EXHIBIT 51

IA-PC HPET (High Precision Event Timers) Specification

Revision: 1.0a
Date: October 2004

LEGAL DISCLAIMER

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted herein.

This specification is a preliminary draft provided for comment and informational purposes only, and is subject to change without any notice, obligation or liability. Readers should not rely on this specification in any way for product design purposes.

It is Intel's intent to provide a Version 1.0 of this specification, that will be made available subject to an appropriate license agreement. However, Intel is under no obligation or liability to do so.

The contents of the areas marked as "reserved" in this specification will not be licensed under the Intel license for this specification.

Table of Contents

1. IA-PC HPET

1.1 Revision History:

Version	Comments
0.97	Last Updated: 03/07/2000 • Incorporated technical editing changes, released for external feedback.
0.97a	Last Updated: 05/18/2000 • Incorporated various non-technical and legal feedbacks.
0.98	01/20/2002 • Product name changed: from **Multimedia Timer** to **HPET** (High Precision Event Timer) • Technologic term changed: from **Legacy Mode** to **LegacyReplacement Mode** for clarity purpose • ETDT ACPI table changed: **ETDT** (Event Timer Descriptor Table) is changed to **HPET** table and its content of the table has been updated. • IA64 platform support: Use GAS(Generic Address Structure) format in HPET table and up to 64KB timer block.
0.98a	08/31/2001 • Modified the accuracy of clock frequency drift to 0.05% • Add "write lock" note to the programming requirement
1.0	6/8/2004 • Removed color-code for read-only fields • Added programming notes for 64-bit register access in a 64-bit platform • Explicitly mark "Reserved" in reserved fields of FSB Registers
1.0a	6/8/2004 • Clarifications to 64 bit accesses. • General cleanup

1.2 Scope

This specification provides register model and programming interface definitions for new event timer hardware for use on Intel Architecture-based Personal Computers. In this specification, the terms 'IA-PC HPET and 'Event Timers' refer to the same timer hardware.

The IA-PC HPET Specification defines timer hardware that is intended to initially supplement and eventually replace the legacy 8254 Programmable Interval Timer and the Real Time Clock Periodic Interrupt generation functions that are currently used as the 'de-facto' timer hardware for IA-PCs.

This new timer hardware can be used by system software for:

- **Synchronizing**
  - Real-Time Digital Audio & Video Streams
    - 64-bit free running up-counter
- **Scheduling**
  - Threads, Tasks, Processes, etc.
    - Fixed Rate (Periodic) Interrupt Generation
      - System Heart Beat
      - Non-Real Time Thread Scheduler
    - Variable Rate (One-Shot) Interrupt Generation
      - Scheduling real time tasks associated with host-based signal processing applications
- **Time Stamping**
  - On Multiprocessor platforms
    - 64-Bit free running up-counter can be utilized as DIG64 "platform timer" for Time Stamping Applications. This provides a time-base that is insensitive to clock frequency drifts on individual CPU's on a N-Way MP systems.

Note:
The name of the timer block has been changed from Multimedia Timer to HPET (High Precision Event Timer). However, before the new name was adopted, many related documents continue to use or reference the term of "Multimedia Timer". Therefore, for the purposes of designing products to this specification, the terms HPET, Multimedia Timer, MMT and MM Timer should be treated as the same timer hardware.

EXHIBIT 52

US006897904B2

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

(10) **Patent No.: US 6,897,904 B2**
(45) **Date of Patent: May 24, 2005**

(54) **METHOD AND APPARATUS FOR SELECTING AMONG MULTIPLE TUNERS**

(75) Inventors: **Peter J. Potrebic**, Calistoga, CA (US); **Geoffrey Smith**, Mountain View, CA (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 583 days.

(21) Appl. No.: **10/039,225**

(22) Filed: **Jan. 4, 2002**

(65) **Prior Publication Data**

US 2003/0128302 A1 Jul. 10, 2003

(51) **Int. Cl.**[7] **H04N 5/50**
(52) **U.S. Cl.** **348/731**; 348/732
(58) **Field of Search** 348/731, 732, 348/733, 565, 566, 567, 569; 386/46, 83; H04N 5/50

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,272,784	A *	6/1981	Saito et al.	386/83
5,757,441	A *	5/1998	Lee et al.	348/731
6,188,448	B1 *	2/2001	Pauley et al.	348/731

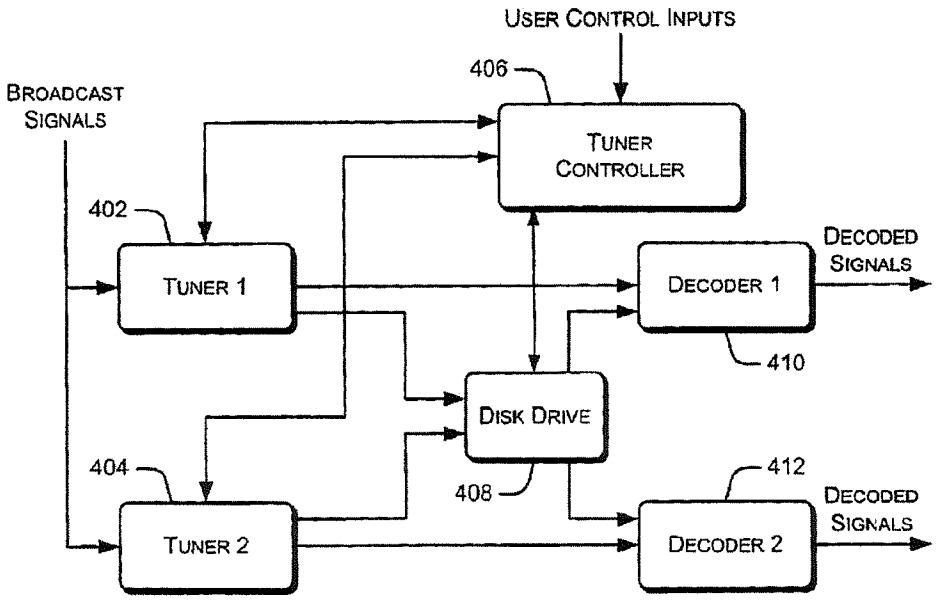
* cited by examiner

Primary Examiner—Michael H. Lee
(74) *Attorney, Agent, or Firm*—Lee & Hayes, PLLC

(57) **ABSTRACT**

A system or method selects among multiple tuners to tune a particular channel. A request is received to tune a first channel. In response to this request, a first tuner is assigned to tune the first channel. A request is received to tune a second channel. If the program tuned by the first tuner is not being recorded, the first tuner is assigned to tune the second channel. If the program tuned by the first tuner is being recorded, the second tuner is assigned to tune the second channel.

24 Claims, 6 Drawing Sheets



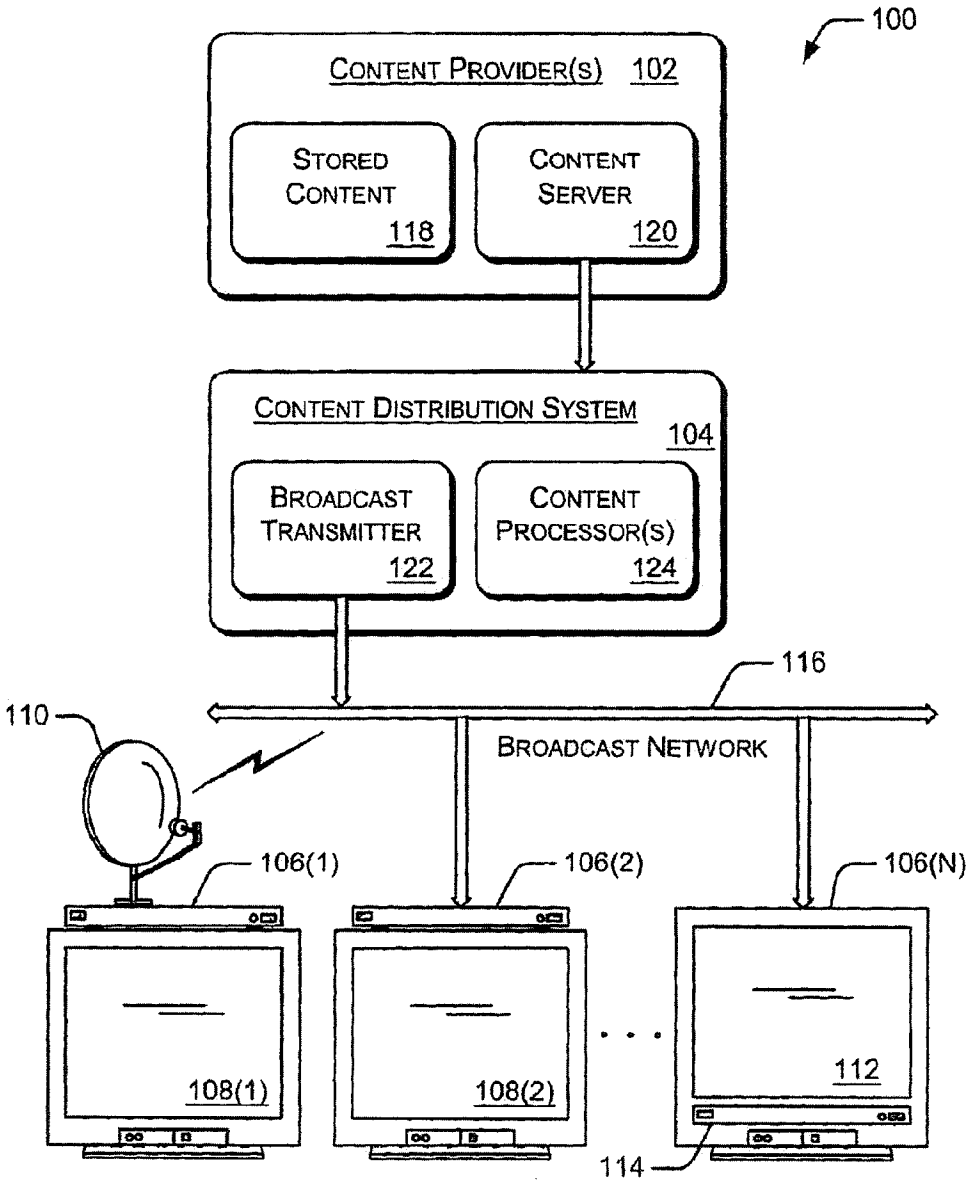

100
CONTENT PROVIDER(S) 102
STORED CONTENT 118
CONTENT SERVER 120
CONTENT DISTRIBUTION SYSTEM 104
BROADCAST TRANSMITTER 122
CONTENT PROCESSOR(S) 124
116
110
BROADCAST NETWORK
106(1)
106(2)
106(N)
108(1)
108(2)
112
114

Fig. 1

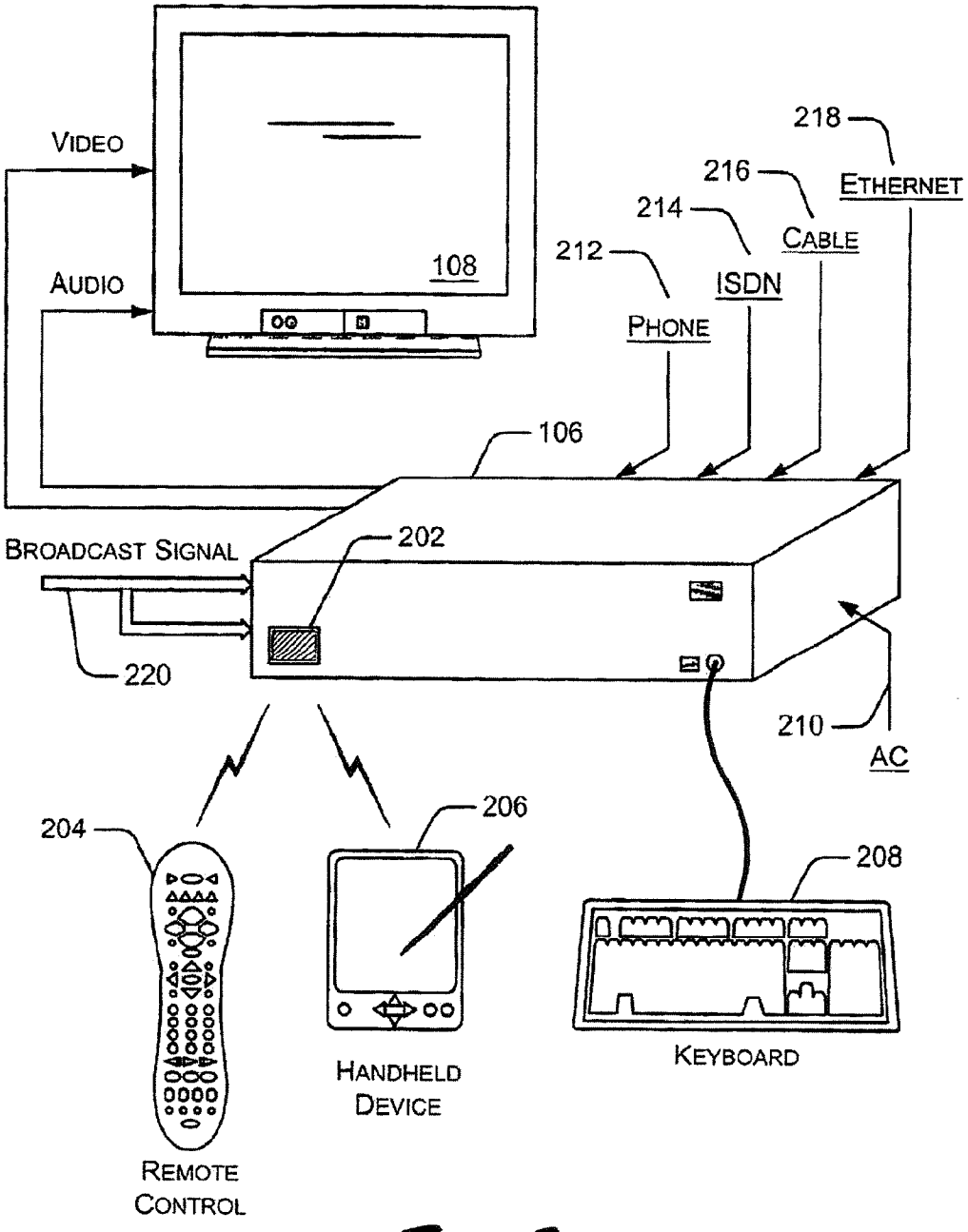

VIDEO
AUDIO
108
218
216
214
212
ETHERNET
CABLE
ISDN
PHONE
106
BROADCAST SIGNAL
202
220
210
AC
204
206
208
REMOTE CONTROL
HANDHELD DEVICE
KEYBOARD

Fig. 2

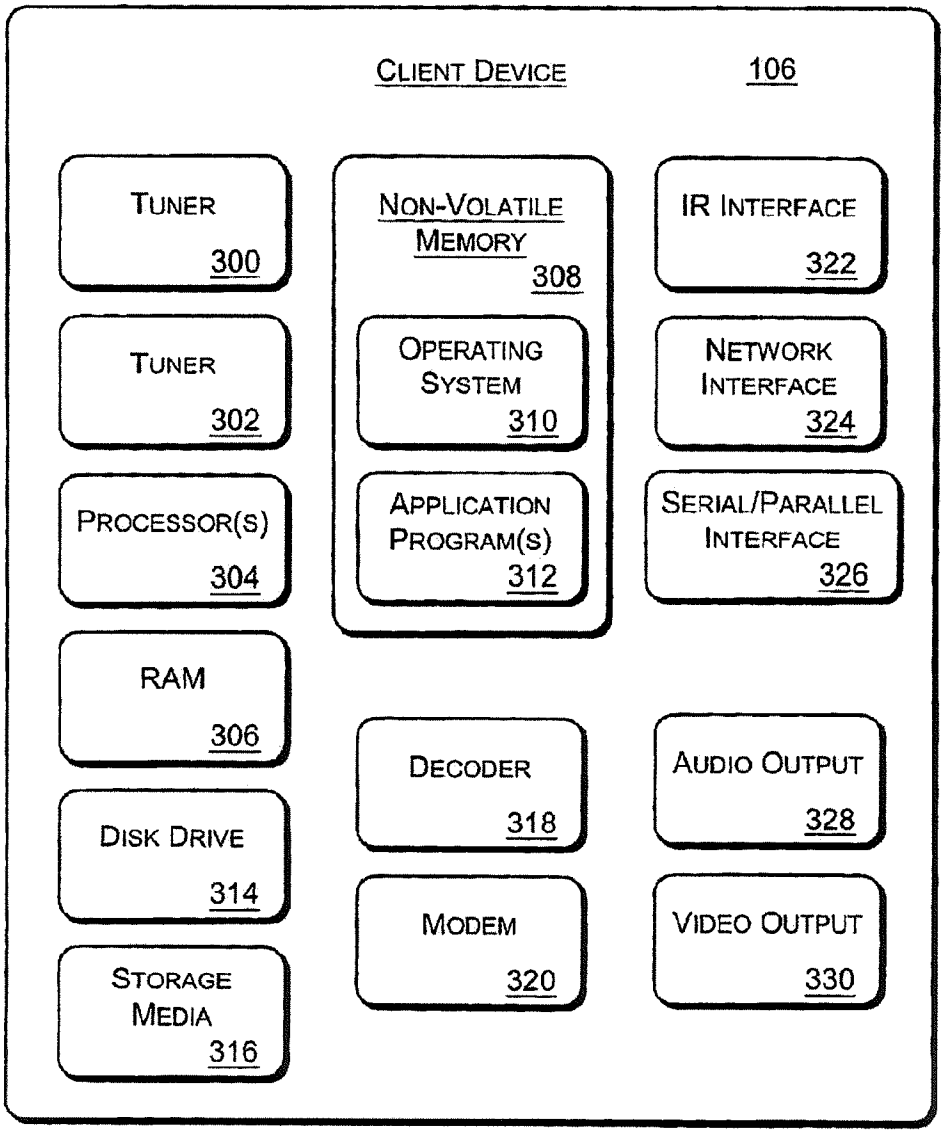

CLIENT DEVICE 106
TUNER 300
TUNER 302
PROCESSOR(S) 304
RAM 306
DISK DRIVE 314
STORAGE MEDIA 316
NON-VOLATILE MEMORY 308
OPERATING SYSTEM 310
APPLICATION PROGRAM(S) 312
DECODER 318
MODEM 320
IR INTERFACE 322
NETWORK INTERFACE 324
SERIAL/PARALLEL INTERFACE 326
AUDIO OUTPUT 328
VIDEO OUTPUT 330

Fig. 3

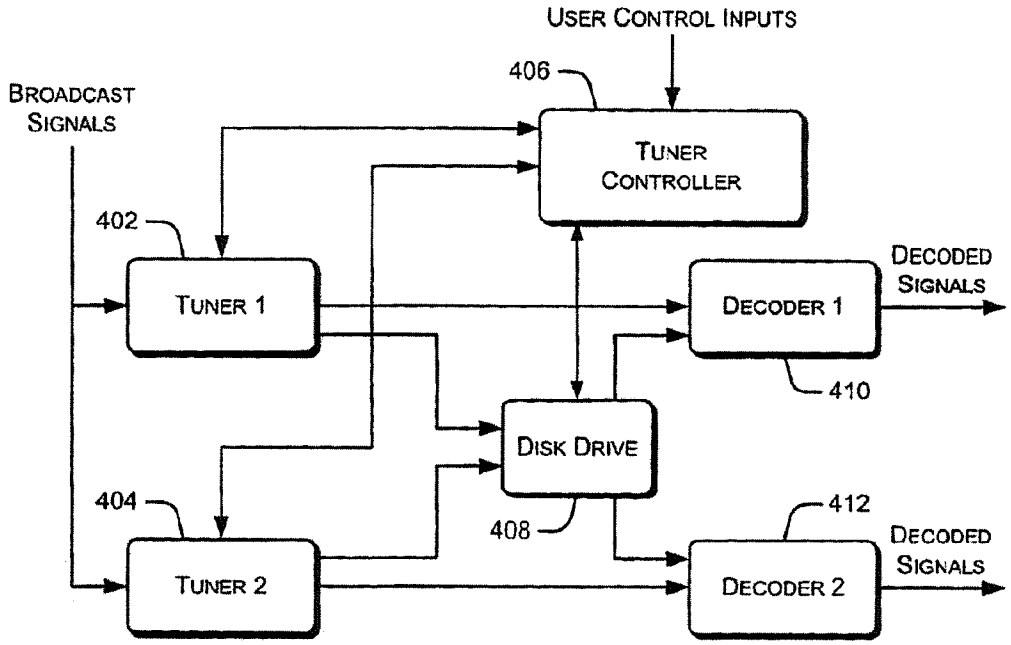

USER CONTROL INPUTS
406
BROADCAST SIGNALS
TUNER CONTROLLER
402
TUNER 1
DECODER 1
DECODED SIGNALS
410
DISK DRIVE
404
408
412
TUNER 2
DECODER 2
DECODED SIGNALS

Fig. 4

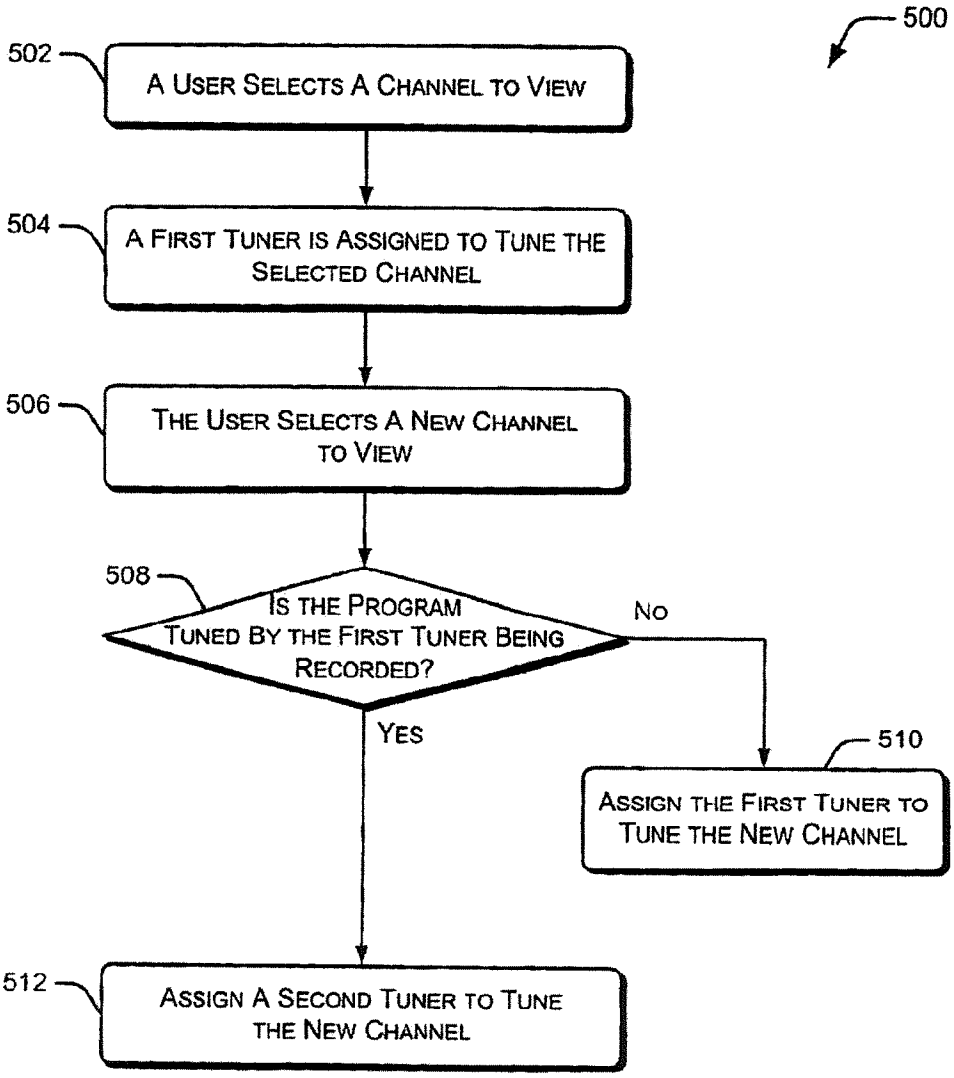

500
502
A USER SELECTS A CHANNEL TO VIEW
504
A FIRST TUNER IS ASSIGNED TO TUNE THE SELECTED CHANNEL
506
THE USER SELECTS A NEW CHANNEL TO VIEW
508
IS THE PROGRAM TUNED BY THE FIRST TUNER BEING RECORDED?
No
YES
510
ASSIGN THE FIRST TUNER TO TUNE THE NEW CHANNEL
512
ASSIGN A SECOND TUNER TO TUNE THE NEW CHANNEL

Fig. 5

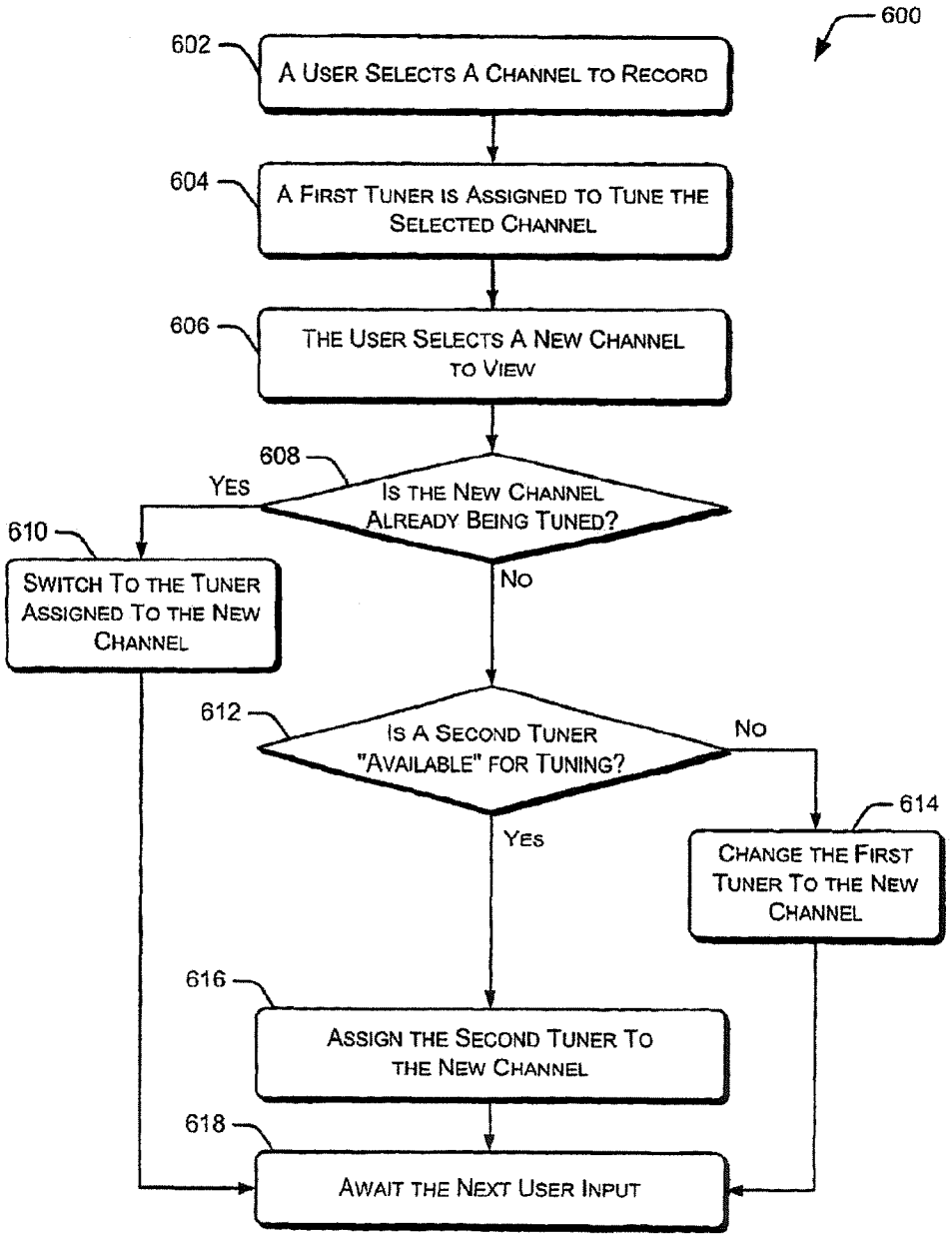

600
602
A USER SELECTS A CHANNEL TO RECORD
604
A FIRST TUNER IS ASSIGNED TO TUNE THE SELECTED CHANNEL
606
THE USER SELECTS A NEW CHANNEL TO VIEW
608
IS THE NEW CHANNEL ALREADY BEING TUNED?
YES
NO
610
SWITCH TO THE TUNER ASSIGNED TO THE NEW CHANNEL
612
IS A SECOND TUNER "AVAILABLE" FOR TUNING?
NO
YES
614
CHANGE THE FIRST TUNER TO THE NEW CHANNEL
616
ASSIGN THE SECOND TUNER TO THE NEW CHANNEL
618
AWAIT THE NEXT USER INPUT

Fig. 6

METHOD AND APPARATUS FOR SELECTING AMONG MULTIPLE TUNERS

TECHNICAL FIELD

This invention relates to selecting among multiple tuners to tune a particular channel.

BACKGROUND OF THE INVENTION

Existing televisions, set top boxes, and other devices that tune broadcast signals contain a single tuner which is capable of tuning one of several different channels contained in a broadcast signal. Since these existing device contain a single tuner, they are not capable of tuning two or more different channels simultaneously (without the help of an external tuning device). Thus, when a user submits a request to change channels, the single tuner is instructed to change to the requested channel. Since there is only one tuner, there is no question as to which tuner will tune the requested channel.

However, with the development of televisions, set top boxes, and other client devices that contain multiple tuners, issues arise regarding which tuner should be used to tune a particular channel when a request to change channels is received. For example, once a particular tuner begins tuning and recording a program, that tuner cannot be used to tune any other channel without stopping the recording process. If a request to change channels is received while a particular channel is being recorded, the system must determine whether the user desires to stop the recording and view the new channel or to continue recording the previous channel while viewing the new channel. Repeatedly prompting the user to select a tuner to tune a particular channel is likely to be frustrating and distracting to the user.

Therefore it is desirable to provide a system that selects among multiple tuners to tune a particular channel in a manner that is least intrusive to the user of the system.

SUMMARY OF THE INVENTION

The systems and methods described herein select among multiple tuners to tune a requested channel. These systems and methods require little or no input from the user as to which tuner should be assigned to tune a particular channel. For example, if a new channel is selected by the user and a tuner is available to tune the new channel, the available tuner is automatically assigned to tune the new channel without requiring any user input or otherwise disrupting the channel selection process.

In a particular embodiment, a request is received to tune a first channel. A first tuner is assigned to tune the first channel. Another request is received to tune a second channel. The first tuner is assigned to tune the second channel if the program tuned by the first tuner is not being recorded. The second tuner is assigned to tune the second channel if the program tuned by the first tuner is being recorded.

Another embodiment receives a request to record a program on a first channel. A first tuner is assigned to tune the first channel. Another request is received to tune a second channel. A determination is made regarding whether the second channel is already being tuned. Another determination is made regarding whether a second tuner is available for tuning. The first tuner is assigned to tune the second channel if the second channel is not already being tuned and the second tuner is not available for tuning. The second tuner is assigned to tune the second channel if the second channel is not already being tuned and the second tuner is available for tuning.

According to one aspect of the invention, the system switches to the tuner assigned to the second channel if the second channel is already being tuned.

BRIEF DESCRIPTION OF THE DRAWINGS

The same reference numerals are used throughout the drawings to reference like components and features.

FIG. **1** illustrates an exemplary environment in which the methods and systems described herein may be implemented.

FIG. **2** is a block diagram of an example client device, a television, and various input devices that interact with the client device.

FIG. **3** is a block diagram of selected components of the client device shown in FIGS. **1** and **2**.

FIG. **4** is a block diagram of an exemplary client device that includes two tuners, a tuner controller, a disk drive, and a pair of decoders.

FIG. **5** is a flow diagram illustrating an embodiment of a procedure for determining which tuner to use when a user selects a new channel to view.

FIG. **6** is a flow diagram illustrating an embodiment of a procedure for determining which tuner to use when one tuner is recording a program and a change channel command is received.

DETAILED DESCRIPTION

FIG. **1** illustrates an exemplary environment **100** in which the methods and systems described herein may be implemented. One or more content providers **102** include stored content **118** and a content server **120**. Content server **120** controls the movement of content (including stored content **118**) from the content provider **102** to a content distribution system **104**, which is coupled to the content provider. Additionally, the content server **120** controls the movement of live content (e.g., content that was not previously stored by the content provider) and content stored at other locations to the content distribution system.

The content distribution system **104** contains a broadcast transmitter **122** and one or more content processors **124**. Broadcast transmitter **122** broadcasts signals (e.g., cable television signals) across a broadcast network **116**, such as a cable television network. Broadcast network **116** may include wired or wireless media using any broadcast format or broadcast protocol. Content processor **124** processes the content received from content provider **102** prior to transmitting the content across the broadcast network **116**. A particular content processor may encode or otherwise process the received content into a format that is understood by multiple client devices **106** coupled to the broadcast network **116**. Although FIG. **1** shows a single content provider **102** and a single content distribution system **104**, a particular environment may include any number of content providers coupled to any number of content distribution systems.

A client device **106(1)** receives broadcast content from a satellite-based transmitter via a satellite dish **110**. Client device **106(1)** is also referred to as a set-top box, game console or a satellite receiving device. Client device **106(1)** is coupled to a television **108(1)** for presenting the content received by the client device (i.e., audio data and video data) as well as a graphical user interface. A particular client device **106** may be coupled to any number of televisions **108**. Similarly, any number of client devices **106** may be

coupled to a television **108**. Another client device **106(2)** is coupled to receive broadcast content from broadcast network **116** and provide the received content to a television **108(2)**. Another client device **106(N)** is a combination of a television **112** and a set-top box **114**. In this example, the various components and functionality of the set-top box are incorporated into the television, rather than using two separate devices. The set-top box incorporated into the television may receive broadcast signals via a satellite dish (similar to satellite dish **110**) and/or via broadcast network **116**. In alternate embodiments, client devices **106** may receive broadcast signals via the Internet or any other broadcast medium.

FIG. **2** is a block diagram of an example client device **106**, television **108**, and various input devices that interact with the client device. As discussed above, client device **106** may also be referred to as a set-top box, a game console or a satellite receiver. Client device **106** includes a wireless receiving port **202** (e.g., an infrared (IR) wireless port) for receiving wireless communications from a remote control device **204**, a handheld device **206** (such as a personal digital assistant (PDA) or handheld computer), or other wireless device, such as a wireless keyboard. Additionally, a wired keyboard **208** is coupled to client device **106** for communicating with the client device. In alternate embodiments, remote control device **204**, handheld device **206**, and/or keyboard **208** may us an RF communication link (or other mode of transmission) to communicate with client device **106**.

Client device **106** receives one or more broadcast signals **220** from one or more broadcast sources (e.g., from a broadcast network or via satellite). Client device **106** includes hardware and/or software for receiving and decoding broadcast signal **220**, such as an NTSC, PAL, SECAM or other TV system video signal, and providing video data to the television **108**. Client device **106** also includes hardware and/or software for providing the user with a graphical user interface by which the user can, for example, access various network services, configure the client device **106**, and perform other functions.

Client device **106** receives AC power on line **110**. Client device **106** is capable of communicating with other devices via a conventional telephone link **212**, an ISDN link **214**, a cable link **216**, and an Ethernet link **218**. A particular client device **106** may use any one or more of the various communication links **212–218** at a particular instant. Client device **106** also generates a video signal and an audio signal, both of which are communicated to television **108**. The video signals and audio signals can be communicated form client device **106** to television **108** via an RF (radio frequency) link, S-video link, composite video link, component video link, or other communication link. Although not shown in FIG. **2**, a particular client device **106** may include one or more lights or other indicators identifying the current status of the client device. Additionally, a particular client device **106** may include one or more control buttons or switches (not shown) for controlling operation of the client device.

FIG. **3** is a block diagram of selected components of the client device **106** shown in FIGS. **1** and **2**. Client device **106** includes multiple tuners **300** and **302**, one or more processors **304**, a random access memory (RAM) **306**, and a non-volatile memory **308** that contains, for example, an operating system **310** and one or more application programs **312**. Client device **106** also includes a disk drive **314** and storage media **316**. Although client device **106** is illustrated having both a RAM **306** and a disk drive **314**, a particular device may include only one of the memory components. Additionally, although not shown, a system bus typically couples together the various components within client device **106**.

Processor(s) **304** process various instructions to control the operation of client device **106** and to communicate with other electronic and computing devices. The memory components (e.g., RAM **306**, disk drive **314**, storage media **316**, and non-volatile memory **308**) store various information and/or data such as configuration information and graphical user interface information.

Client device **106** also includes a decoder **318**, such as an MPEG-2 decoder that decodes MPEG-2-encoded signals. A modem **320** allows client device **106** to communicate with other devices via a conventional telephone line. An IR interface **322** allows client device **106** to receive input commands and other information from a user-operated device, such as a remote control device or an IR keyboard. Client device **106** also includes a network interface **324**, a serial/parallel interface **326**, an audio output **328**, and a video output **330**. Interfaces **324** and **326** allow the client device **106** to interact with other devices via various communication links. Although not shown, client device **106** may also include other types of data communication interfaces to interact with other devices. Audio output **328** and video output **330** provide signals to a television or other device that processes and/or presents the audio and video data. Although client **106** is illustrated having multiple interfaces, a particular client may only include one or two such interfaces.

Client device **106** also includes a user interface (not shown) that allows a user to interact with the client device. The user interface may include indicators and/or a series of buttons, switches, or other selectable controls that are manipulated by a user of the client device.

General reference is made herein to one or more client devices, such as client device **106**. As used herein, "client device" means any electronic device having data communications, data storage capabilities, and/or functions to process signals, such as broadcast signals, received from any of a number of different sources.

FIG. **4** is a block diagram of a portion of an exemplary client device that includes two tuners, a tuner controller, a disk drive, and a pair of decoders. The exemplary client device contains additional components (such as those shown in FIG. **3**) which are omitted from FIG. **4** to simplify discussion of specific components. A pair of tuners **402** and **404** each receive broadcast signals from a one or more broadcast sources. Certain broadcast signals may include encoded data, such as encoded video data and encoded audio data. In this situation, a decoder is used to decode the encoded data.

In a particular embodiment, the broadcast signals include data encoded using the MPEG-2 (Moving Pictures Experts Group) encoding format. MPEG-2 is a standard for digital video and digital audio compression. MPEG-2 supports a variety of audio/video formats, including legacy TV, HDTV (High-Definition Television), and five channel surround sound. However, the methods and systems described herein can be used with any type of signal using any type of encoding format as well as signals that do not use any encoding.

Referring again to FIG. **4**, each tuner **402** and **404** maintains an internal "state". This state information may include, for example, the channel currently being tuned (if any) and whether the tuner is creating a pause buffer (e.g.,

for "pausing" a live broadcast) or creating an archival recording of the program. Each tuner **402** and **404** is coupled to a tuner controller **406**. Tuner controller **406** receives user control inputs, such as a user request to change channels or record a program being broadcast on a particular channel. These user control inputs may be provided through a remote control device, wireless keyboard, or other input device. Tuner controller **406** processes the received user control inputs and sends control instructions, if necessary, to tuner **402** and/or tuner **404**. For example, if the user requests to change from viewing channel **204** to channel **206**, the tuner controller **406** instructs one of the tuners **402** or **404** to begin tuning channel **206**. Additionally, tuners **402** and **404** can communicate information to tuner controller **406**, such as the current channel being tuned by the tuner. Although two tuners are illustrated in FIG. **4**, a particular implementation may contain any number of tuners.

A disk drive **408** is coupled to tuners **402** and **404**, tuner controller **406**, and a pair of decoders **410** and **412**. Disk drive **408** is capable of storing program data received from tuner **402** and/or **404** and replaying that program data at a later time. Tuner controller **406** controls the recording of programs by sending appropriate commands to disk drive **408**. Disk drive **408** may also store other information used by the client device such as configuration information. Disk drive **408** outputs encoded program content to decoder **410** and/or **412**. The decoder **410**, **412** then decodes the encoded program content and outputs decoded signals, such as decoded video signals and decoded audio signals. Tuners **402** and **404** are also coupled to decoders **410** and **412** and may provide tuned signals directly to decoder **410**, **412** if the tuned signal is being watched live (i.e., not being played back from the disk drive **408**). Although FIG. **4** illustrates two decoders **410** and **412**, alternate embodiments include a single decoder that decodes signals from both tuners **402** and **404**, or from disk drive **408**.

Alternatively, disk drive **408** may output signals directly (i.e., not through decoder **410**, **412**) if the program content stored on the disk drive does not require decoding. Similarly, tuners **402**, **404** may output signals directly if the program content being tuned does not require decoding.

FIG. **5** is a flow diagram illustrating an embodiment of a procedure **500** for determining which tuner to use when a user selects a new channel to view. Initially, a user selects a channel to view (block **502**). A first tuner is assigned to tune the channel selected by the user (block **504**). The user then selects a new channel to view (block **506**). The procedure **500** then determines whether the program being tuned by the first tuner is being recorded (block **508**). This determination can be performed by querying the state of the first tuner to determine whether the program is being recorded. If the program being tuned by the first tuner is not being recorded, the procedure assigns the first tuner to tune the new channel (block **510**). If the program being tuned by the first tuner is being recorded, the procedure assigns a second tuner to tune the new channel (block **512**). The manner in which a tuner is selected for the new channel minimizes disruption to the user of the client device. For example, if the client device is already recording a program on one channel, that recording is not disturbed by the user's request to view a different channel. Instead of changing the channel assigned to the tuner that is tuning the recorded channel, a second tuner is used to tune the new channel.

FIG. **6** is a flow diagram illustrating an embodiment of a procedure **600** for determining which tuner to use when one tuner is recording a program and a change channel command is received. Initially, a user selects a channel to record (block **602**). A first tuner is assigned to tune the selected channel (block **604**). The user then selects a new channel to view (block **606**). The procedure **600** then determines whether the new channel is already being tuned by another tuner (block **606**). If the new channel is already being tuned by another tuner, then block **610** of the procedure switches to the tuner assigned to the new channel (i.e., the new "active" tuner is the tuner that was already tuning the newly selected channel). The procedure then awaits the next user input (block **618**). If the "new channel" was being recorded, the user is provided with a graphical indication on the television screen and/or an audible sound indicating that the program is being recorded.

By checking to see if the requested channel is already being tuned by another tuner, the system maintains the highest number of available tuners and avoids the situation where one tuner is recording a program from a particular channel and another tuner is being used to tune and display the program from the same channel. Also, by switching back to a tuner that is tuning and recording the requested channel, the user has access to the previously recorded program content. Certain systems empty the "pause buffer" (i.e., recorded portions of the program) in response to a channel change. If a new tuner was used to tune and display the selected channel, the previously recorded portions of the program would not be available to the user. However, by switching control to the tuner already recording the content, the user has access to the recorded content.

When switching back to a channel that is being recorded, the system can begin displaying the program content currently being tuned by the tuner. Alternatively, the system can begin displaying previously recorded program content, such as playing back the recorded program starting at the beginning of the program or playing back the recorded program from the point at which the user previously changed channels (i.e., switched away from the recorded program).

Referring again to FIG. **6**, if the new channel selected at block **606** is not already being tuned by another tuner, the procedure **600** determines whether a second tuner is "available" for tuning the new channel (block **612**). An "available" tuner as used herein may be defined in several different ways. An "available" tuner may be idle or may be engaged in a task that has a lower priority than the priority of allowing a user to change a channel being viewed. For example, using the tuner to receive non-urgent data may be a lower priority than changing a viewed channel, but using the tuner to record a second program may be a higher priority than changing a viewed channel. One tuner can determine whether another tuner is available by querying the other tuner to determine its status and priority.

If a second tuner is not available for tuning the new channel, then the procedure changes the first tuner to the new channel (block **614**). The procedure then awaits the next user input (block **618**). If a second tuner is available for tuning the new channel, then the procedure assigns the second tuner to the new channel (block **616**). The procedure then awaits the next user input (block **618**).

In a particular embodiment, recorded program content is associated with the tuner that originally tuned the recorded program. When a first tuner is tuning program content that is being recorded and a problem occurs with the first tuner or a higher priority task is assigned to the first tuner, a new tuner is selected to tune the program. In this situation, the recorded program content may be changed such that the recorded program content is associated with the new tuner. Thus, the user is still able to view the previously recorded

program content (tuned by the first tuner) even though the first tuner is no longer available. For example, if the first tuner malfunctions or the signal line providing broadcast signals to the first tuner is damaged, the tuning operation is switched to a second tuner and the previously recorded program content is modified to be associated with the new tuner. Alternatively, the program content tuned by the second tuner may be stored as a separate file that is "linked" to the previously recorded program content. By linking the two files, the user is able to easily access the entire recorded program content even though the program content is saved in two different files.

In one embodiment, one or more user interface features are provided that indicate to the user that, upon switching back to a channel that is being recorded, the user is now watching a recorded show. This indication reassures the user that their recording is proceeding properly. This indication also reminds the user that they have the ability to access previously recorded portions of the program, if desired.

Portions of the systems and methods described herein may be implemented in hardware or a combination of hardware, software, and/or firmware. For example, one or more application specific integrated circuits (ASICs) or programmable logic devices (PLDs) could be designed or programmed to implement one or more portions of the systems and procedures described herein.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

What is claimed is:

1. A method comprising:

receiving a request to tune a first channel;

assigning a first tuner to tune the first channel;

receiving a request to tune a second channel;

assigning the first tuner to tune the second channel if the program tuned by the first tuner is not being recorded; and

assigning a second tuner to tune the second channel if the program tuned by the first tuner is being recorded, wherein assigning a second tuner to tune the second channel includes determining whether a second tuner is available for tuning.

2. A method comprising:

receiving a request to tune a first channel;

assigning a first tuner to tune the first channel;

receiving a request to tune a second channel;

assigning the first tuner to tune the second channel if the program tuned by the first tuner is not being recorded;

assigning a second tuner to tune the second channel if the program tuned by the first tuner is being recorded; and

assigning a third tuner to tune the first channel if the first tuner is unable to tune the first channel.

3. A method comprising:

receiving a request to tune a first channel;

assigning a first tuner to time the first channel;

receiving a request to tune a second channel;

assigning the first tuner to tune the second channel if the program tuned by the first tuner is not being recorded;

assigning a second tuner to tune the second channel if the program tuned by the first tuner is being recorded; and

assigning a third tuner to tune the second channel if the second tuner is unable to tune the second channel.

4. A method comprising:

receiving a request to tune a particular channel;

determining whether a second tuner is available for tuning;

assigning a first tuner to tune the particular channel if the program currently tuned by the first tuner is not being recorded; and

assigning the second tuner to tune the second channel if the second tuner is available for tuning and the program currently tuned by the first tuner is being recorded.

5. A method as recited in claim 4, further comprising assigning the first tuner to tune the particular channel if the second tuner is not available for tuning.

6. A method as recited in claim 4, further comprising assigning a third tuner to tune the particular channel if the first tuner and the second tuner are not available to tune the particular channel.

7. A method as recited in claim 4, wherein determining whether a second tuner is available for tuning includes a first tuner querying the second tuner.

8. A method comprising:

receiving a request to record a program on a first channel;

assigning a first tuner to tune the first channel;

receiving a request to tune a second channel;

determining whether the second channel is already being tuned;

determining whether a second tuner is available for tuning;

assigning the first tuner to tune the second channel if the second channel is not already being tuned and the second tuner is not available for tuning; and

assigning the second tuner to tune the second channel if the second channel is not already being tuned and the second tuner is available for tuning.

9. A method as recited in claim 8, further comprising switching to the tuner assigned to the second channel if the second channel is already being tuned.

10. A method as recited in claim 8, further comprising assigning a third tuner to tune the first channel if the first tuner is not available to tune the first channel.

11. A method as recited in claim 8, wherein determining whether a second tuner is available for tuning includes querying the second tuner to determine the status of the second tuner.

12. A method comprising:

receiving a request to record a program on a first channel;

assigning a first tuner to tune the first channel;

receiving a request to tune a second channel;

assigning a second tuner to tune the second channel if the second tuner is available for tuning;

receiving a request to tune the first channel;

switching to the first tuner; and

displaying an indicator that the user is now watching a recorded program.

13. A method as recited in claim 12, wherein the recorded program data from the first channel is associated with the first tuner.

14. A method as recited in claim 12, further comprising changing the status of the second tuner to an available state.

15. A method as recited in claim 12, further comprising assigning the second tuner to tune the first channel if the first tuner is not available to tune the first channel.

16. A method as recited in claim 15, wherein the recorded program data associated with the first tuner is modified to be associated with the second tuner if the first tuner is not available to tune the first channel.

17. A method as recited in claim 15, wherein a new set of recorded program data is generated and associated with the second tuner.

18. A method as recited in claim 12, wherein the method is performed by a set top box.

19. A method as recited in claim 12, wherein switching to the first tuner includes displaying the program content currently being tuned by the first tuner.

20. A method as recited in claim 12, wherein switching to the first tuner includes displaying previously recorded program content if the first tuner has been recording the tuned content.

21. One or more computer-readable media having stored thereon a computer program that, when executed by one or more processors, causes the one or more processors to:

receive a request to tune a first channel;

assign a first tuner to tune the first channel;

receive a request to tune a second channel;

determine whether a second tuner is available for tuning;

assign the first tuner to tune the second channel if the second tuner is not available for tuning; and

assign the second tuner to tune the second channel if the second tuner is available for tuning.

22. One or more computer-readable media as recited in claim 21, wherein the second tuner is queried for a status of the second tuner to determine whether the second tuner is available for tuning.

23. One or more computer-readable media as recited in claim 21, further causing the one or more processors to assign a third tuner to tune the second channel if the first tuner and the second tuner are not available to tune the second channel.

24. One or more computer-readable media as recited in claim 21, further causing the one or more processors to display an indicator if the second channel is being recorded.

* * * * *

EXHIBIT 53

SMAPI User's Guide

IBM ViaVoice Software Developer's Kit

Table of Contents

About This Document

This document provides detailed information on developing speech-aware applications using the IBM ViaVoice? Software Developer's Kit (SDK) and the IBM Speech Manager API (SMAPI) interface set.

Who Should Read This Document

Read this document if you are a programmer interested in writing speech-aware applications that use ViaVoice for speech and are built using the IBM SMAPI interface set.

How This Document Is Organized

This document is divided into the following parts:

1. Introduction, describes speech-aware application programming using the ViaVoice SDK, discusses how to design applications that use ViaVoice.
2. Developing Command and Control Applications, explains how to develop command and control applications using the ViaVoice SDK. It covers identifying grammars, creating grammar files, compiling grammars, building a dictionary, and testing grammars. It also includes information on using dynamic command vocabularies as a method for implementing command and control in an application.
3. Developing Dictation Applications, explains how to develop dictation applications using the ViaVoice SDK. It presents basic functions that dictation applications should support. It also covers the steps involved in developing dictation macros and templates.
4. Speech Manager API (SMAPI), provides an overview of the Speech Manager APIs, describes function call processing, and explains the tasks associated with each phase of a speech-aware application session.
5. Dictation Macro API (DMAPI), provides an overview of the Dictation Macro APIs and explains the tasks involved in supporting dictation macros and templates.
6. SMAPI Grammar Compiler API, provides an overview of the SMAPI Grammar Compiler APIs and explains the tasks involved in compiling grammars from within an application.
7. Custom Audio DLLs, provides an overview of the Custom Audio DLLs and explains the tasks involved with using multiple audio sources within an application.
8. Appendix A, Notices, describes copyright and trademark information for this publication.
9. Appendix B, Glossary, defines terms and abbreviations found in this publication.

Related Publications

Refer to the following publications included with this version for additional programming, reference, and design information:

1 Introduction to SMAPI Developer's Guide

The IBM ViaVoice SDK for Windows, 95/98 or Windows NT provides programmers with the necessary tools to develop applications that incorporate speech. It includes a robust set of application programming interfaces (APIs) that allows an application to access speech resources. It contains several utilities that enable developers to define and manage what the user can say within an application. There are also several sample programs that can help programmers as they develop their applications for speech. Finally, there are distributable runtime elements that are included with an application that uses IBM ViaVoice.

1.1 IBM Native Architecture Overview

IBM SMAPI supports only speech recognition functions. The SMAPI interface set is the native interface for the ViaVoice engine. This section contains a description of the overall architecture of ViaVoice.

The heart of a speech recognition system is known as the speech recognition engine. The speech recognition engine recognizes speech input and translates it into text that an application understands. The application decides what to do with the recognized text. It can transcribe it literally for dictation, or it can act on it for commands.

Applications can access the speech recognition engine through a speech recognition API. For ViaVoice, this API is known as the Speech Manager API, or SMAPI, for short. SMAPI is a conventional API. This means that the API is defined as part of the resource; in this case, SMAPI is defined as part of the speech engine. With an API, speech becomes a resource to all applications, just like any system resource (mouse, video, and so on). Any of the ViaVoice SDK interfaces can be used in this manner, but SMAPI is the focus of the material in this guide.

Windows developers note: The SMAPI interface set cannot be used in conjunction with other speech interfaces, such as SAPI, from within the same source code.

1.1.1 Speech Resources

Before we can discuss the architecture of the speech recognition engine, we should first tell you about the various resources with which the speech recognition engine works. The speech recognition engine uses the following resources to process spoken words:

- User's language of origin
- Domains

1.1.1.1 User's Language of Origin

The language of origin is the language used by the speaker. ViaVoice supports U.S. English, six European, and three Asian languages. The supported languages and their associated language keys are:

Data Files], page 108). The personal extensions added with SmWordCorrection persist across recognition sessions and can be removed with SmRemoveFromVocab.

The code sample below shows how to use SmEnableVocab to enable the ViaVoice predefined dictation vocabulary, called "text". Since dictation vocabularies are all predefined, there is no need for an explicit function call to define the vocabulary.

```
SM_MSG reply;
int rc;

/*-------------------------------------------------------------*/
/* Enable the predefined "text" vocabulary.                    */
/*-------------------------------------------------------------*/

rc = SmEnableVocab("text",        // Name of vocabulary
                   &reply);       // synchronous form

if (rc != SM_RC_OK)
   // Error processing goes here...
```

13.2.2 Processing Speech Input

The bulk of the work that a speech-aware application performs involves getting the recognized text from the engine and determining what to do with it. The responsibilities of an application in this area are:

- Ensuring that the application has speech focus when it needs it. If the application needs to have speech focus whenever it has input focus, then it needs to call SmRequestFocus when it receives notification from Windows that it has input focus. For a complete discussion of issues related to speech focus, refer to "Speech Focus" on page 148.
- Managing and displaying current microphone state. Having a visual display that indicates the current microphone state is a good idea, and it makes the application easier to use. Also, a well-designed speech application responds to asynchronous requests to turn the microphone on or off.
- Keeping audio processing moving by handling recognized text messages and ensuring that an SmRecognizeNextWord call is pending while the engine decoder is running. For command or grammar vocabulary processing, this means calling SmRecognizeNextWord each time it receives an SM_RECOGNIZED_WORD or SM_RECOGNIZED_PHRASE message from the engine.

13.2.2.1 Vocabulary Processing

The speech recognition engine processes words from command and grammar vocabularies differently from the way it processes dictation vocabularies, and it sends the data to the application in different messages. In all cases, however, the engine uses the SM_WORD structure to communicate relevant information about each word to the application (refer to "Data Types" in the API Reference for a description of the SM_WORD data type).

When the engine finds the best match for a spoken word in a command vocabulary, the decoded word and the alternative choices made by the engine are stored in an array of SM_WORD structures and sent to the application in an SM_RECOGNIZED_WORD message. After sending the SM_RECOGNIZED_WORD message, the engine stops decoding until the application calls SmRecognizeNextWord. Use SmGetFirmWords to retrieve the recognized text from the message.

When the engine finds the best match for a spoken word in a grammar vocabulary, it creates an array of SM_WORD structures containing the decoded word and alternative choices and passes it to the application in an SM_RECOGNIZED_PHRASE message. After sending the SM_RECOGNIZED_PHRASE message, the engine stops decoding until the application calls SmRecognizeNextWord. Use SmGetFirmWords or SmGetAnnotations to retrieve the recognized text from the message.

When the engine finds the best match for a spoken word in a dictation vocabulary, it creates an array of SM_WORD structures for the firm words, and returns this array in an SM_RECOGNIZED_TEXT message. To improve accuracy during dictation, the engine uses the neighboring words to help in its selection of the best decoded word. After sending the SM_RECOGNIZED_TEXT message, the engine continues decoding, and sends additional, asynchronous SM_RECOGNIZED_TEXT messages as subsequent words are recognized. Use SmGetFirmWords to retrieve the recognized text from the message.

If a recognized word occurs in two or more vocabularies enabled at the same time, the engine selects the word from the more recently enabled command or grammar vocabulary. Command vocabularies always override dictation vocabularies. When a recognized word occurs in a command vocabulary and a dictation vocabulary that are enabled at the same time, the engine selects the command vocabulary word. As a result, if a command vocabulary with a single word (such as "Stop Dictation") is enabled during a dictation session, the user can indicate the end of the session with a spoken command.

If a word from a command vocabulary follows words from a dictation vocabulary, the engine sends an SM_RECOGNIZED_TEXT message before it sends the SM_RECOGNIZED_WORD message.

13.2.2.2 Handling Rejections

Occasionally, the engine cannot adequately match an input sound against any word in the defined vocabularies. This might be caused by an out-of-vocabulary command, or by extraneous noise or speech. The engine returns an indication of these events to the application, so that it can inform the user that some sound was processed, and it was rejected.

The rejection is conveyed through an SM_RECOGNIZED_WORD message with an empty word spelling string and an empty vocabulary name string. As with all the other SM_RECOGNIZED_WORD messages, the engine then halts, waits for any vocabulary state changes from the application, then continues on an SmRecognizeNextWord call.

Consequently, all applications need to define a handler for the SM_RECOGNIZED_WORD message (or SmRecognizedWordCallback), even if no dynamic command vocabularies are defined. This handler should, at a minimum, issue the SmRecognizeNextWord call when processing a rejection.