could be equipped with or without buttons or other features to augment its selection capabilities. In one embodiment, a stylus could be implemented as a "pencil" or "pen", in which one end constitutes a writing portion and the other end constitutes an "eraser" end, and which, when moved across the display, indicates portions of the display are to be erased. Other types of input devices, such as a mouse, trackball, or the like could be used. Additionally, a user's own finger could be used for selecting or indicating portions of the displayed image on a touch-sensitive or proximity-sensitive display. Consequently, the term "user input device", as used herein, is intended to have a broad definition and encompasses many variations on well-known input devices.

Referring now to Fig. 3, an example of how a left click drag of a mouse is simulated is now discussed. Any or all of steps set forth herein and in Fig. 3 may be performed by special software that runs in conjunction with, or is part of, the operating system of the computer 201. To simulate a left click drag, the user may touch the stylus 204 or other similar user input device to the display surface 202, an action referred to hereafter as "placing the stylus down" (step 301). Responsive to detecting the stylus being placed down, the computer 201 may begin counting time, e.g., by using a timer, up to a threshold amount of time. A timeout condition is defined responsive to the threshold amount of time having passed. In a preferred embodiment, the threshold amount of time is about 600 milliseconds. However, the threshold amount of time may be any amount of time, such as between 300 and 600 milliseconds, or between 600 milliseconds and 1 second, or greater then 1 second. If the stylus moves prior to the timeout condition occurring (step 302), then in response the computer 201 may generate a standard Microsoft WINDOWS LeftMouseButtonDown event in the original position where the stylus 204 was first brought down (step 314), which may be sent to the software application presently in use. At this point, any subsequent movement of the stylus 204 across the display surface 202 may be detected (step 315), an in response the computer 201 may generate one or more Microsoft WINDOWS MouseMove events (step 316). When the stylus 204 is eventually removed from the display surface 202 (hereafter referred to as "bringing the stylus up") (step 317), the computer 201 in response may generate a Microsoft WINDOWS LeftMouseButtonUp event (step 318). Thus, via steps 301, 302, and 314-318, a left click drag of a mouse is simulated.

To simulate a left click of a mouse without dragging, if the stylus 204 is not moved prior to the timeout condition (step 302), but if the stylus 204 is instead brought up prior to the timeout condition (step 303), then in response the computer 201 may generate first a LeftMouseButtonDown event and then a LeftMouseButtonUp event (steps 304 and 318). Thus, via steps 301-304 and 318, a slightly delayed left click of a mouse is simulated.

To simulate a right click of a mouse without dragging, if the stylus 204 is not moved either prior to the timeout condition or after the timeout condition (steps 302 and 305), and if the stylus 204 is not brought up until after the timeout condition (steps 303 and 306), then in response the computer 201 may generate first a Microsoft WINDOWS RightMouseButtonDown event (step 307), then delay for a period of time (step 308), preferably about 20 milliseconds, and then generate a Microsoft WINDOWS RightMouseButtonUp event (step 313). The computer 201 may also generate and display a feedback indicator that indicates to the user that the stylus 204 has been placed down for a certain minimum amount of time (step 319), as will be discussed below in more detail. Note that step 319 may be in a variety of places within the flowchart of Fig. 3 other than the one shown, such as between steps 305 and 306. Steps 301-303, 305-308, and 313, thus simulate a delayed right click of a mouse.

To simulate a right click drag of a mouse, if the stylus 204 is not moved until after the timeout condition (step 305), then in response the computer 201 may generate a RightMouseButtonDown event (step 309). Upon the computer 201 detecting any subsequent movement of the stylus 204 along the display surface 202 (step 310), the computer 201 may in response generate one or more MouseMove events (step 311).

Once the stylus 204 is brought up (312), in response the computer 201 may generate a RightMouseButtonUp event (step 313). This simulates, via steps 301-303, 305, and 309-313, a right click drag of a mouse.

As a further feature of the right click simulation, a special hold event may be defined wherein the stylus 204 is held against the display screen 202 without substantial movement for at least another threshold amount of time, say about 2 seconds, or about 5 seconds or more, the computer 201 may detect such a hold event and in response act by immediately interpreting any subsequent holding of the stylus 204 as holding the right mouse button down. Thus, the computer 201 may, after 600 milliseconds of stylus holding, go on to steps 305 and 319, and where the stylus 204 is released prior to, say, 5 seconds, then steps 306-308 and 313 may be implemented. However, if instead the stylus 204 is further held at least 5 seconds, then the special subroutine may, in response, immediately generate a RightMouseButtonDown event. Then, when the stylus is eventually released, a RightMouseButtonUp event would be generated by the special subroutine. If the stylus 204 is moved along the display surface 202, then the computer 201 may immediately in response generate one or more MouseMove events. When the stylus 204 is eventually removed from the display surface 202, all pending RightMouse events that would normally have been generated in accordance with the method shown in Fig. 3 would be canceled.

In the above and remaining discussion, it is assumed that a Microsoft WINDOWS operating system is being used with the computer 201. However, any operating system, especially a GUI operating system, may be used in accordance with the present invention. Further, although the above embodiments are the preferred methods for simulating mouse gestures using a stylus, there are many variations that are encompassed by the present invention. For instance, the stylus gestures required for simulation a left click and a right click might be reversed, or a user may not need to always keep the stylus 204 down on the display surface 202 during the entire time that a drag is executed, such that a first touch/touch-and-hold of the stylus 204 at a first location on the display surface 202, and a second touch/touch-and-hold at a second location, may thereby simulate a drag from the first location to the second location.

As discussed in the copending patent applications incorporated by reference herein, the computer 201 according to embodiments of the present invention may have the ability to select ink when in pen mode. A user can press and hold, then drag over ink, text, or other objects, within a document to select those objects. A press and hold may switch into selection mode. Because the user is using the stylus 204 or other similar user input device, there are different considerations as compared with when a mouse is used. For instance, while the user of a mouse may hear and feel the click of a mouse button to ensure that a click has been performed, the user of the stylus 204 may not receive clear enough feedback that the intended gesture has been made properly. Also, it is desirable to indicate to the user that a particular requested mode has really been switched to, such as selection mode. Further, where the user must hold down the stylus 204 for at least a certain threshold amount of time to simulate, for example, a right click, feedback to the user indicating that the threshold time has passed would be useful.

Such feedback may be in the form of a feedback indicator, which may be visual (e.g., a graphical icon, a symbol, a bitmapped or other image, etc., displayed on the display surface 202 such as feedback indicator 205 in Fig. 2), auditory (e.g., a click, a beep, tone, etc. emanating from the computer 201 and/or stylus 204), and/or tactile (e.g., vibration of the computer 201 and/or stylus 204). For example, where the user selects the pen icon in the application toolbar (as described in at least one of the copending patent applications incorporated herein by reference), the user may drag the stylus 204 over a document displayed on the display surface 202 such that digital ink is generated on the display surface 202. At this point, the user may switch to selection mode by pressing and holding the stylus 204 for at least a threshold amount of time that may be the same as the threshold amount of time or that may be different. For instance, the threshold minimum amount of time that the user must hold the stylus 204 against the display surface 202 may preferably be about 400 milliseconds, or between about 400 milliseconds and about 1 second, or more than about 1 second. Responsive to the user holding without substantial movement the stylus 204 against the display surface 202 for the minimum threshold amount of time, the computer 201 may generate a feedback indicator on the display surface 202 which may preferably be located at or near the location of the tip of the stylus 204.

In some embodiments, the feedback indicator may be animated and/or may otherwise change over time. For instance, the feedback indicator 205 is shown as a partially completed clock-like loop that may become more complete over time in, e.g., a clockwise direction. In such embodiments with animation, the feedback indicator may actually begin prior to the threshold amount of time passing, as an indication to the user that the threshold amount of time will soon pass. For instance, where a first threshold amount of time is 1 second, holding the stylus 204 against the display surface 202 for 500 milliseconds may cause an icon (a feedback indicator) in the shape of a wand tip to appear on the display surface 202 at or near the location of the stylus 204. As the stylus 204 is continued to be held, the icon may animate and begin to glow to indicate that a gesture is beginning to form. At the first threshold amount of time, the animation may have reached the point where the wand tip is fully glowing. Moreover, at the passing of the first threshold of time, the user may thereby be placed in selection mode which is indicated by the status of the feedback indicator.

As a further possibility in connection with the above example, at a second threshold amount of time of, e.g., 2 seconds, the wand tip may further animate and/or change to indicate that a right click will now be simulated when the stylus 204 is removed from the display surface 202.

Where the user removes the stylus 204 from the display surface 202 prior to the threshold of time passing, then any subsequent stylus movement may be interpreted by the computer 201 as normal stylus input. Also, in this situation any feedback indicator and/or animation thereof may be immediately canceled and removed from the display surface 202.

While exemplary systems and methods embodying the present invention are shown by way of example, it will be understood, of course, that the invention is not limited to these embodiments. Modifications may be made by those skilled in the art, particularly in light of the foregoing teachings. For example, each of the elements of the aforementioned embodiments may be utilized alone or in combination with elements of the other embodiments.

We claim:

1. A method for simulating a gesture such as a right click of a mouse using a stylus on a touch-sensitive display surface.

2. A method for simulating a right click of a mouse comprising the steps of:

touching and holding a stylus against a touch-sensitive display surface for at least a threshold amount of time; and

then releasing the stylus from the touch-sensitive display surface.

3. A method for providing a feedback indicator to a user responsive to a stylus being held down on a display surface of a computer for at least a threshold amount of time, thereby indicating a status.

[illegible] 03797.00072 1 of 3

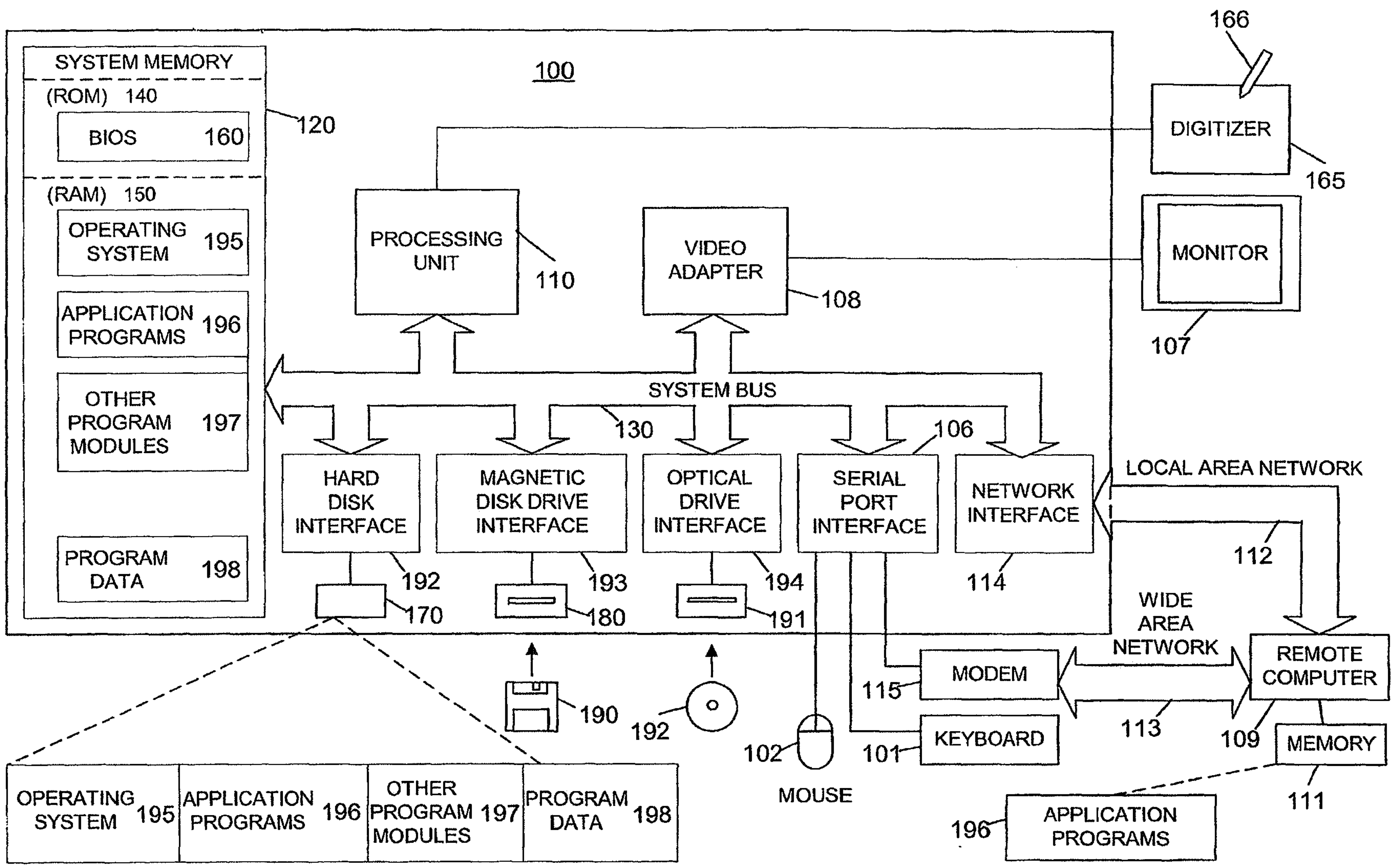


FIG. 1

03797,00072

of 3

[illegible]

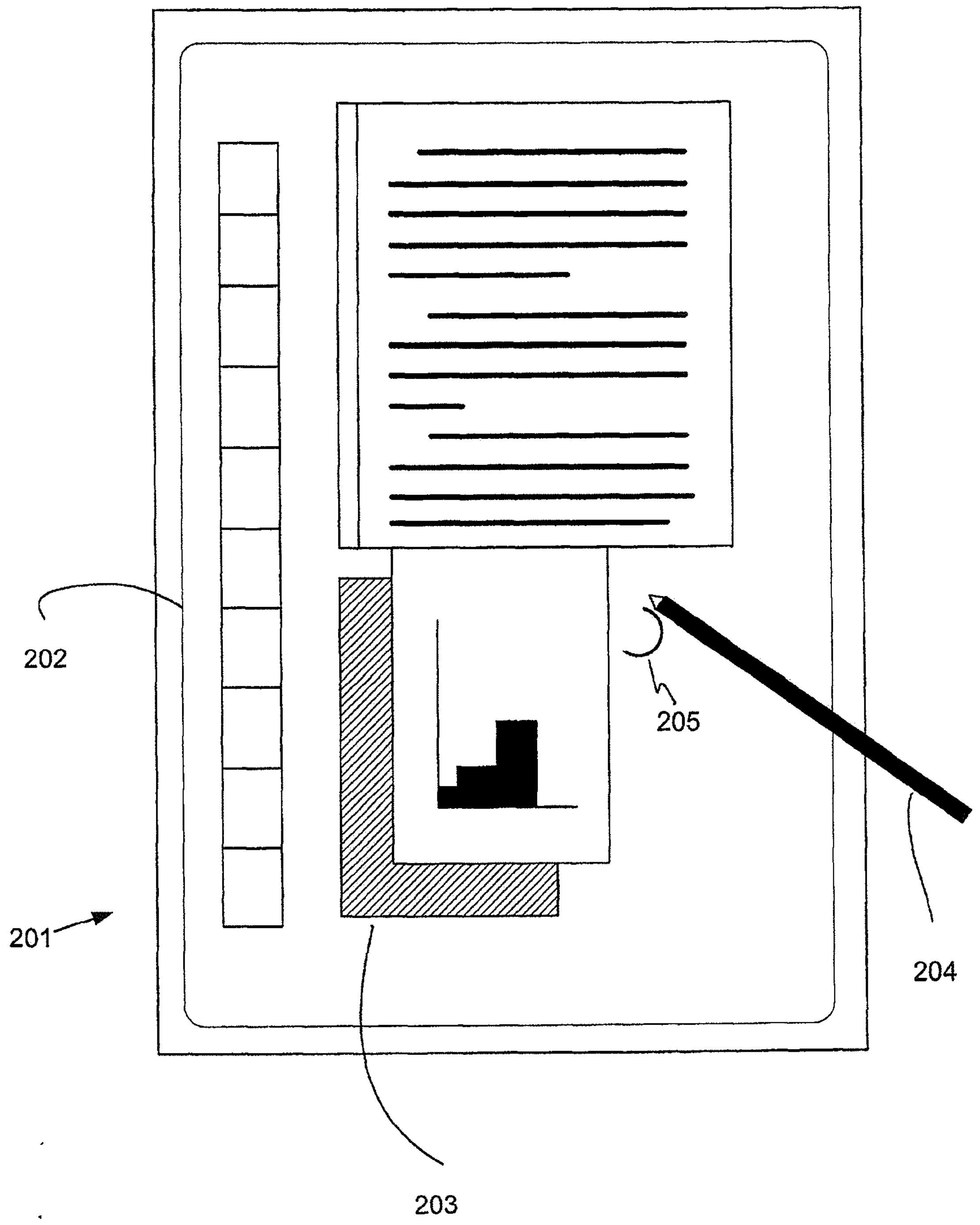


FIG. 2

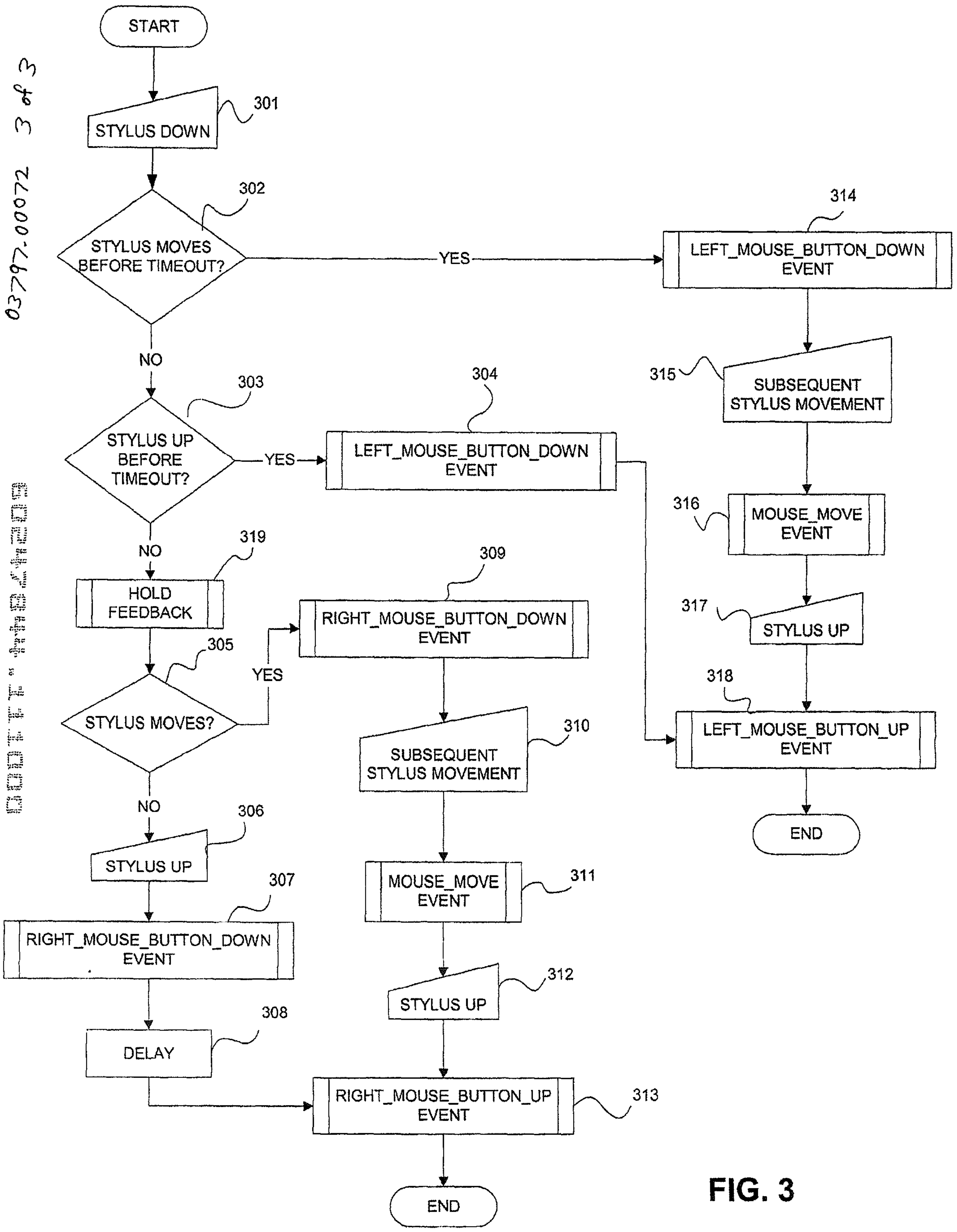

START
STYLUS DOWN
301
STYLUS MOVES BEFORE TIMEOUT?
302
YES
NO
LEFT_MOUSE_BUTTON_DOWN EVENT
314
SUBSEQUENT STYLUS MOVEMENT
315
STYLUS UP BEFORE TIMEOUT?
303
YES
LEFT_MOUSE_BUTTON_DOWN EVENT
304
NO
HOLD FEEDBACK
319
MOUSE_MOVE EVENT
316
STYLUS UP
317
RIGHT_MOUSE_BUTTON_DOWN EVENT
309
STYLUS MOVES?
305
YES
NO
LEFT_MOUSE_BUTTON_UP EVENT
318
SUBSEQUENT STYLUS MOVEMENT
310
END
STYLUS UP
306
MOUSE_MOVE EVENT
311
RIGHT_MOUSE_BUTTON_DOWN EVENT
307
STYLUS UP
312
DELAY
308
RIGHT_MOUSE_BUTTON_UP EVENT
313
END

FIG. 3

E
X
H
I
B
I
T

50

US007383460B2

(12) **United States Patent**
Sherwin, Jr. et al.

(10) **Patent No.: US 7,383,460 B2**
(45) **Date of Patent: Jun. 3, 2008**

(54) METHOD AND SYSTEM FOR CONFIGURING A TIMER

(75) Inventors: **Bruce J Sherwin, Jr.**, Woodinville, WA (US); **Eric Nelson**, Woodinville, WA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 392 days.

(21) Appl. No.: **11/089,957**

(22) Filed: **Mar. 25, 2005**

(65) **Prior Publication Data**

US 2006/0218429 A1 Sep. 28, 2006

(51) **Int. Cl.**

G06F 1/04	(2006.01)
G06F 1/12	(2006.01)
G06F 5/06	(2006.01)
G06F 11/00	(2006.01)
G06F 3/00	(2006.01)
G06F 9/44	(2006.01)
G06F 9/46	(2006.01)
G06F 13/00	(2006.01)

(52) **U.S. Cl.** 713/600; 714/55; 719/328

(58) **Field of Classification Search** 713/600; 714/55; 719/328

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,519,851	A *	5/1996	Bender et al.	710/301
6,078,747	A *	6/2000	Jewitt	717/164
2003/0204792	A1*	10/2003	Cahill et al.	714/55
2005/0022166	A1*	1/2005	Wolff et al.	717/124

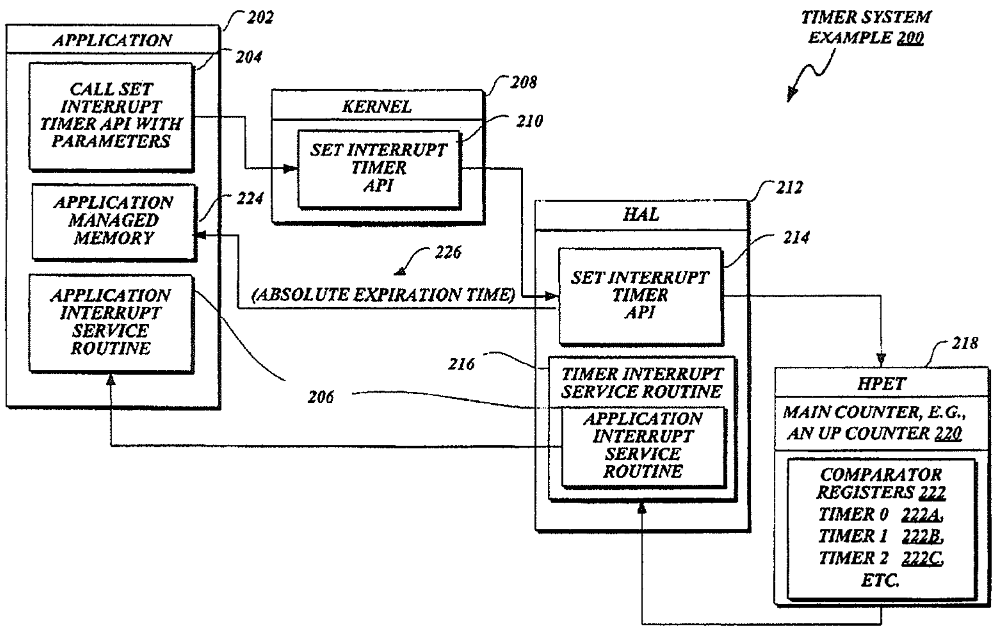
* cited by examiner

Primary Examiner—Chun Cao
Assistant Examiner—Jaweed A Abbaszadeh
(74) *Attorney, Agent, or Firm*—Christensen O'Connor Johnson Kindness PLLC

(57) **ABSTRACT**

The present invention facilitates access to timers in a computing device. In particular, a timer system facilitates configuring a hardware interrupt timer in a computing device, the timer being guaranteed to expire at a specific time in a non-real-time environment. A calling application passes parameters to a hardware independent application programming interface (API) to the hardware interrupt timer. The hardware independent API validates the parameters and relays them to a hardware dependent API. The hardware dependent API establishes a connection with the timer in accordance with the validated parameters, and executes a service routine associated with the application upon expiration of the timer.

15 Claims, 7 Drawing Sheets


TIMER SYSTEM EXAMPLE 200
202
APPLICATION
204
CALL SET INTERRUPT TIMER API WITH PARAMETERS
APPLICATION MANAGED MEMORY
224
APPLICATION INTERRUPT SERVICE ROUTINE
208
KERNEL
210
SET INTERRUPT TIMER API
226
(ABSOLUTE EXPIRATION TIME)
212
HAL
214
SET INTERRUPT TIMER API
216
TIMER INTERRUPT SERVICE ROUTINE
APPLICATION INTERRUPT SERVICE ROUTINE
206
218
HPET
MAIN COUNTER, E.G., AN UP COUNTER 220
COMPARATOR REGISTERS 222
TIMER 0 222A
TIMER 1 222B
TIMER 2 222C
ETC.

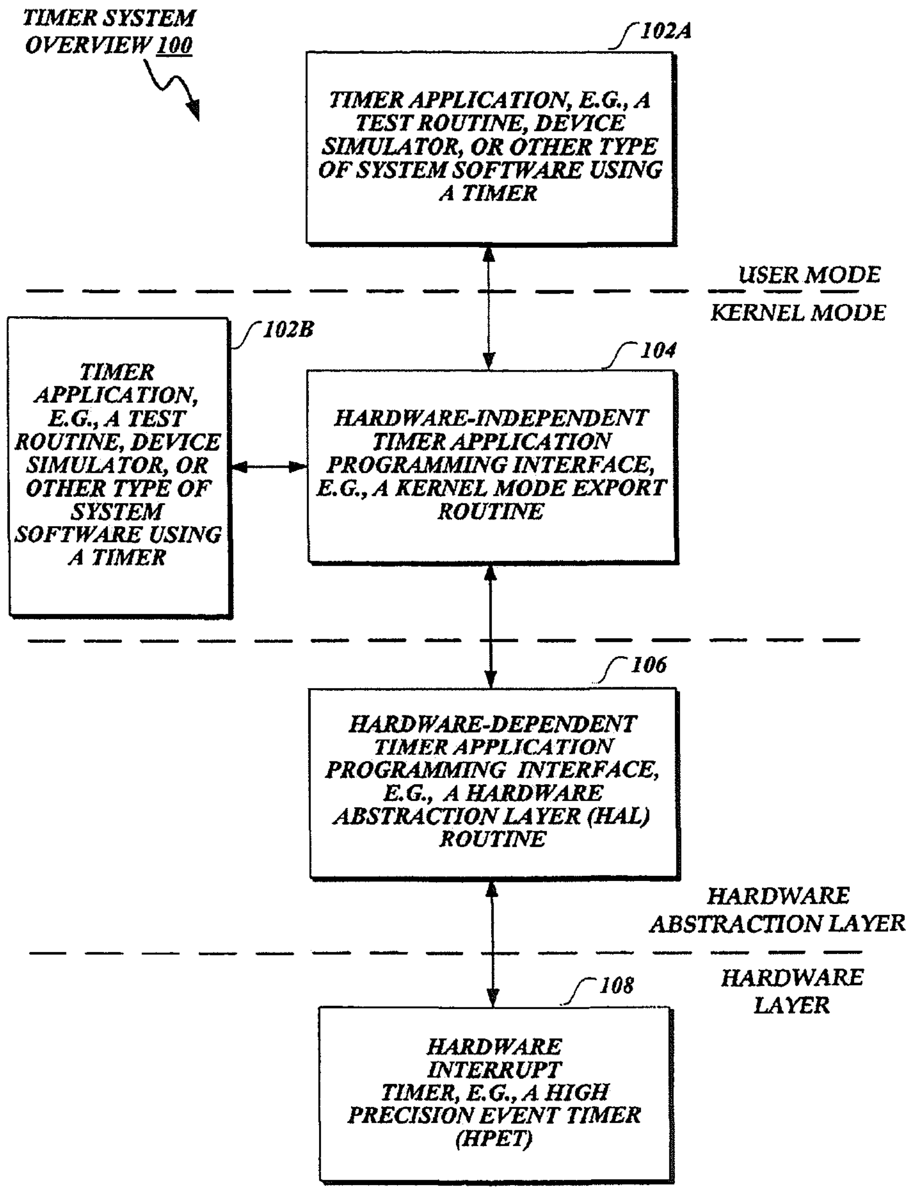

TIMER SYSTEM OVERVIEW 100
102A
TIMER APPLICATION, E.G., A TEST ROUTINE, DEVICE SIMULATOR, OR OTHER TYPE OF SYSTEM SOFTWARE USING A TIMER
USER MODE
KERNEL MODE
102B
TIMER APPLICATION, E.G., A TEST ROUTINE, DEVICE SIMULATOR, OR OTHER TYPE OF SYSTEM SOFTWARE USING A TIMER
104
HARDWARE-INDEPENDENT TIMER APPLICATION PROGRAMMING INTERFACE, E.G., A KERNEL MODE EXPORT ROUTINE
106
HARDWARE-DEPENDENT TIMER APPLICATION PROGRAMMING INTERFACE, E.G., A HARDWARE ABSTRACTION LAYER (HAL) ROUTINE
HARDWARE ABSTRACTION LAYER
HARDWARE LAYER
108
HARDWARE INTERRUPT TIMER, E.G., A HIGH PRECISION EVENT TIMER (HPET)

Fig. 1.

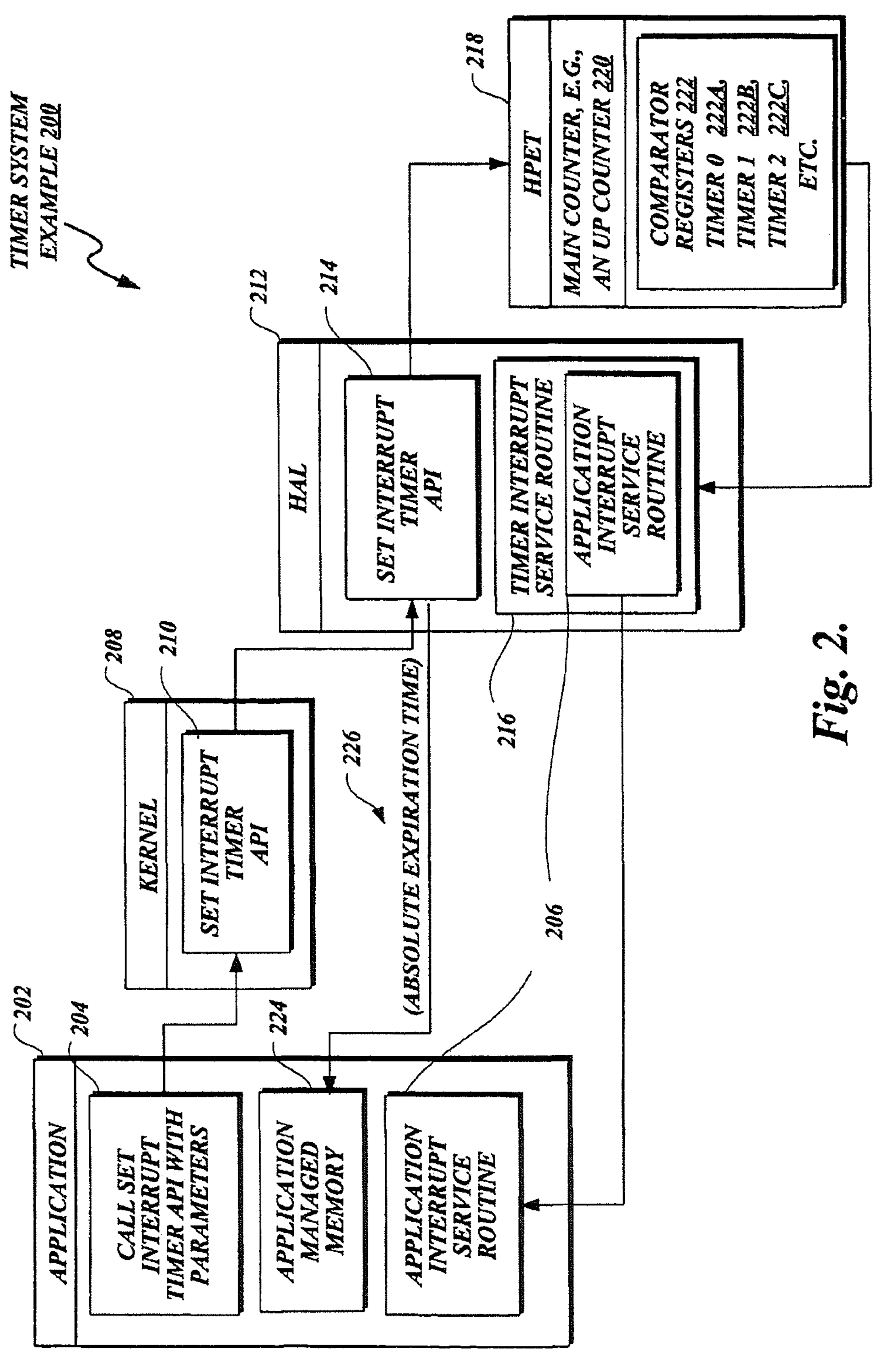
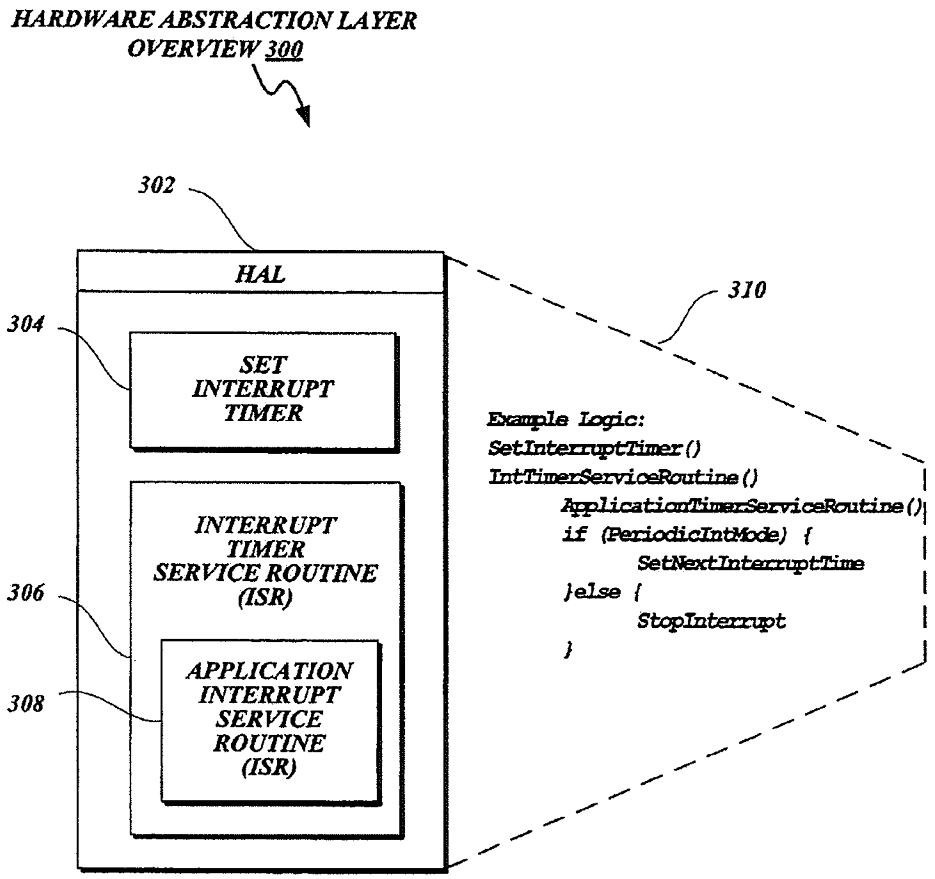

TIMER SYSTEM EXAMPLE 200
APPLICATION
202
204
CALL SET INTERRUPT TIMER API WITH PARAMETERS
224
APPLICATION MANAGED MEMORY
APPLICATION INTERRUPT SERVICE ROUTINE
206
KERNEL
208
210
SET INTERRUPT TIMER API
226
(ABSOLUTE EXPIRATION TIME)
HAL
212
214
SET INTERRUPT TIMER API
TIMER INTERRUPT SERVICE ROUTINE
APPLICATION INTERRUPT SERVICE ROUTINE
216
HPET
218
MAIN COUNTER, E.G., AN UP COUNTER 220
COMPARATOR REGISTERS 222
TIMER 0 222A,
TIMER 1 222B,
TIMER 2 222C,
ETC.

Fig. 2.


HARDWARE ABSTRACTION LAYER OVERVIEW 300
302
HAL
304
SET INTERRUPT TIMER
INTERRUPT TIMER SERVICE ROUTINE (ISR)
306
APPLICATION INTERRUPT SERVICE ROUTINE (ISR)
308
310

```
Example Logic:
SetInterruptTimer()
IntTimerServiceRoutine()
        ApplicationTimerServiceRoutine()
        if (PeriodicIntMode) {
                SetNextInterruptTime
        }else {
                StopInterrupt
        }
```

Fig. 3.

MOTM_24063_II_ 0014893

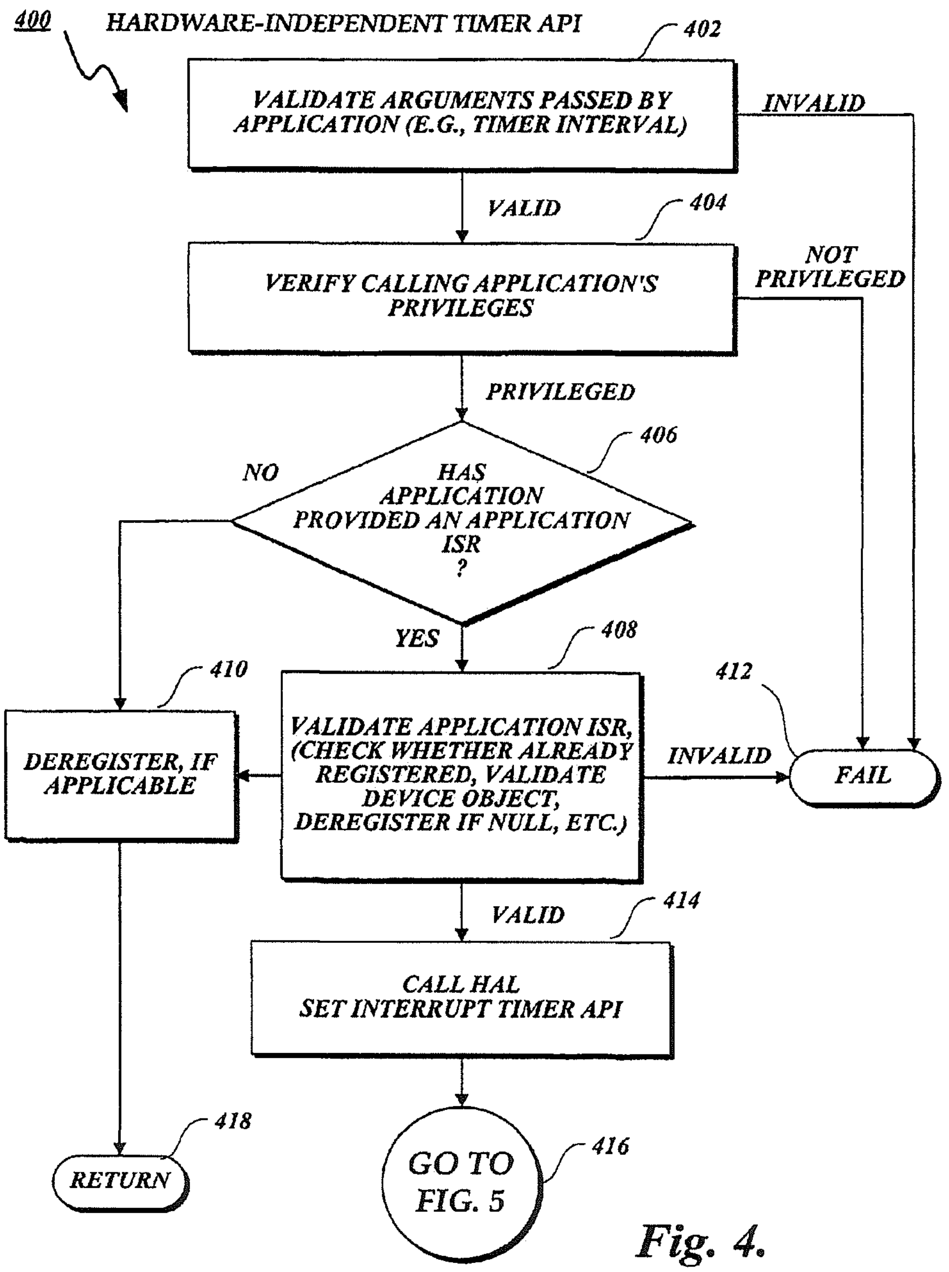

400
HARDWARE-INDEPENDENT TIMER API
402
VALIDATE ARGUMENTS PASSED BY APPLICATION (E.G., TIMER INTERVAL)
INVALID
VALID
404
VERIFY CALLING APPLICATION'S PRIVILEGES
NOT PRIVILEGED
PRIVILEGED
406
HAS APPLICATION PROVIDED AN APPLICATION ISR ?
NO
YES
408
VALIDATE APPLICATION ISR, (CHECK WHETHER ALREADY REGISTERED, VALIDATE DEVICE OBJECT, DEREGISTER IF NULL, ETC.)
410
DEREGISTER, IF APPLICABLE
412
INVALID
FAIL
VALID
414
CALL HAL SET INTERRUPT TIMER API
418
RETURN
416
GO TO FIG. 5

Fig. 4.

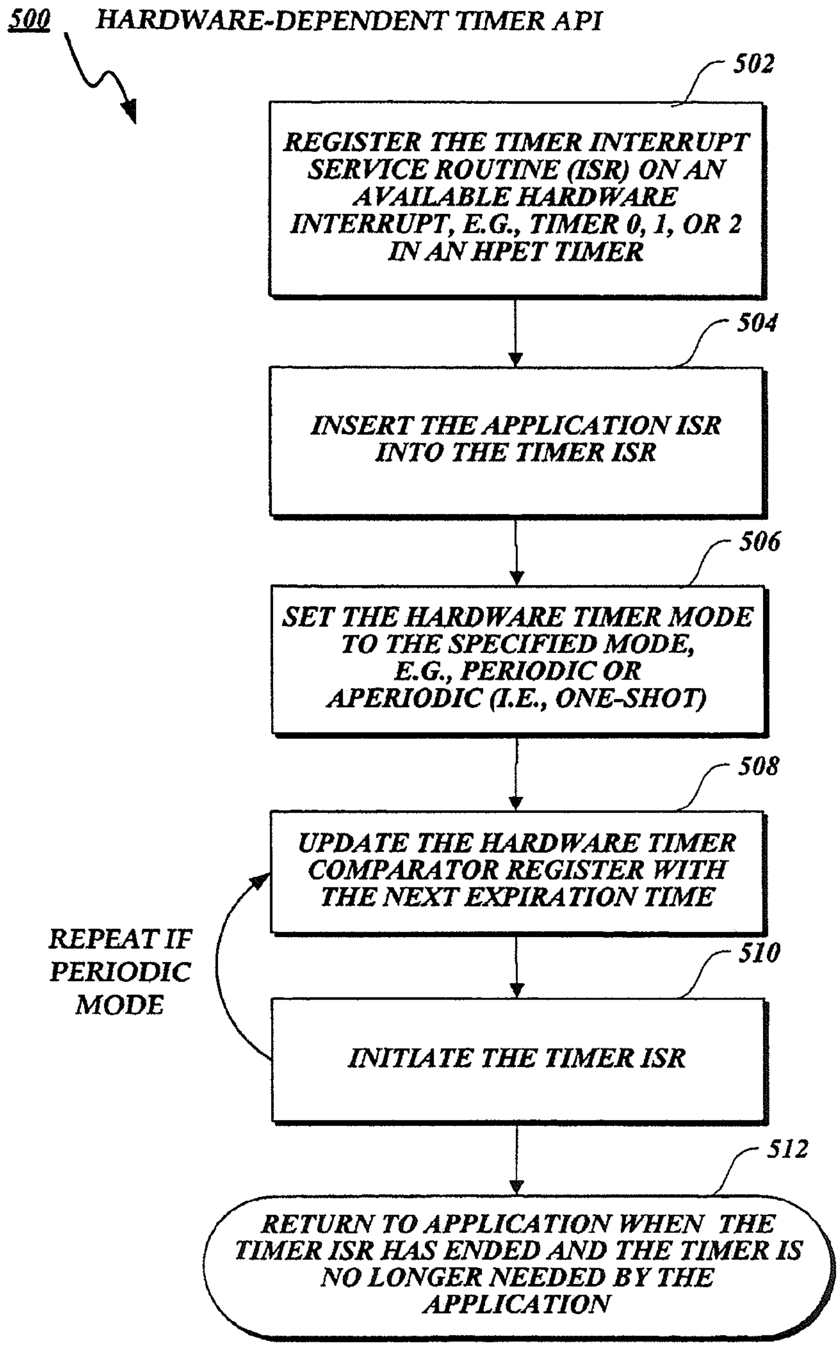

500 HARDWARE-DEPENDENT TIMER API
502
REGISTER THE TIMER INTERRUPT SERVICE ROUTINE (ISR) ON AN AVAILABLE HARDWARE INTERRUPT, E.G., TIMER 0, 1, OR 2 IN AN HPET TIMER
504
INSERT THE APPLICATION ISR INTO THE TIMER ISR
506
SET THE HARDWARE TIMER MODE TO THE SPECIFIED MODE, E.G., PERIODIC OR APERIODIC (I.E., ONE-SHOT)
508
UPDATE THE HARDWARE TIMER COMPARATOR REGISTER WITH THE NEXT EXPIRATION TIME
REPEAT IF PERIODIC MODE
510
INITIATE THE TIMER ISR
512
RETURN TO APPLICATION WHEN THE TIMER ISR HAS ENDED AND THE TIMER IS NO LONGER NEEDED BY THE APPLICATION

Fig. 5.

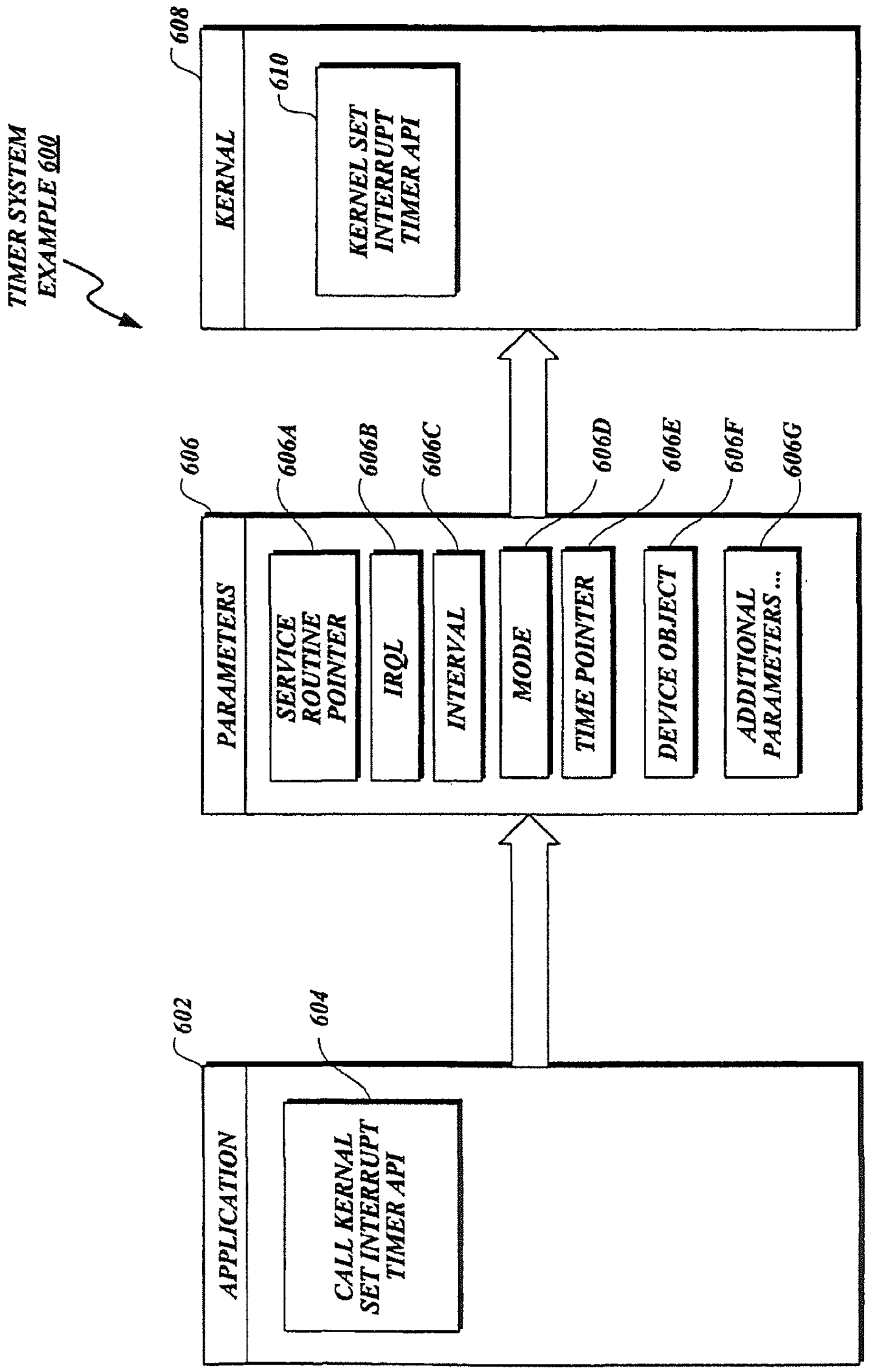

TIMER SYSTEM EXAMPLE 600
608
KERNAL
610
KERNEL SET INTERRUPT TIMER API
606
PARAMETERS
606A
SERVICE ROUTINE POINTER
606B
IRQL
606C
INTERVAL
606D
MODE
606E
TIME POINTER
606F
DEVICE OBJECT
606G
ADDITIONAL PARAMETERS...
602
APPLICATION
604
CALL KERNAL SET INTERRUPT TIMER API

Fig. 6A.

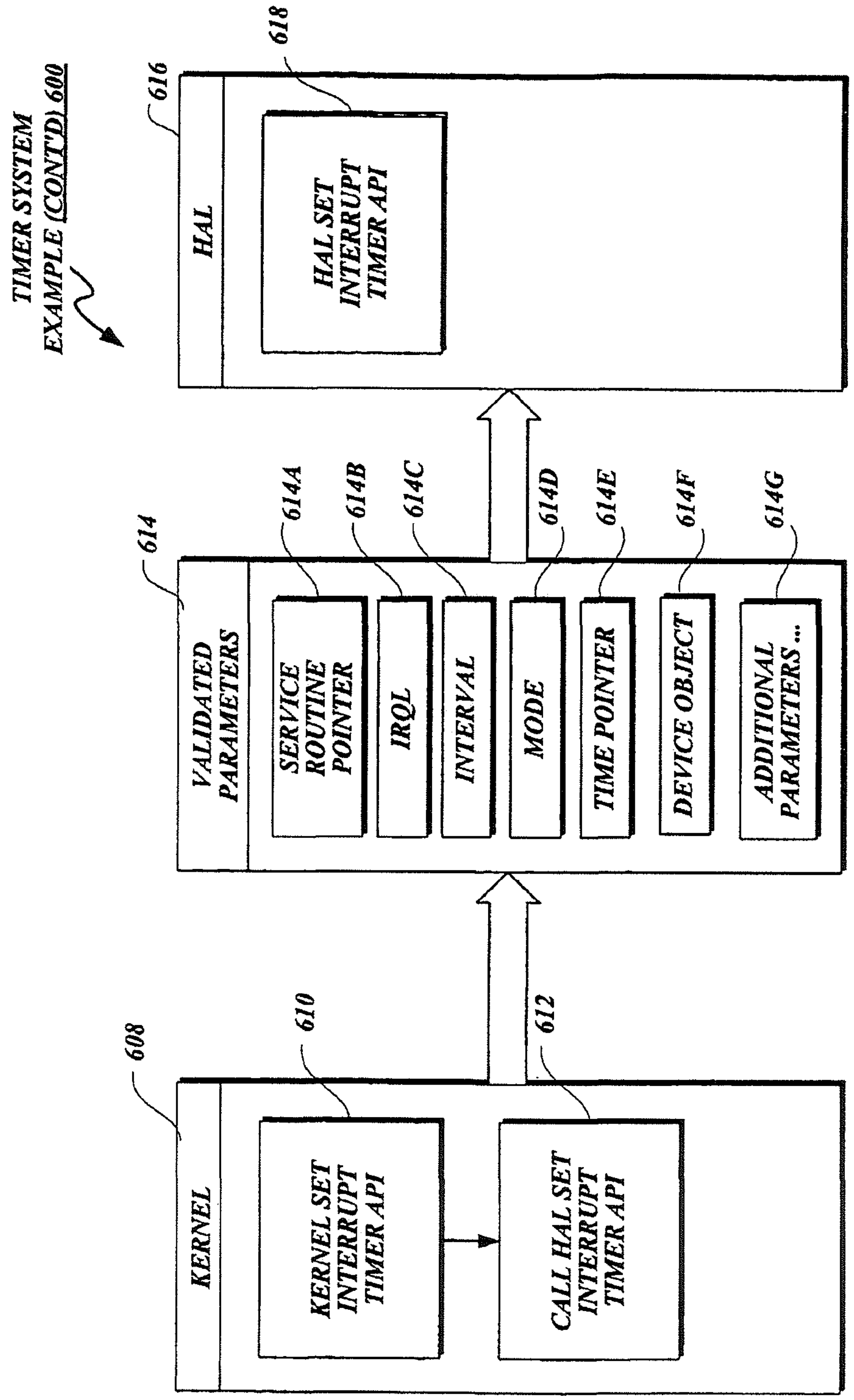

TIMER SYSTEM EXAMPLE (CONT'D) 600
616
HAL
618
HAL SET INTERRUPT TIMER API
614
VALIDATED PARAMETERS
614A
SERVICE ROUTINE POINTER
614B
IRQL
614C
INTERVAL
614D
MODE
614E
TIME POINTER
614F
DEVICE OBJECT
614G
ADDITIONAL PARAMETERS...
608
KERNEL
610
KERNEL SET INTERRUPT TIMER API
612
CALL HAL SET INTERRUPT TIMER API

Fig. 6B.

METHOD AND SYSTEM FOR CONFIGURING A TIMER

TECHNICAL FIELD

In general, the present invention relates to timers in a computing device and, more particularly, to configuring timers in a computing device.

BACKGROUND

Operating systems, such as Microsoft's Windows NT, typically provide the ability to program a timer. However, such timers are not guaranteed to expire at a programmed time; rather, they are only guaranteed to not expire before the programmed time. The lack of specificity of the time at which the timer expires makes the timer unsuitable for many applications. For example, certain test scenarios, performance and/or power consumption algorithms require a timer that is guaranteed to expire at a certain time, even when used in a non-real-time environment, such as the environment provided by Windows NT.

Timers that are guaranteed to expire at a certain time are typically hardware timers. For example, personal computer hardware provided by Intel Corporation supports a minimum of three such hardware timers, referred to as High Precision Event Timers ("HPET").

In Windows NT, access to computer hardware is controlled by the Windows NT hardware abstraction layer ("HAL"). Among the advantages of a HAL is that a single device driver can use standard HAL routines to support a device on many different hardware platforms, making device driver development much easier, and allowing different hardware configurations accessible in a similar manner. However, because the HAL operates at a level between the hardware and the Windows NT executive services, a disadvantage of the HAL is that applications and device drivers are unaware of hardware-dependent details, such as I/O interfaces and interrupt controllers, including the HPET timers. Applications and device drivers are no longer allowed to deal with hardware directly and must make calls to HAL routines to determine hardware-specific information. As a result, access to HPET timers that have the necessary specificity (i.e., access to timers that are guaranteed to expire at a certain time) for use in certain test scenarios, performance and/or power consumption algorithms is either not permitted, or is difficult at best.

SUMMARY

The foregoing problems with the prior state of the art are overcome by the principles of the present invention, which is directed toward methods, systems, computer program products, and data structures for facilitating access to timers in a computing device. The present invention is further directed toward methods, systems, computer program products, and data structures for configuring a timer in a computing device, the timer being guaranteed to expire at a specific time in a non-real-time environment.

According to one aspect of the invention, a method is provided for configuring timer hardware that is guaranteed to expire at a specified time in a non-real time environment. Upon expiration, the method may return control to the calling application in the non-real-time environment, including providing a generic software callback to the calling application. Upon expiration, the method may further initiate the timer's interrupt service routine (ISR), which may

be optionally modified to initiate an application ISR specified by the calling application.

According to another aspect of the invention, configuring timer hardware that is guaranteed to expire at a specified time may include configuring timer hardware to expire in an aperiodic or periodic mode. When configuring timer hardware to expire in the aperiodic mode, the timer is guaranteed to expire once at a specified time. In the periodic mode, the timer is guaranteed to expire more than once, at a specified interval. In the periodic mode, the timer may be further guaranteed to expire more than once at a specified interval commencing at a specified time.

According to still another aspect of the invention, the method for configuring timer hardware may include an application programming interface (API) comprising a hardware-independent API and a hardware-dependent API. A timer application may call the hardware-independent API specifying, among other parameters, at least one of an interval, an interrupt request level (IRQL), and an application ISR. Among the functions provided by the hardware-independent API is validation of the specified parameters, verification of the calling application's privileges, and registration or deregistration of the application ISR.

According to yet another aspect of the invention, the hardware-independent API may, in turn, call the hardware-dependent API, passing, among other parameters, the specified time, interval, IRQL, and application ISR. Among the functions provided by the hardware-dependent API is the registration of a timer ISR on an available hardware interrupt, wrapping the application ISR into the timer ISR, setting the timer mode of operation to periodic when an interval is specified, updating the hardware timer's comparator register with the specified time and/or interval, initiating the timer ISR upon expiration of the timer, initiating the application ISR from the timer ISR, and returning control to the calling application.

In accordance with yet other aspects of the present invention, a computer-accessible medium for facilitating access to timers in a computing device is provided, including a medium for storing data structures and computer-executable components for establishing a connection between an application and a hardware timer, specifying and validating parameters for the operation of the hardware timer, initiating the timer on behalf of the application, and returning control to the application when the timer has ended. The data structures define the hardware timer parameters, and other timer data in a manner that is generally consistent with the above-described systems and methods. Likewise, the computer-executable components, including the hardware-independent and hardware-dependent APIs to the hardware timers, are capable of performing actions generally consistent with the above-described systems and methods.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same become better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIG. 1 is a block diagram overview of an exemplary timer system and one suitable operating environment in which access to a timer may be facilitated in accordance with the present invention;

FIG. 2 is a block diagram illustrating in further detail an arrangement of certain components of the timer system illustrated in FIG. 1 for implementing an embodiment of the present invention;

FIG. 3 is a block diagram illustrating in further detail certain aspects of a hardware-dependent interface formed in accordance with an embodiment of the present invention;

FIG. 4 is a flow diagram illustrating certain aspects of a hardware-independent application programming interface for implementing an embodiment of the present invention;

FIG. 5 is a flow diagram illustrating certain aspects of a hardware-dependent application programming interface for implementing an embodiment of the present invention;

FIG. 6A is a block diagram illustrating certain aspects of a hardware-independent application programming interface for implementing an embodiment of the present invention; and

FIG. 6B is a block diagram illustrating certain aspects of a hardware-dependent application programming interface for implementing an embodiment of the present invention.

DETAILED DESCRIPTION

To successfully run certain types of test routines, device simulators, or other types of application or system software on a computing device, the software must have access to a timer that is guaranteed to expire at a specified time and/or at a specified interval. However, the functionality of a timer that is guaranteed to expire at a specified time and/or specified interval may not always be accessible to the application or system software that needs it. For example, in the Microsoft Windows NT environment, the hardware application layer (HAL) does not permit application and system software to directly access the hardware-specific information needed to use a hardware timer, such as the High Precision Event Timer (HPET) timer that is provided with Intel computer hardware. Unfortunately, hardware timers are generally the only timers that are guaranteed to expire at a specified time. To address this and other problems with accessing timers, a computing system suitable for implementing a method for facilitating access to a timer guaranteed to expire at a specified time or interval in accordance with embodiments of the present invention is described in detail in the following discussion.

As already noted, guaranteed timers are generally hardware timers. Since the specific details of programming a hardware timer may vary from one hardware platform to the next, certain aspects of the method for facilitating access to a timer in accordance with embodiments of the present invention are typically performed in a hardware-independent component, and certain other aspects of the method are typically performed in a hardware-dependent component.

FIG. 1 is a block diagram overview of an exemplary timer system 100 and one suitable operating environment in which access to a timer may be facilitated in accordance with the present invention. As shown, the timer system 100 may include a timer application 102A and/or 102B, residing in either the user mode or the kernel mode, needing access to a timer that can be guaranteed to expire at a specified time and/or at a specified interval. Examples of such timer applications include, but are not limited to, various system timer test routines, device simulators, processor power management testing routines, scheduling algorithms, and kernel performance algorithms.

The timer applications 102A, 102B, establish a connection with a timer 108 that is guaranteed to expire at a specified time and/or a specified interval through a hardware-independent timer application programming interface (API) 104. The hardware-independent API 104 is configured to receive a number of parameters from the timer applications 102A, 102B, such as a pointer to an application interrupt service routine (ISR) that should be initiated upon expiration of the timer, the mode of the timer, i.e., periodic or aperiodic, the interval at which the timer should expire, and so forth.

During operation of the timer 100, a hardware-independent timer API 104 initiates the establishment of a connection between the calling timer application 102A, 102B, and the timer 108 by, among other actions, validating parameters received from the calling timer application, verifying whether the application is authorized to set an interrupt timer using the API 104, and if so, relaying the validated parameters to a hardware-dependent API 106. The hardware-dependent API 106 completes the process of establishing the connection between the calling timer application 102A, 102B, and the hardware timer 108 by, among other actions, setting the hardware timer in accordance with the validated parameters received from the calling timer application.

In one embodiment, in the context of an operating system such as the Microsoft Windows NT operating system, the hardware-independent timer API 104 may be implemented in a kernel mode export driver. A kernel mode export driver may be advantageously used to implement certain hardware-independent aspects of a method for facilitating access to a timer in accordance with an embodiment of the invention. In one embodiment, the kernel mode export driver is a kernel mode dynamic link library (DLL) capable of being loaded by other components of an operating system. The hardware-independent timer API 104 may be implemented in a routine that is part of the hardware abstraction layer, or HAL. A HAL routine has access to the various components of the timer 108 that are necessary to establish a connection with and set the timer according to the validated parameters received from the calling timer application 102A, 102B by way of the hardware-independent API 104.

In one embodiment, the timer 108 may be a high precision event timer (HPET) such as that provided in Intel computer hardware. Establishing a connection with the timer 108 and setting the timer in accordance with an embodiment of the invention will be described in further detail below.

FIG. 2 is a block diagram illustrating in further detail an arrangement of certain components of the timer system illustrated in FIG. 1 for implementing an embodiment of the present invention. A calling application 202 contains a process to call a kernel mode routine 208 having a set interrupt timer API 210 capable of receiving one or more parameters. In one embodiment, the calling application 202 may pass parameters 204 that specify the various timer settings, such as the mode, the interval, the interrupt request level (IRQL) at which the kernel mode routine 208 is to run, and an interrupt service routine (ISR) 206 (or a name or pointer to an application ISR) that should be run upon expiration of the timer with which the calling application 202 ultimately establishes a connection. In one embodiment, the parameters 204 may also include a pointer or other reference to an area of application managed memory 224 in which will be written the actual time that the timer with which the calling application 202 ultimately establishes a connection will expire, also referred to as the absolute expiration time 226.

In one embodiment, the set interrupt timer API 210 validates the parameters 204 and verifies whether the calling application 202 is authorized to set a timer using the API 210. The API 210 then passes the validated parameters to a

HAL routine **212**, also having a set interrupt timer API **214** corresponding to the API **210**.

In one embodiment, the HAL routine **212** includes its own timer interrupt service routine (ISR) **216**, or a pointer to a timer ISR, that should be run upon expiration of the timer with which the calling application **102A**, **102B**, or **202** ultimately establishes a connection. In one embodiment, the timer ISR **216** may be modified to embed the application ISR **206**, also referred to as wrapping the application ISR **206** in the timer ISR **216**, or otherwise run the application ISR **206** upon the expiration of the timer.

In one embodiment, the HAL set interrupt timer API **214** accesses an available HPET timer **218** and sets the timer by, among other actions, writing an expiration time at which the timer **218** should expire in a corresponding comparator register **222**, where the corresponding comparator register includes, for example, comparator registers corresponding to Timer **0**, **222A**, Timer **1**, **222B**, or Timer **2**, **222C**, each located at different offsets in the available HPET timer. In one embodiment, the HAL set interrupt timer API **214** also writes the expiration time at which the timer **218** should expire in an area of application managed memory **224**, the location of which may be conveyed to the HAL set interrupt timer API **214** in one of the validated parameters. The operation of the HAL set interrupt timer API **214** will be discussed in further detail with reference to FIG. **3**, below.

Once the timer **108** has been set, the connection between the calling application **102A**, **102B**, or **202** and the timer **108**, or **218** is established. The general behavior of the timer **108** is to generate an interrupt at the specified IRQL when the timer expires in accordance with the mode and interval specified in the parameters. For example, when using an HPET timer **218**, the timer expires when the main counter **220** of the timer reaches the value written to the comparator register **222A**, **222B**, or **222C**, where the value written to the comparator register is derived from the mode and interval specified in the parameters passed by the calling application **202**. The general behavior and operation of HPET timers are known in the art and are set forth in Intel's *Intel Architecture/Personal Computer (IA/PC) HPET (High Precision Event Timers) Specification, Revision* 1.0*a*, October 2004. Accordingly, details regarding HPET timers will not be further discussed except as the operation and general behavior of the HPET timer pertains to the described embodiments of the present invention.

FIG. **3** is a block diagram illustrating in further detail a hardware abstraction layer (HAL) overview **300** of a hardware-dependent interface formed in accordance with an embodiment of the present invention. In one embodiment, the hardware-dependent interface to a timer may include a HAL routine **302** corresponding to HAL routine **212** in FIG. **2**.

The HAL routine **302** includes a process **304** to initially set a hardware interrupt timer **108** in accordance with the validated parameters received from the calling application **102A**, **102B**, or **202** (in FIGS. **1**, **2**). As already noted, when using an HPET timer **218**, the set interrupt timer process **304** may include, among other actions, writing an expiration time at which the HPET timer **218** should expire in the corresponding timer's comparator register **222**, where the corresponding timer is one of the timers, e.g., Timer **0 222A**, Timer **1 222B**, or Timer **2**, **222C**, with which the calling application **202** has established a connection. The process **304** may include other actions to initially set the hardware interrupt timer **108** depending on the particular type of hardware interrupt timer that is being used.

In one embodiment, the expiration time at which the hardware interrupt timer **208** should expire is derived from at least one of the validated parameters received by the HAL routine **302**, including the parameters that specified the interval and mode with which to set the interrupt timer using process **304**. For example, in a typical embodiment, the expiration time is the current clock time plus the amount of time represented by the interrupt interval specified in the parameters passed by the calling application **102A**, **102B**.

In one embodiment, the HAL routine **302** also writes the expiration time to an application-managed area of memory **224** (FIG. 2) in the computing device, the pointer to which may optionally be specified in the validated parameters received by the HAL routine **302**. Writing the expiration time to this area of memory enables the application to have access to the actual time at which the hardware timer **108** is set to expire, referred to as the absolute expiration time **226** (FIG. 2).

The HAL routine **302** may also include a process to service the interrupts generated by the timer **108**, i.e., a timer ISR **306**. The timer ISR process **306** contains the logic to set the next expiration time, or to stop the interrupts, depending on the mode specified in the validated parameters received by the HAL routine **302**. For example, when the mode is periodic, the derivation and writing of the expiration time is repeated indefinitely, i.e., until the calling application terminates the connection to the timer; otherwise the derivation and writing of the expiration time is carried out once only, i.e., the timer is a one-shot timer.

In one embodiment, the HAL routine **302** may modify the timer ISR **306** to include an additional process to further service the interrupts generated by the timer **108**. For example, with reference to FIG. **2**, the additional process may be an application ISR **206** that is typically supplied by the calling application **202** as specified in the validated parameters. Depending on the particular embodiment, the calling application **102A**, **102B**, **202** may supply an application ISR **206** indirectly, by name or by pointer, or directly as executable code.

In an actual embodiment, the HAL routine **302** may be implemented in the example code **310** illustrated in FIG. **3**, and set forth below in Table 1.

TABLE 1

HAL Set Interrupt Timer and Service Interrupt Timer Example Logic

```
SetInterruptTimer( )
IntTimerServiceRoutine( )
    ApplicationTimerServiceRoutine( )
    if (PeriodicIntMode) {
        SetNextInterruptTime
    }else {
        StopInterrupt
    }
```

FIG. **4** is a flow diagram illustrating certain aspects of a hardware-independent timer API **104** (in FIG. **1**) for implementing an embodiment of the present invention. In particular, the hardware-independent logic **400** embodied in the hardware-independent timer API **104** will be described with reference to the foregoing descriptions of the various components of a timer system overview **100** referenced in FIG. **1** and the timer system example **200** referenced in FIG. **2**, including, among others, the calling application **102A**, **102B**, **202** and parameters passed by the application, e.g., the parameters **204** and the application ISR **206**, the hardware-dependent timer API **106** and example component **212**, including the timer API **214** and the timer ISR **216**, and the hardware interrupt timer **108** such as the HPET timer **218**.

The hardware-independent logic **400** begins at process block **402** with validating the arguments, e.g., parameters **204** passed by the calling timer application **202**. In a typical embodiment, validating the parameters includes, among other actions, insuring that the mode has been specified as periodic or aperiodic, making sure that the calling timer application **102A**, **102B**, **202** has expressed the specified time interval in appropriate system time units, e.g., 100-nanosecond intervals, and that the specified interval is of sufficient duration to use with the available hardware timer **108**. For example, when setting an HPET timer **218**, as a practical matter, the interval specified in parameters **204** should be sufficiently long enough to allow the derived expiration time to be written to a comparator register **222** before the main counter **220** actually reaches that time. Otherwise, the hardware-dependent timer API **106**, e.g., the HAL API **214**, will not be able to set the HPET timer **218** properly.

Additional cross-validation of two or more arguments may be performed, such as that described below in blocks **406** and **408** with reference to the application ISR and an associated device object, both of which may also be specified in parameters **204**. If any of the parameters **204** are invalid, either alone or in combination, the hardware-independent timer API **104** branches to termination fail process **412** to return to the calling application with an appropriate error message.

The hardware-independent logic **400** continues at process block **404** with verifying the calling application's privileges, i.e., making sure that the calling application **102A**, **102B**, **202** is operating at a system privilege level that allows access to the various components of the timer **100**, such as the APIs **104** and **106**, and the hardware interrupt timer **108**. If the application If the calling application **102A**, **102B**, **202** is not authorized, the hardware-independent timer API **104** branches to termination fail process **412** to return to the calling application with an appropriate error message.

The hardware-independent logic **400** continues at decision block **406** to determine whether the calling application **102A**, **102B**, **202**, has provided an application ISR, such as application ISR **206** (FIG. 2). The application ISR **206** may be provided in any number of ways, but is typically provided by passing a pointer to an application ISR **206** in the parameters **204**. In one embodiment, when an application ISR **206** has not been provided, the hardware-independent logic **400** may optionally perform a process **410** to deregister any previously specified application ISR **206**, after which control is returned at termination block **418** to the calling application **102A**, **102B**, **202**. Otherwise, if an application ISR **206** has been provided, the hardware-independent logic **400** may optionally perform a process **408** to validate the application ISR **206**, including cross-validating the application ISR **206** with a device object, also specified in parameters **204**. For example, if the pointer to the application ISR **206** is null, and the device object points to the same location used to register the current application ISR **206**, the hardware-independent logic **400** may optionally perform a process **410** to deregister the current application ISR. If the device object points to a different location in the calling application's memory than that used to register the current application ISR **206**, the application ISR **206** that was provided cannot be validated, and the hardware-independent timer API **104** branches to termination fail process **412** to return to the calling application with an appropriate error message.

The hardware-independent logic **400** continues at process block **414**, where the timer system **100** transfers control to the hardware-dependent portion of the **100**, i.e., the hardware-dependent API **106**, the description of which is referenced in connector circle **416** at FIG. **5** below.

FIG. **5** is a flow diagram illustrating certain aspects of a hardware-dependent timer API **106** (in FIG. 1) for implementing an embodiment of the present invention. In particular, the hardware-dependent logic **400** embodied in the hardware-dependent timer API **106** will be described with reference to the foregoing descriptions of the various components of a timer system overview **100** referenced in FIG. **1** and the timer system example **200** referenced in FIG. **2**, including, among others, the calling application **102A**, **102B**, **202** and parameters passed by the application, e.g., the parameters **204** and the application ISR **206**, the hardware-dependent timer API **106** and example components **212**, including the timer API **214** and the timer ISR **216**, and the hardware interrupt timer **108** such as the HPET timer **218**.

The hardware-dependent logic **500** begins at process block **502** to register the timer ISR, e.g. the timer ISR **216** contained in HAL routine **212**, on an available hardware interrupt, i.e., on one of the timers, Timer **0**, **222A**, Timer **1**, **222B**, or Timer **2**, **222C** in an available HPET timer **218**. Registering the timer ISR initiates the connection between the calling application **102A**, **102B**, and the hardware interrupt timer **108**.

In one embodiment, the hardware-dependent logic **500** continues at process block **504** by inserting (also referred to as embedding or wrapping) the application ISR into the timer ISR. For example, as described with reference to FIGS. **2** and **3**, the HAL routine **302** (reference **212** in FIG. 2) set interrupt timer process **304** (reference **214** in FIG. 2) inserts the application ISR **308** (reference **206** in FIG. 2) into the timer ISR **306** (reference **216** in FIG. 2).

In one embodiment, the hardware-dependent logic **500** continues at process block **506** by setting the hardware interrupt timer **108** to an aperiodic mode or periodic mode, depending on the mode that was specified in the parameters passed by the calling application **102A**, **102B**. Setting the timer **108** to the periodic mode will cause the timer to generate an interrupt at regular time intervals, as indicated in the interval specified in the parameters, e.g. parameters **204**. Setting the timer **108** to the aperiodic mode will cause the timer **108** to generate just one interrupt, based on the interval specified in the parameters.

In one embodiment, the hardware-dependent logic **500** continues at process block **508** to update the hardware interrupt timer **108** with the actual expiration time. The actual expiration time will be determined by the interval that was specified in the parameters passed by the calling application **102A**, **102B**. As such, the interval represents a relative time at which the timer **108** will expire, and the actual time depends on the current clock time, also referred to as the current system time of the computing device in which the hardware interrupt timer resides. In a typical embodiment, at process block **508** the specified interval is added to the current system time to obtain the actual expiration time. When the timer system **100** is being implemented in a device using an HPET timer **218**, the actual expiration time is written to the comparator register **222** corresponding to the timer, Timer **0**, **222A**, Timer **1**, **222B**, or Timer **2**, **222C**, on which the timer ISR **216** was registered in process block **502**. In one embodiment, at process block **508**, the actual expiration time may be optionally written to an area of application managed memory, such as memory

224 in FIG. 2, the location of which is determined by a pointer or other information specified in the parameters passed by the calling application 102A, 102B. This enables the calling application 102A, 102B to have access to the actual expiration time of the timer with which a connection has been established.

The hardware-dependent logic 500 continues at process block 510 to initiate the timer ISR (and embedded application ISR, if any), upon expiration of the timer 108. The processes in process blocks 508 and 510 are repeated when the timer is set to periodic mode, incrementing the actual expiration time by the time in the specified interval until, at termination process 512, the application terminates the connection that was established with the timer 108, or the timer otherwise stops operating.

In one embodiment, the appropriate error messages referred to in FIGS. 4 and 5 may include, but are not limited to, an insufficient resources message indicating that another application ISR may have already registered with the API, a hardware timer not supported message indicating that suitable hardware interrupt timers are not present in the computing device, and an invalid parameter message indicating that one or more parameters 204 are invalid, or the combination of parameters is invalid.

FIGS. 6A and 6B illustrate further details of an example implementation 600 of a timer system 100. In particular, FIG. 6A is a block diagram illustrating certain aspects of a hardware-independent application programming interface for implementing an embodiment of the present invention. As shown, a calling application 602 contains a call 604 to a kernel mode routine 608 having a set interrupt timer API 610. Passed in the call 604 are one or more parameters 606, including an interrupt service routine pointer 606A that points to an application-provided ISR to be run upon expiration of the timer, an IRQL 606B that specifies the request level at which the application-provided ISR is to operate, an interval 606C, that specifies a unit of time after which the timer should expire, a mode 606D, that specifies whether the timer should operate in periodic or aperiodic mode, a time pointer 606E, that specifies the location in application-managed memory in which the actual time that the timer is set to expire is to be written, and another pointer 606F to application managed memory associated with the interrupt service routine pointer 606A. Other additional parameters 606G may be passed in the call 604 depending on the implementation. For example, in some implementations, the calling application 602 may specify a value representing the set of processors on which device interrupts can occur. As another example, in some implementations, the calling application 602 may specify yet another pointer to an area in application managed memory that represents a service context that will be supplied to the application-provided ISR when executed upon expiration of the timer.

In FIG. 6B, the kernel mode routine 608 having a set interrupt timer API 610 further includes a call 612 to a hardware application layer (HAL) routine 616 having its own set interrupt timer API 618. The HAL set interrupt timer API 618 receives the validated parameters 614 passed by the call 612 from the kernel mode routine. The validated parameters 614 correspond to the parameters 606 passed by the calling application 602, and include an interrupt service routine pointer 614A that points to an application-provided ISR to be run upon expiration of the timer, an IRQL 614B that specifies the request level at which the application-provided ISR is to operate, an interval 614C, that specifies a unit of time after which the timer should expire, a mode 614D, that specifies whether the timer should operate in periodic or aperiodic mode, a time pointer 614E, that specifies the location in application-managed memory in which the actual time that the timer is set to expire is to be written, another pointer 614F to application managed memory associated with the interrupt service routine pointer 614A, and other additional parameters 614G relayed by the call 612.

As noted earlier, there are several scenarios in which the above-described timer system 100 may be particularly useful, including processor power management testing, device simulation, system timer testing, and to quantify the relative load of interrupts currently being serviced at a particular IRQL.

For example, in the processor power management testing scenario, a processor in the relatively light C1 idle state will resume to C0 more quickly than it will from the deeper C2 idle state; likewise for C3 and C4. The deeper the idle state, the more power savings, but the higher the exit latency. Since an interrupt returns the processor to the C0 running state, it may be possible to determine which processor idle state a processor is in based on the relative latency in servicing that interrupt, i.e., the exit latency. To calculate the exit latency, a calling timer application 102A calls the set interrupt timer APIs 610, 618, to the HPET timer to specify an interval to generate an interrupt at time T1, and then queries the HPET timer's up counter at time T2 in the timer application ISR associated with that interrupt. The exit latency equals time T1 subtracted from time T2.

In the device simulation scenario, a device simulator would ordinarily need to usurp all of the capacity of one CPU to successfully simulate a device in a multi-processor system. However, using the functionality of the set interrupt timer APIs 610, 618 to perform periodic processing at a very high resolution at a nearly guaranteed rate allows the simulation to run instead on an UP processor system. An example is the simulation of USB isochronous data transfers used for streaming video and audio. A real USB 2.0 controller is equipped with a processor that polls a shared-memory structure every 125 micro-seconds to check for data that needs to be moved. A calling application simulates the polling of the shared-memory using the set interrupt timer APIs 610, 618 to generate an interrupt and poll the memory at a rate that is very close to the real controller without usurping the processor.

As already discussed, in the system timer testing scenario, testing is difficult in a system operating Microsoft Windows NT, because timers are never guaranteed to expire at a specific time, but rather are guaranteed not to expire before a specific time. Also, there are no APIs to determine which timers are in use on a given system. By using the set interrupt timer APIs 610, 618 to generate a non-shared hardware interrupt to drive an application ISR, the system timer test routines may provide a higher degree of assurance that the timer will expire at the specific time that the calling application intended. This is because hardware interrupts are generally serviced with much less latency than the software deferred procedure calls used in the kernel mode architecture. Furthermore, hardware interrupts are not directly tied to the processing load of a system, allowing more aggressive testing of hardware timers and measuring software latencies.

In the scenario of quantifying the relative load of interrupts currently being serviced at a particular IRQL, the set interrupt timer APIs 610, 618 may be used to observe trends in interrupt latency over long periods of time. Possible consumers for this information include: test tools, stress scenarios, performance calculations, scheduling algorithms, and power state transition algorithms.

The foregoing discussion has been intended to provide a brief, general description of a computing system suitable for implementing various features of the invention. Although described in the general context of a personal computer, those skilled in the art will appreciate that the invention may be practiced with many other computer system configurations. For example, the invention may be practiced with a personal computer operating in a standalone environment, or with multiprocessor systems, minicomputers, mainframe computers, and the like. In addition, those skilled in the art will recognize that the invention may be practiced on other kinds of computing devices including laptop computers, tablet computers, personal digital assistants (PDA), or any device upon which computer software or other digital content is installed.

For the sake of convenience, some of the description of the computing system suitable for implementing various features of the invention included references to the Windows NT operating system. However, those skilled in the art will recognize that those references are only illustrative and do not serve to limit the general application of the invention. For example, the invention may be practiced in the context of other operating systems such as the LINUX or UNIX operating systems.

Certain aspects of the invention have been described in terms of programs executed or accessed by an operating system in conjunction with a personal computer. However, those skilled in the art will recognize that those aspects also may be implemented in combination with various other types of program modules or data structures. Generally, program modules and data structures include routines, subroutines, programs, subprograms, methods, interfaces, processes, procedures, functions, components, schema, etc., that perform particular tasks or implement particular abstract data types.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method for configuring a timer in a computing device, the method comprising:

receiving a request from an application to set a hardware interrupt timer, the application including:

(a) a process that causes said request to be sent to a kernel mode routine having a first set interrupt timer application programming interface (API) capable of receiving said request,

(b) an application managed memory, and

(c) an application interrupt service routine to be run upon expiration of the hardware interrupt timer, said request including a parameter chosen from the group comprising:

(i) a mode in which the hardware interrupt timer is requested to operate;

(ii) an interval;

(iii) a reference to an area of the application managed memory in which a hardware-dependent process is to store a value representing an actual time at which the hardware interrupt timer has been set to expire in accordance with the validated request, and

(iv) an interrupt request level at which the application interrupt service routine should execute;

validating the request in a hardware-independent process, said hardware-independent process comprising the kernel mode routine having the first set interrupt timer API, wherein validating the request includes validating the parameter included in the request and verifying that the application is authorized to set the hardware interrupt timer;

relaying the validated request to the hardware-dependent process, the hardware-dependent process comprising:

(a) a timer interrupt service routine containing logic to set an expiration time and the application interrupt service routine, and a second set interrupt timer API corresponding to the first set interrupt timer API;

setting the hardware interrupt timer in the hardware-dependent process to expire in accordance with the validated request, wherein setting the hardware interrupt timer comprises storing the expiration time in the hardware interrupt timer and storing the expiration time in the area of the application managed memory;

inserting the application interrupt service routine in the timer interrupt service routine scheduled to execute upon expiration of the timer; and

returning control to the application when execution of the timer interrupt service routine and the inserted application interrupt service routine is complete.

2. The method of claim **1**, wherein the group of parameters also includes arm interval representing a period of time after which the hardware interrupt timer is requested to expire, and wherein validating the request includes determining that the interval is of substantially sufficient duration to set the hardware interrupt timer.

3. The method of claim **1**, wherein validating the request includes determining that the mode is one of periodic and aperiodic.

4. The method of claim **1**, wherein the group of parameters also includes a parameter that specifies an application interrupt service routine that is to be executed upon expiration of the hardware interrupt timer, and wherein validating the request includes determining that the application interrupt service routine is properly registered.

5. The method of claim **4**, wherein the group of parameters also includes a parameter that specifies a device object, and wherein validating the request includes determining that the application service routine corresponds to the device object.

6. The method of claim **4**, wherein setting the hardware interrupt timer includes registering a timer service routine to be executed upon expiration of the hardware interrupt timer, the timer service routine being modified to run the application service routine.

7. A system to configure a timer in a computing device, the system comprising:

a timer substantially guaranteed to expire at a time certain;

a hardware-independent interface to the timer, wherein the hardware-independent interface is a kernel mode routine having a set interrupt timer application programming interface (API) for receiving parameters associated with a request from the application to set the timer, and validating the request, wherein validating the request includes validating the parameters by the hardware-independent interface;

a hardware-dependent interface to the timer; and

a processor in which the hardware-independent interface operates to validate a request from an application to set the timer and to relay the validated request to the hardware-dependent process, and further in which the hardware-dependent interface operates to set the timer to expire in accordance with the validated request and to execute a timer interrupt service routine upon expiration of the timer.

8. The system of claim 7, wherein the timer is a high precision event timer (HPET).

9. The system of claim 8, wherein the hardware-dependent interface operates to set the timer by writing an actual time at which the HPET should expire to a comparator register associated with the HPET, the actual tune being determined by the hardware-dependent interface in accordance with the validated request.

10. The system of claim 7, wherein the parameters specify an interval representing a period of time after which the hardware interrupt timer is requested to expire, and wherein the processor operates to validate the request by determining that the interval is of substantially sufficient duration to set the timer.

11. The system of claim 7, wherein the parameters specify a mode in which the timer is requested to operate, and wherein the processor operates to validate the request by determining that the mode is one of periodic and aperiodic.

12. The system of claim 7, wherein the hardware-dependent interface is a hardware application layer (HAL) routine having an interface to receive the validated parameters associated with the request relayed from the hardware-independent interface.

13. The system of claim 7, wherein the hardware-dependent interface further operates to execute an application service routine upon expiration of the timer.

14. A computer-accessible medium having instructions for setting a timer in a computing device when executed by a processor included in said computing device, the instructions comprising:

a hardware-independent process to:

(a) receive a request from an application to set a timer in the computing device, the timer being substantially guaranteed to expire at a time certain; wherein said request contains at least one parameter, the at least one parameter chosen from the group comprising:

(i) a mode in which the hardware interrupt timer is requested to operate;

(ii) an interval;

(iii) a reference to an area of the application managed memory in which a hardware-dependent process is to store-a value representing an actual, time at which the hardware interrupt timer has been set to expire in accordance with the validated request, and

(iv) an interrupt request level at which the application interrupt service routine should execute;

(b) determine whether the application is privileged to make the request;

(c) validate parameters associated with the request; and

a hardware-dependent process to set the timer to expire in accordance with the validated parameters.

15. The computer-accessible medium of claim 14, wherein the instructions comprising the hardware-dependent process further include instructions to:

insert an application service routine in a timer service routine scheduled to execute upon expiration of the timer; and

return control to the application when execution of the timer service routine and inserted application service routine is complete.

* * * * *