

(19)



Europäisches Patentamt

European Patent Office

Office européen des brevets



(11)

EP 0 882 286 B1

(12)

EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention
of the grant of the patent:

21.06.2000 Bulletin 2000/25

(21) Application number: **97907795.5**

(22) Date of filing: **21.02.1997**

(51) Int Cl.7: **G10H 7/02**

(86) International application number:
PCT/US97/02811

(87) International publication number:
WO 97/31363 (28.08.1997 Gazette 1997/37)

(54) **PC AUDIO SYSTEM WITH FREQUENCY COMPENSATED WAVETABLE DATA**

AUDIOSYSTEM FÜR PC MIT FREQUENZKOMPENSIERTEN WELLENFORMDATEN

SYSTEME AUDIO DE MICRO-ORDINATEUR A COMPENSATION EN FREQUENCE DES
DONNEES DE TABLEAUX D'ONDES

(84) Designated Contracting States:
DE FR GB IT

(30) Priority: **21.02.1996 US 604558**

(43) Date of publication of application:
09.12.1998 Bulletin 1998/50

(73) Proprietor: **Advanced Micro Devices, Inc.**
Austin, TX 78741 (US)

(72) Inventor: **HEWITT, Larry**
Austin, TX 78746 (US)

(74) Representative: **Picker, Madeline Margaret**
Brookes & Martin,
High Holborn House,
52/54 High Holborn
London WC1V 6SE (GB)

(56) References cited:
EP-A- 0 474 177 **WO-A-92/15087**
US-A- 4 508 001

EP 0 882 286 B1

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

Description

[0001] This invention relates to a PC audio system including a wavetable audio synthesizer and a memory which supplies frequency compensated wavetable data. More particularly, this invention relates to a PC audio system, including a wavetable audio synthesizer and wavetable cache, which interfaces with a PC system memory supplying frequency compensated wavetable data.

[0002] WO-A-92 15087 is concerned with musical data storage techniques. A plurality of data segments, such as sound recordings, are stored on a mass storage device such as a disc, and a first portion of each sound segment is stored in an IC memory so as to be instantly available. Addressing circuits read the first portion of a data segment stored in the IC memory, and then the portion of the mass storage device so as to give substantially simultaneous playback of data. In order to simulate fast playback of data, the data is recorded on the disc with every *n*th sample in the data stream also being recorded in a fast block, so that when playback speed is increased, only data from the fast block is re-played.

[0003] EP-A-0 474 177 discloses a tone signal generating device, which includes waveform generating means for generating digital waveform sample data at a frequency corresponding to a designated pitch. Sequentially generated digital waveform sample data are operated with generated coefficients which corresponds to a desired interpolation characteristic and the operated data is synthesized to form one sample data. In this way, the interpolation characteristic can be controlled for the desired filter characteristic.

[0004] US-A-4 508 001 discloses an electronic musical instrument having an optical disc memory, which can only be accessed at low speed, and a semiconductor memory which can be accessed at high speed. The semiconductor memory is used to store an initial portion of a waveshape, and the disc memory stores the remaining portion. A readout circuit starts to readout the initial portion and the remaining portion simultaneously, to compensate for the low-speed accessibility of the optical disc memory.

[0005] Several types of digital "synthesizers," i.e. devices that generate sound through audio digital-signal-processing, are now available. One modern type of digital synthesizer is a wavetable synthesizer. Wavetable synthesizers generate sounds through digital processing of entire digitized sound waveforms or portions of digitized sound waveforms stored in wavetable memory. See U.S. patent application serial No. 08/334,461, entitled "Digital Signal Processor Architecture for Wavetable Audio Synthesizer," by Norris, *et al.*

[0006] Wavetable synthesizers generate sounds by "playing back" from wavetable memory, to a digital-to-analog converter (DAC), a particular digitized waveform. The addressing rate of the wavetable data controls the frequency or pitch of the analog output. The bit width of the wavetable data affects the resolution of the sound being generated. For example, better resolution can be achieved with 16-bit wide data versus 8-bit wide data. 16-bit digital audio is becoming the standard in the industry.

[0007] Wavetable synthesizers have application in personal computers. Typically, personal computers are manufactured with only limited audio capabilities. These limited capabilities provide monophonic tone generation to provide audible signals to the user concerning various simple functions, such as alarms or other user alert signals. The typical personal computer system has no capability of providing stereo, high-quality audio which is a desired enhancement for multimedia and video game applications, nor do they have built-in capability to generate or synthesize music or other complex sounds. Musical synthesis capability is necessary when the user desires to use a musical composition application to produce or record sounds through the computer to be played on an external instrument, or through analog speakers and in multimedia (CD-ROM) applications as well.

[0008] Additionally, users at times desire the capability of using external analog sound sources, such as stereo equipment, microphones, and non-MIDI electrical instruments, to be recorded digitally and/or mixed with digital sources before recording or playback through their computer. To satisfy these demands, a number of add-on products have been developed. One such line of products is referred to in the industry as a sound card. These sound cards are circuit boards carrying a number of integrated circuits, many times including a wavetable synthesizer, wavetable memory and other associated circuitry which the user installs in expansion slots provided by the computer manufacturer. The expansion slots provide an interface to the system bus thereby enabling the host processor to access sound generation and control functions on the board under the control of application software. Typical sound cards also provide MIDI interfaces and game ports to accept inputs from MIDI instruments such as keyboard and joysticks for games.

[0009] One prior art sound card is that offered by Advanced Gravis and Forte under the name Ultrasound. This sound card is an expansion slot embodiment which incorporates into one chip (the "GF-1") a wavetable synthesizer, MIDI and game interfaces, DMA control and Adlib Sound Blaster compatibility logic. In addition to this ASIC, the Ultrasound card includes on-board DRAM (1 megabyte) for wavetable data; an address decoding chip; separate analog circuitry for interfacing with analog inputs and outputs; a separate programmable ISA bus interface chip; an interrupt PAL chip; and a separate digital-to-analog/analog-to-digital converter chip. See U.S. patent application serial No. 072,838, entitled "Wave Table Synthesizer," by Travers, *et al.*, which is incorporated herein by reference.

[0010] On-board sound card memory typically has a size of between one-half to four megabytes and stores all the

wavetable data used to synthesize music. At a cost of about \$25.00 per megabyte, sound card memory cost is a significant factor in the overall cost of the sound card. Therefore, if PC system memory could be used to supply the wavetable data, thereby eliminating or reducing the need for sound card memory, sound cards would be less expensive.

[0011] Utilizing PC system memory to store wavetable data, however, raises some concerns. One concern is that available PC system memory is limited and cannot be spared for wavetable data. However, this should be less of a concern in future state-of-the-art PCs which are expected to contain larger system memories and should have space available for wavetable data. Another concern with using system memory is the numerous accesses to memory that are required by prior art synthesizers. For example, prior art wavetable synthesizers which can synthesize thirty-two independent voices (i.e., instrument sounds) must access memory thirty-two times every 22.7 microseconds to retrieve the required data samples. If this number of accesses was made to system memory, an unacceptably high percentage of the system bus bandwidth would be used for synthesizer operations, and thus less of the bus bandwidth could be used for other PC operations.

[0012] A further concern is that the synthesizer might process wavetable data faster than it receives it from system memory (i.e., faster than the system's maximum bus latency). Such a situation would be unacceptable since the processed data would have gaps, and undesirable pops would occur in the synthesized music as it is played.

[0013] Therefore, there is a need for a PC audio system which synthesizes music from wavetable data supplied by system memory, but does not utilize an unacceptable percentage of bus bandwidth. Furthermore, there is a need for a PC audio system which obtains data from system memory at a rate which is at least as fast as the rate it processes data (i.e., the maximum bus latency is less than or equal to the PC audio system's rate of processing data).

[0014] Accordingly, the present invention provides a method of providing a frequency compensated version of a first patch of wavetable data having a first sample frequency and stored in a first location of a memory, wherein said frequency compensated patch is stored in said memory and either said first patch or said frequency compensated patch is accessed from said memory by a digital wavetable audio synthesizer and used to generate digital audio signals having a second sample frequency which is higher than said first sample frequency, said method comprising the steps of:

- (a) accessing said first patch of wavetable data from said first location of said memory;
- (b) deriving, from said first patch of wavetable data, a patch of wavetable data which has a third sample frequency greater than said first sample frequency, wherein said derived patch of wavetable data comprises said frequency compensated patch; and
- (c) storing said frequency compensated patch in a second location of said memory for use by said digital audio synthesizer in generating digital audio signals having said second sample frequency.

[0015] The present invention will be described with reference to a PC audio circuit which is designed to interface with and provide audio enhancement to a host personal computer of the type including a central processor, system memory and system bus. The PC audio circuit includes a cache memory that is of a significantly reduced size and cost and can only store portions of the total wavetable data at a time. Instead, all the wavetable data is stored in system memory of the host PC and transferred in portions to the cache memory, as needed by the PC audio circuit. The PC audio circuit processes the data and generates digital audio signals, such as music or sound effects. Because the cache memory is of reduced size and cost, the PC audio circuit has a lower overall cost than prior art systems.

[0016] Unlike prior art PC audio systems, the PC audio circuit processes several frames of data samples for a voice before processing the next designated voice. Thus, several wavetable data samples for a given voice can be retrieved from system memory at one time and made available in the cache memory, thereby reducing the total number of accesses to memory required and the percentage use of system bus bandwidth. Processing the data samples in this manner also allows for certain parallel processing operations. For example, while a plurality of data samples are being processed for active voices, other groups of data samples can be retrieved from system memory and made available for processing in the cache memory. This ensures a continuous supply of data and reduces concerns about the maximum allowable system bus access latency.

[0017] Since the PC audio circuit retrieves several wavetable data samples at once, it is preferable that a voice's data samples be organized together in a block in system memory. Thus, if a consecutive series of data samples are requested, they can be accessed using the system memory's page mode which will increment through the data samples in the block. Preferably, the bus between system memory and the PC audio circuit is a PCI bus, thereby enabling data accessed through the page mode to be transmitted to the PC audio circuit in burst mode.

[0018] In the preferred embodiment of the present invention, the PC audio circuit includes a PCI bus interface block, an internal address data bus, digital signal processor, output control state machine, internal bus arbiter, and cache memory. The PC audio circuit can be formed on a monolithic integrated circuit, which includes the cache memory or with the cache memory external to the integrated circuit. Data in the system memory is transmitted over the PCI bus, through the PCI interface block, over the internal bus, and into the cache memory.

[0019] The digital signal processor (DSP) performs computations and other processing to translate the data samples in the cache memory into digital audio signals suitable for conversion into desired analog audio signals. Preferably, the DSP can generate up to 32 independent digital audio signals or voices at a 44.1 KHz frame rate.

[0020] The digital audio signals generated for each voice by the DSP are accumulated in the cache memory, or can be accumulated in a separate cache memory, until they are ready to be output to an external digital-to-analog converter (DAC). The output control state machine (OCSM) controls the transmission of the accumulated data from the cache out to the external DAC at a sample rate of 44.1 KHz. The internal bus arbiter (IBA) is responsible for directing traffic between the various blocks that will access the internal bus, including the OCSM, the cache, the PCI interface block, and the DSP. The internal bus operates at 33 MHz, along with most of the logic, from a clock that is provided as part of the PCI standard.

[0021] The cache preferably is a low-cost SRAM having a capacity of about 8 to 32 kilobytes. The available memory in the cache can be assigned to data sample storage, accumulator storage, and general storage for the DSP. Data samples can be stored in data queues A and B, while the digital audio signals generated by the DSP can be stored in accumulator queues A and B. In a suitable embodiment, data queues A and B each store up to 64 16-bit data samples for each of 32 voices, while accumulator queues A and B each accumulate the generated data samples for up to 32 voices. The generated data samples are accumulated together in accumulator queue A or B as one set of 64 16-bit data samples.

[0022] The PCI interface block detects when there is a need to update the cache with data samples and initiates bus master requests. The addresses in system memory from which the data samples are to be retrieved are sent from the PCI interface block to the PCI address bus. Under the control of the PCI interface block, data samples retrieved from system memory are transmitted on the internal data bus to the cache.

[0023] At start-up of the preferred PC audio circuit, 128 data samples are loaded into the cache (64 data samples in each of data queues A and B) for each active voice. Once data queues A and B are loaded with data, the DSP processes the data samples in one of the data queues, for the first active voice. The other data queue is presently inactive. Then, the DSP processes the data samples for the next designated active voice. As the DSP processes these data samples, the data samples just generated by the DSP are accumulated in one of the accumulator queues. This process continues until all active voices have been processed, and then the accumulator queues toggle and the other accumulator queue will accumulate generated data samples while the accumulated data samples in the first accumulator queue can be output to an external DAC.

[0024] Once the data samples for each active voice in the data queue are processed, the PCI interface block sends requests on the PCI bus for additional data samples from system memory. The data samples retrieved from system memory are stored in the first data queue, thereby writing over the data samples just processed. While these data samples are being retrieved, the DSP processes the data samples in the other queue. Then, the data queues toggle, and the process continues, allowing up to 64 data samples to be processed at a time.

[0025] If the DSP processes the data samples at the same frequency as the sampling frequency used during analog-to-digital conversion (recording) of the original audio signal, then when the audio signals generated by the DSP are converted to analog and played, the resulting audio signal will sound the same (i.e., have the same frequency) as the original audio signal used to create the data samples. When the frequency of the audio signal being played is the same as the recording frequency, its frequency ratio (F_c) equals 1. If $F_c > 1$, then generated audio signals will have a higher pitch than the signal recorded. If $F_c = 1$ for each of the active voices, then the maximum allowable PCI bus latency equals the time it takes to process 64 frames of data samples at the 44.1 KHz frame rate. However, if $F_c > 1$ for one or more active voices, the maximum allowable PCI bus latency is reduced because the DSP processes more than one data sample per frame per voice. For F_c equal to about 2.0, the reduction in the maximum allowable bus latency may become a problem.

[0026] The latency problem for $F_c > 2$ can be avoided by having the PC audio circuit retrieve only the data samples which will be processed and not the data samples which will be skipped by the DSP. Thus, all the data samples retrieved and stored in a data queue will be processed. This feature is implemented by providing means in PCI interface block for accessing the F_c values for the active voices, and then calculating the next system memory address for retrieving data samples for a given voice based on the current system memory address and the F_c value. Retrieving only select samples for each active voice when $F_c > 1$ reduces the available PCI bandwidth since the burst mode cannot be used for transmitting the data samples. Even if the burst mode is not used, the PC audio circuit's percentage usage of the bandwidth may be acceptable, but the percentage usage will be less desirable.

[0027] In the preferred embodiment of the present invention, the PC audio system includes driver software which facilitates the creation of frequency compensated files or patches of wavetable data which are stored in system memory and can be transmitted to cache memory in burst mode, thereby reducing the PCI bus bandwidth requirements. The frequency compensated files or patches contain only the data samples which will be actually processed by the DSP for a voice having $F_c > 2$. For example, for an active voice having $F_c = 4$, the DSP only needs to process every fourth data sample in the patch (the "original patch") of wavetable data associated with this active voice. The driver software

facilitates the creation of a frequency compensated version of the original patch, containing only every fourth sample. This frequency compensated file or patch is stored in system memory and can be transmitted in burst mode to the PC audio circuit for processing by the DSP.

[0028] A suitable PC audio system includes a PC audio circuit, of the type described above, driver software, and a MIDI or a comparable file. The MIDI file contains parameters that define the song or other audio signals to be generated by the PC audio circuit. The driver software performs the function of interpreting the parameters contained in the file and programming the PC audio circuit to generate the desired audio signals from wavetable data in system memory. As discussed below, the driver software also contains instructions which control the function of deriving frequency compensated patches for high F_c voices.

[0029] The system CPU determines for a given voice the ratio of the desired frequency for the voice to the recording frequency of the data in system memory associated with the voice. For $F_c > 2.0$, the CPU drives a frequency compensated patch as described below.

[0030] A frequency compensated patch can be derived a number of ways. One way, which requires the least CPU processing, is to copy or transpose a fraction of the wavetable data samples from the original patch for the voice into a new file or patch stored in system memory. The fraction of data samples transposed to the new patch is based on the F_c value calculated by the system CPU. The frequency compensated patch has a frequency that is higher than the frequency of the original patch. For example, if every fourth data sample is copied from the original patch to create the frequency compensated patch, the frequency compensated patch has a frequency which is four times the frequency of the original patch; the frequency compensated patch has an effective frequency (F_{eff}) equal to four.

[0031] Since bus latency problems occur when the PC audio circuit generates audio signals at more than twice the recording frequency of a patch of data samples, the goal when deriving frequency compensated patches is to provide an effective patch frequency which is high enough that the PC audio circuit does not have to more than double the patch's frequency to generate the desired audio signals.

[0032] Another way of deriving frequency compensated patches is to digitally filter the original patch of wavetable data samples. Digitally filtering requires more CPU processing but is preferred over the above technique since digitally filtering removes the high frequency component of wavetable data such that the generated digital audio signals have less noise. One example of a digital filtering technique is to take the average of every n th sample. Another example, which requires even more CPU processing power, is to calculate the average of the moving average.

[0033] The system CPU can be directed by the driver software to either: (1) derive all of the required frequency compensated patches just prior to the PC audio circuit's processing of the MIDI file; or (2) derive each patch as the PC audio circuit processes through the file.

[0034] When the PC audio circuit processes a frequency compensated patch, adjustments must be made to account for the higher frequency of the patch. The driver software programs the PC audio circuit to make these adjustments. Thus, if the PC audio circuit is originally programmed to generate digital audio signals at $F_c = 8$, but the PC audio circuit processes a frequency compensated patch with $F_{eff} = 4$, the PC audio circuit is then programmed to divide the frequency ratio for processing the data by four such that the data is processed with $F_c = 2$. Since it is easier to divide by a factor of two in digital circuitry, the frequency compensated files preferably should have an effective frequency which is a factor of two.

Brief Description of the Drawings

[0035] A better understanding of the present invention can be obtained when the following detailed description of the preferred and alternative embodiments is considered in conjunction with the following drawings, in which:

Fig. 1 is a block diagram of the PC audio circuit of the present invention as interfaced with the system bus of a host computer;

Fig. 2 depicts how memory can be assigned in the cache memory of the present invention;

Fig. 3 is a block diagram of the PCI bus interface block of the present invention as interfaced with system and internal buses;

Fig. 4 is a block diagram of an output control state machine of the present invention as interfaced with internal buses; and

Fig. 5 is a block diagram of a PC audio system which provides frequency compensated wavetable data in accordance with the present invention.

Detailed Description

I. PC AUDIO CIRCUIT OVERVIEW

[0036] The following description sets forth the preferred and alternative embodiments of a PC audio circuit which can be formed on a monolithic integrated circuit. The PC audio circuit is designed to interface with and provide audio enhancement to a host personal computer of the type including a central processor, system memory and system bus. The fundamental difference between the PC audio circuit of the present invention and prior art PC audio circuits is that the local memory is of a significantly reduced size (e.g., 8-32 kilobytes) and can only store portions of the total wavetable data at a time. Instead, all the wavetable data (e.g., 1-4 megabytes) is stored in system memory of the host PC and transferred in portions to the PC audio circuit's local memory, also known as a cache memory, as needed by the PC audio circuit. The PC audio circuit uses the data to generate digital audio signals such as music or sound effects.

[0037] As discussed in the Background of the Invention, if system memory is utilized to store wavetable data, thereby reducing the size of the local memory, the overall cost of the PC audio circuit will be reduced. However, the use of system memory raises concerns that: (i) an unacceptable percentage of system bus bandwidth will be used; and (ii) the PC audio circuit will process wavetable data faster than the host computer's maximum bus latency. The PC audio circuit of the present invention is designed to alleviate these concerns.

[0038] The typical frame rate for audio is 44.1 KHz. At this frame rate, each frame is approximately 22.7 microseconds. Thus, if a prior art PC audio circuit generates 32 voices during a frame, 32 data accesses must be made to memory during this short time period. This is not a problem if the data accesses are to local memory. If the number of accesses is made to system memory, however, bus bandwidth usage and bus latency would become a concern.

[0039] Unlike prior art systems, the PC audio circuit of the present invention processes several frames of data samples for a voice before processing the next designated voice. Thus, several wavetable data samples for a given voice can be retrieved from system memory at one time and made available in the cache memory, thereby reducing the total number of accesses to memory required and the percentage use of bus bandwidth. Processing the data samples in this manner also allows for certain parallel processing operations. For example, while a plurality of data samples are being processed for active voices, other groups of data samples can be retrieved from system memory and made available for processing in the PC audio circuit's cache memory. This ensures a continuous supply of data and reduces concerns about the bus access latency.

[0040] Since the PC audio circuit of the present invention retrieves several wavetable data samples at once, it is preferable that a voice's data samples be organized together in a block in the system memory. Thus, if a consecutive series of data samples are requested, they can be accessed using the system memory's page mode which will increment through the data samples in the block. If the bus between system memory and the PC audio circuit is a PCI bus (i.e., a higher performance bus), data accessed through the page mode can be transmitted to the PC audio circuit in burst mode (i.e., at a faster rate). Use of the burst mode decreases the maximum bus latency and the percentage of bandwidth usage.

II. PC AUDIO CIRCUIT ARCHITECTURE

[0041] Figure 1 illustrates the preferred architecture of the PC audio circuit. As illustrated, the PC audio circuit 10 includes a PCI bus interface block 12, internal address data bus 14, digital signal processor 16, output control state machine 18, internal bus arbiter 20, and cache memory 22. Data is passed from the PCI bus 24, through PCI interface block 12, over the internal bus 14, and into the cache 22. The PC audio circuit 10, including cache 22, can be formed on a monolithic integrated circuit. The dashed box in Figure 1 represents the perimeter of a preferred embodiment of such an integrated circuit. Alternatively, the cache 22 may be external to the circuit.

[0042] The digital signal processor (DSP) 16 operates on the data similarly to the wavetable synthesizer DSP disclosed in U.S. patent application Serial No. 08/334,461, by Norris, et al., which is incorporated herein by reference. In other words, the DSP 16 of the present invention performs computations and other processing to translate raw wavetable data into digital audio signals suitable for conversion into the desired analog audio signals. The DSP 16 operates from instructions stored in ROM code 26 and preferably can generate up to 32 independent digital audio signals or voices at a 44.1 KHz frame rate. Unlike the wavetable synthesizer disclosed in the above-referenced patent application, however, the DSP 16 of the present invention processes several frames of wavetable data samples from voice to voice, rather than one data sample per voice per frame. The implementation details for DSP 16 are within the level of skill possessed by those of ordinary skill in the art.

[0043] The digital audio signals generated for each voice by DSP 16 are accumulated in cache 22, or can be accumulated in a separate cache memory, until they are ready to be output on port 28 to an external audio digital-to-analog converter (DAC). The output control state machine (OCSM) 18 is responsible for transmitting the accumulated data from the cache 22 out to the external DAC at the sample rate of 44.1 KHz. OCSM 18 utilizes its own 16.9344 MHz

EP 0 882 286 B1

clock 30 to ensure synchronization with to the sample rate. The internal bus arbiter (IBA) 20 is responsible for directing traffic between the various blocks that will access the internal bus 14, including the OCSM 18, the cache 22, the PCI interface block (PCI I/F block) 12 and the DSP 16. The internal bus 14 operates at 33 MHz, along with most of the logic, from a clock 32 that is provided as part of the PCI standard.

[0044] Internal Bus. The internal bus 14 has a 32-bit data bus and a 16-bit address bus. The address map for the internal bus is as follows:

Address Range (hexadecimal)	Data
0000 through 1FFF	8Kx32 SRAM cache. This space provides the port into the cache memory.
2000 through 201F	32 PCI I/F-block voice cache status registers. There are 32 of these registers, one to correspond to each of the 32 possible voices. Bit[0] of these registers is set (by the DSP) when that voice needs cache queue A updated with data from the PCI bus. (See discussion below regarding cache.) Bit[1] of these registers is set (by the DSP) when that voice needs cache queue B updated with data from the PCI bus. After the PCI interface block has successfully updated the data in the cache queue for a voice, then it clears the bit. Bit[2] is high to indicate that the voice is active and low to indicate that the voice is not active. When bit[2] goes low, the current system address register is reset to become the same as the system address start register (see discussion below).
2020 through 203F	32 PCI I/F-block system address start registers. These are 32-bit pointers to each of the 32 voices' system memory start addresses for the sample. When processing of a voice starts, data is initially brought in starting from this location in system memory.
2040 through 205F	32 PCI I/F-block system address end registers. These are 32-bit pointers to each of the 32 voices' system memory end addresses for the sample.
2060 through 207F	32 PCI I/F-block system address loop point registers. These are 32-bit pointers to each of the 32 voices' system memory loop point addresses for the sample. As data for a voice is brought into the PC audio circuit from the PCI bus, if the address crosses over the end address, then it jumps back to the address specified by these registers.
2080 through 209F	32 PCI I/F-block current system address registers. These registers store the current address in system memory from which the sample data for each of the 32 voices is accessed. They increment whenever a new 32-bit word is brought in from system memory to the cache. They jump from the system address end register location to the system address loop point location when the current address passes the end point.
3000	OCSM sample count register. This is a 7 bit counter that increments from its starting point, zero, whenever accumulated data is output from the cache and sent to the external DAC. The DSP can observe bit 7 of this counter to determine when it is time to start accumulating the next group of 64 samples.
3001	OCSM control register. When bit[0] of this register is cleared (by the DSP) then no data is passed out to the DAC. When it is high, then data is drawn from the accumulator cache and passed to the external DAC.

[0045] PCI Bus. The PCI bus is assigned a block of 256 I/O (byte wide) addresses through standard PCI plug and play circuitry. These addresses are used by the system's central processor as follows:

I/O Address Range (hexadecimal)	Data
80 through 81	Internal bus address register. The system CPU is allowed access to the internal bus by setting up the 16-bit address in these two ports and writing or reading through the data ports below.

(continued)

I/O Address Range (hexadecimal)	Data
84 through 87	Internal data bus port. Access (read and write) to the internal bus is allowed via this port with the internal address specified by 80-81 above.

[0046] Wavetable Cache RAM. Cache 22 preferably is a low-cost SRAM having a capacity of about 8 to 32 kilobytes. The available memory in cache 22 can be assigned to data sample storage, accumulator storage, and general storage for DSP 16. Figure 2 depicts how the available memory can be suitably assigned between data sample storage and accumulator storage. Data samples can be stored in data queues "A" and "B", while the digital audio signals generated by DSP 16 (generated data samples) can be stored in accumulator queues "A" and "B." See Figure 2. Data queues A and B can each store up to 64 16-bit data samples for each of 32 voices. Accumulator queues A and B each can accumulate the generated data samples for up to 32 voices. The generated data samples are accumulated together in queue A or B as one set of 16-bit data samples. There can be up to 64 data samples in a set.

[0047] Data queues A and B together can store up to 8 kilobytes, while accumulator queues A and B together can store up to 256 bytes. Additional memory can be provided in cache 22 for general DSP storage.

[0048] Preferably, one of the two data queues for a voice is used to store data samples as they are retrieved from system memory while the other data queue supplies data samples to the DSP 16. Thus, if data queue A supplies data samples to DSP 16, then data queue B stores data samples retrieved from system memory. Data queue B is filled with the next set of data samples to be processed by DSP 16, and must be filled before the DSP completes the processing of the data samples in data queue A. Otherwise, there will be undesirable gaps in the generated digital audio signals. When all the data samples in data queue A have been processed, data queues A and B are toggled, and DSP 16 processes the data samples stored in data queue B, and data samples retrieved from system memory are stored in data queue A. This process continues as long as DSP 16 processes data samples.

[0049] Similarly, one of the accumulator queues is used to supply accumulated data samples to an external DAC, while the other accumulator accumulates data samples generated by DSP 16. Thus, if accumulator queue A supplies accumulated data samples, then accumulator queue B accumulates data samples. The generated data samples for all of the active voices must be accumulated in accumulator queue B before all the data samples in accumulator queue A have been transmitted to the external DAC. Otherwise, there will be gaps in the analog signal. When all the data samples in accumulator queue A have been transmitted to the external DAC, accumulator queues A and B toggle, and data samples accumulated in data queue B are transmitted to the external DAC, and generated data samples are accumulated in data queue A. The overall operation of the present invention is further discussed below.

[0050] An address map for a wavetable cache suitable for the present invention is as follows:

Address Range (hexadecimal)	Data
0000 through 001F	Cache queue "A" for voice 0 of 31
0020 through 003F	Cache queue "B" for voice 0 of 31
0040 through 005F	Cache queue "A" for voice 1 of 31
0060 through 007F	Cache queue "B" for voice 1 of 31
0080 through 07BF	Caches for voices 2 through 30
07C0 through 07DF	Cache queue "A" for voice 31 of 31
07E0 through 07FF	Cache queue "B" for voice 31 of 31
0800 through 083F	Accumulator cache "A"
0840 through 087F	Accumulator cache "B"
0880 through 1FFF	General Storage for the DSP

[0051] The PCI Interface. Figure 3 sets forth a block diagram of PCI interface block 12. PCI interface block 12 includes PCI interface controller 34, buffers 36 and 38, internal bus address register 40, and PCI I/F block registers 42. As illustrated, PCI controller 34 is connected to the PCI address bus, buffers 36 and 38, internal bus address register 40, and PCI I/F block registers 42. Buffer 36 connects to the PCI data bus, PCI controller 34, internal bus address register 40, and the internal data bus, while buffer 38 connects to the PCI address bus, PCI controller 34, and PCI I/F block registers 42. Internal bus address register 40 connect to PCI controller 34, the internal address bus, the

internal data bus, and buffer 36. Finally, PCI I/F block registers 42 are connected to buffer 38, PCI controller 34, and the internal data and address buses.

[0052] PCI I/F block registers 42 contain status and address information which indicates which voice requires additional data samples to be stored in cache and the address in system memory to obtain the data samples. A detailed description of these registers is set forth in the above address map for the internal bus. Internal bus address register 40 is used by the system central processor to access the PC audio circuit registers on the internal bus. For example, the central processor may need access to the PCI I/F block registers in order to write system memory addresses which indicate wavetable data storage locations. Internal bus address register 40 also stores the addresses of cache 22 at which wavetable data samples from system memory are stored. As set forth in the I/O address table above, the central processor accesses a register on the internal bus by writing its address, via buffer 36, in the internal bus address register 40. Read or write accesses to a particular register is provided through the port specified in the above table.

[0053] Based on the status information stored in PCI I/W block registers 42, PCI interface controller 34 detects when there is a need to update cache 22 with data samples and initiates bus master requests. Under the control of PCI interface controller 34, the addresses in system memory from which data samples are to be retrieved are sent from the PCI interface block registers 42, through buffer 38, to the PCI address bus. Retrieved data samples from system memory are sent on the PCI data bus to buffer 36. Under the control of PCI interface controller 34, data samples in buffer 36 are transmitted on the internal data bus to cache 22. The addresses in cache 22 for storing the data samples are contained in internal bus address register 40 and transmitted on the internal address bus. Preferably, PCI interface block 12 can request data samples for more than one active voice at a time.

[0054] PCI controller 34 calculates the cache addresses for storing the data samples by determining which voice is being updated, whether queue A or B is being updated, and which 32-bit word of the queue is being updated. PCI controller 34 contains thirty-two 5-bit counters-one for each voice-to determine which sample in the queue is the next to be updated by the PCI interface block 12. PCI interface block registers 42 include thirty-two 1-bit toggle registers-one for each voice-to indicate which queue each voice is currently using. These registers toggle each time a queue is filled by the PCI interface block 12. The PCI controller 34 stores the calculated cache addresses in the internal bus address register 40 and controls when they are output onto the internal address bus. The implementation details of PCI interface block 12 are within the level of skill possessed by those of ordinary skill in the art.

[0055] **The Output Control State Machine.** Figure 4 sets forth a block diagram of OCSM 18. As illustrated, OCSM 18 includes a control block 46, with buffer, address decode and control block 48, 7-bit counter 50, and 22.66 micro-second timer 54. As illustrated, control block 46 is connected to the internal address and data buses, the address decode and control block 48, the 7-bit counter 50, and FIFO 52. Address decode and control block 48 is connected to the internal address bus, control block 46, 7-bit counter 50, and timer 54. Seven-bit counter 50 is connected to address decode and control block 48, control block 46, and FIFO 52. The seven-bit counter is described in the above address map for the internal bus, and is referred to as the OCSM sample count register. FIFO 52 can store two data samples, one in a top location and the other in a bottom location, and is connected to control block 46, 7-bit counter 50, timer 54, and an external DAC. Timer 54 connects to address decode and control block 48, FIFO 52, and clock generator 30.

[0056] The DSP 16 enables OCSM 18 by writing to its control register. Once OCSM 18 is enabled, two data samples are transmitted, under the control of control block 46, on the internal data bus from an accumulator queue in cache 22, through the buffer in the control block 46, into FIFO 52. Every 22.66 microseconds, as indicated by timer 54, the FIFO 52 shifts the data sample in the bottom location to the top location, thereby enabling it to be output to the external DAC. The data sample previously in the top location is discarded. At the same time, another data sample is retrieved from cache 22 and stored in the bottom location of FIFO 52, and, under the control of address decode and control block 48, the 7-bit counter 50 is incremented. Address decode and control block 48 calculates the addresses of data samples to be retrieved from cache 22 from the 7-bit counter 50 and cache address information supplied on the internal address bus. These calculated addresses are sent to control block 46 where they are used to request specific data samples from cache 22. The implementation details of OCSM 18 are within the level of skill possessed by those of ordinary skill in the art. DSP 16 can observe bit 7 of counter 50 to determine when it is time to start accumulating the next group of data samples.

[0057] **The Internal Bus Arbiter.** Internal bus arbiter 20 is a simple arbiter that has a fixed priority for bus requests from: (i) the DSP 16 (lowest priority); (ii) the PCI bus interface block 12 (middle priority); and (iii) the OCSM 18 (highest priority). Arbiter 20 grants bus access to the requesting device having highest priority, at which point that device is free to drive the address bus and either the READ or WRITE signal. If the access is a read, then the priority device will capture or use the data from the data bus; if the access is a write, then the priority device will drive the data bus. The implementation details of arbiter 20 are within the level of skill possessed by those of ordinary skill in the art.

III. PC AUDIO SYSTEM OPERATION

[0058] At start-up of the preferred embodiment of PC audio circuit 10, 128 data samples are loaded into the cache

22 (64 data samples in each of data queues A and B) for each active voice. Assuming 32 active voices, this makes the worst case for required memory:

(32 voices) (128 samples/voice) (2 bytes/sample) = 8 kilobytes

Additionally, cache 22 requires accumulator queues A and B having capacity of 64 data samples each. Thus, the additional required memory is:

(2 queues) (64 samples/queue) (2 bytes/sample) = 256 bytes

As discussed above, additional memory may be provided for general DSP operations.

[0059] Once data queues A and B are loaded with data, the DSP 16 processes the data samples in one of the data queues, for the first active voice (e.g., voice 0). The other data queue is presently inactive. Then, DSP 16 processes the data samples for the next designated active voice (e.g., voice 1). As DSP 16 processes these data samples, the data samples just generated by DSP 16 are accumulated in one of the accumulator queues. This process continues until all active voices have been processed, and then the accumulator queues toggle and the other accumulator queue will accumulate generated data samples while the accumulated data samples in the first accumulator queue can be output to the external DAC.

[0060] Also, once the data samples for each active voice in the first data queue are processed, the PCI interface block 12 sends requests on the PCI bus for additional data samples from system memory. The data samples retrieved from system memory are stored in the first data queue, thereby writing over the data samples just processed. While these additional data samples are being retrieved, the DSP 16 processes the data samples in the other queue. Then, the data queues toggle, and the process continues, allowing up to 64 data samples to be processed at a time.

[0061] If DSP 16 processes the data samples at the same frequency as the sampling frequency used during analog-to-digital conversion (recording) of the original audio signal, then when the audio signals generated by the DSP are converted to analog and played, the resulting audio signal will sound the same (i.e., have the same frequency) as the original audio signal used to create the data samples. When the frequency of the audio signal being played is the same as the recording frequency, its frequency ratio (F_c) equals 1. For example, if a middle C note (middle C = 440 Hz) of a piano is recorded and $F_c = 1$, then the audio signal generated or played will be at the same frequency and sound the same as the signal recorded. If $F_c > 1$, the generated audio signal will have a higher pitch. For $F_c = 4$, the generated audio signal is two octaves higher than the sampling frequency of the signal recorded.

[0062] If $F_c = 1$ for each of the active voices, then the maximum allowable PCI bus latency equals the time it takes to process 64 frames of data samples at the 44.1 KHz frame rate.

64 frames x 1/44100 seconds = 1.45 milliseconds

However, if $F_c > 1$ for one or more active voices, the maximum allowable PCI bus latency is reduced because DSP 16 processes more than one data sample per frame per voice. In other words, the data samples in a data queue for a particular voice are consumed faster than if $F_c = 1$. For example, for $F_c = 2$, DSP 16 skips every other data sample in the data queue. For F_c greater than about 2.0, the reduction in the maximum allowable bus latency may become a problem.

[0063] The latency problem for $F_c > 2$ can be avoided by having the PC audio circuit 10 retrieve only the data samples which will be processed and not the data samples which will be skipped by DSP 16. Thus, all the data samples retrieved and stored in a data queue of cache 22 will be processed. This feature is implemented by providing means in PCI interface block 12 for accessing the F_c values for the active voices, and then calculating the next system memory address for retrieving data for a given voice based on the current system memory address and the F_c value. For example, if $F_c = 4$ for a given active voice, then: next system memory address = current address + 4. The implementation details for this feature are within the level of skill possessed by those of ordinary skill in the art.

[0064] Retrieving select data samples for each active voice when $F_c > 1$ reduces the available PCI bandwidth since the burst mode cannot be used for transmitting the data samples. However, the percentage usage of the bandwidth is generally expected to be acceptable even without use of the burst mode.

[0065] When the PCI bus is in burst mode, it typically operates at 60 ns/32 bits and the required bandwidth is: $\frac{32 \text{ bits}}{60 \text{ ns}} = 4.2\%$ (Note: 2 voices = 32 bits.) A bandwidth usage of 4.2% is very acceptable. If the PCI bus is not in burst mode, it typically operates four times slower, and the bandwidth usage is 17%. A bandwidth usage of 17% may be acceptable but is less desirable and increases the risk that an excessive amount of the PCI bus bandwidth will be used.

IV. REDUCING PCI BUS BANDWIDTH REQUIREMENTS BY PROVIDING FREQUENCY COMPENSATED WAVETABLE DATA

[0066] As discussed above, when $F_c > 2$ for one or more active voices, bus latency problems may arise because DSP 16 skips data samples and consumes data samples faster than if $F_c = 1$. The bus latency problems can be avoided by having the PC audio circuit 10 retrieve only the data samples which will be processed and not the data samples which will be skipped. However, when only select data samples are retrieved from system memory, the burst mode

cannot be used and this increases the percentage of PCI bus bandwidth usage. Therefore, in the preferred embodiment of the present invention, the PC audio system includes driver software which facilitates the creation of frequency compensated files or patches of wavetable data which are stored in system memory and can be transmitted to cache memory 22 in burst mode, thereby reducing the PCI bus bandwidth requirements. The frequency compensated files or patches contain only the data samples which will be actually processed by DSP 16 for a voice having $F_c > 2$. For example, for an active voice having $F_c = 4$, the DSP 16 only needs to process every fourth data sample in the patch (the "original patch") of wavetable data associated with this active voice. The driver software facilitates the creation of a frequency compensated version of the original patch, containing only every fourth sample. This frequency compensated file or patch is stored in system memory and can be transmitted in burst mode to PC audio circuit 10 for processing by DSP 16.

[0067] Figure 5 sets forth a block diagram of a PC audio system which can provide frequency compensated patches as described above. The PC audio system includes a PC audio circuit 10, of the type described above, driver software 62, and a MIDI or comparable file 64. PC audio circuit 10 is connected to system memory 60 through PCI bus 24. File 64 contains parameters that define the song or other audio signals to be generated by PC audio circuit 10. The driver software 62 performs the function of interpreting the parameters contained in file 64 and programming PC audio circuit 10 to generate the desired audio signals from wavetable data in system memory. As discussed below, the driver software 62 also contains instructions which control the function of providing frequency compensated patches for high F_c voices.

[0068] Upon execution of the instructions in driver software 62, the system CPU performs the following steps:

Step 1: Retrieve the parameters for a note or voice in file 64.

Step 2: Calculate the voice's desired F_c from these parameters and the frequency of wavetable data in system memory associated with the voice.

Step 3: If $F_c > 2.0$, derive a frequency compensated patch from the patch of wavetable data stored in system memory which is associated with the voice. The frequency compensated patch is stored in system memory.

In steps 1 and 2, the system CPU determines for a given voice in file 64 whether the voice is to be played at a frequency higher than the frequency of the wavetable data in system memory for that voice. The ratio of the desired frequency to the frequency of the data in system memory determines the F_c for that voice. The CPU then compares the F_c value to 2.0. For $F_c > 2.0$, the CPU derives a frequency compensated patch as described below. Steps 1-3 are repeated for each voice in file 64.

[0069] A frequency compensated patch can be derived a number of ways. One way, which requires the least CPU processing, is to copy or transpose a fraction of the wavetable data samples from the original patch for the voice into a new file or patch stored in system memory. The fraction of data samples transposed to the new patch is based on the F_c value calculated by the system CPU. The frequency compensated patch has a frequency that is higher than the frequency of the original patch. For example, if every fourth data sample is copied from the original patch to create the frequency compensated patch, the frequency compensated patch has a frequency which is four times the frequency of the original patch; the frequency compensated patch has an effective frequency (F_{eff}) equal to four.

[0070] Since bus latency problems occur when the PC audio circuit 10 generates audio signals at more than twice the recording frequency of a patch of data samples ($F_c > 2$), the goal when deriving frequency compensated patches is to provide an effective patch frequency which is high enough that the PC audio circuit 10 does not have to more than double the patch's frequency to generate the desired audio signals. For example, if $F_c = 8$, the system CPU should copy every fourth data sample into a new file or patch. The new file or patch is "frequency compensated" and has an effective frequency of four ($F_{eff} = 4$). Thus, the PC audio circuit 10 can process this frequency compensated patch at only twice the frequency to provide audio signals with $F_c = 8$. If the PC audio circuit 10 instead processed the original patch of wavetable data, the PC audio circuit would have to process the data at eight times the recording frequency ($F_c = 8$) to generate the desired audio signals, thereby creating bus latency problems.

[0071] Another way of deriving frequency compensated patches is to digitally filter the original patch of wavetable data samples. Digitally filtering requires more CPU processing but is preferred over the above technique of transposing a fraction of the wavetable data samples in a patch. In the above technique, the high frequency component of a patch of wavetable data translates into noise in a frequency compensated patch. Digitally filtering removes the high frequency component and thus results in a frequency compensated patch providing cleaner sound. One example of a digital filtering technique is to take the average of every n th sample. Thus, for a voice with $F_c = 8$, the CPU could take the average of every fourth sample to derive the frequency compensated patch. Another example of a digital filtering technique, which requires even more CPU processing power, is to calculate the average of the moving average.

[0072] The system CPU can be directed by the driver software 62 to either: (1) derive all of the required frequency

compensated patches just prior to the PC audio circuit's 10 processing of MIDI file 64; or (2) derive each patch as the PC audio circuit 10 processes through file 64. The former technique requires more memory since all the frequency compensated patches are derived and stored in system memory prior to processing by the PC audio circuit 10. The former technique also causes a delay before the MIDI file 64 can be played. The later technique is preferred if the CPU

5 has enough power and excess time to perform the necessary calculations as file 64 is processed.
[0073] When PC audio circuit 10 processes a frequency compensated patch, adjustments must be made to account for the higher frequency of the patch. For example, if $F_c = 8$ for a voice and $F_{eff} = 4$ for the frequency compensated patch, the PC audio circuit 10 only needs to process the patch at twice the frequency rather than eight times the frequency to generate digital audio signals having $F_c = 8$. The driver software 64 programs the PC audio circuit 10 to
 10 make these adjustments. Thus, if PC audio circuit 10 is originally programmed to generate digital audio signals at $F_c = 8$, but the PC audio circuit processes a frequency compensated patch with $F_{eff} = 4$, the PC audio circuit is then programmed to divide the frequency ratio for processing the data by four such that the data is processed with $F_c = 2$. Since it is easier to divide by a factor of two in digital circuitry, the frequency compensated files preferably should have an effective frequency which is a factor of two.

15 **[0074]** Although the above discussion focuses on the use of frequency compensated patches to reduce PCI bus bandwidth requirements, frequency compensated patches have application in PC audio systems which obtain wavetable data from local rather than system memory. As discussed above, digitally filtering patches of wavetable data filters out high frequencies which translate into noise when the patch is played at a frequency higher than the recording frequency (e.g., $F_c > 2$). Wavetable data, stored in a local memory and to be used by a wavetable synthesizer to
 20 generate digital audio signals with a high frequency ratio, can be preprocessed, through the above digital filtering techniques, such that the synthesizer generates clean digital audio signals.

[0075] The present invention, therefore, is well adapted to carry out the objects and attain the ends and advantages mentioned herein as well as other ends and advantages made apparent from the disclosure. While preferred embodiments of the invention have been described for the purpose of disclosure, numerous changes and modifications to
 25 those embodiments described herein will be readily apparent to those skilled in the art and are encompassed within the scope of the following claims.

Claims

- 30
1. A method of providing a frequency compensated version of a first patch of wavetable data having a first sample frequency and stored in a first location of a memory, wherein said frequency compensated patch is stored in said memory and either said first patch or said frequency compensated patch is accessed from said memory by a digital wavetable audio synthesizer and used to generate digital audio signals having a second sample frequency which
 35 is higher than said first sample frequency, said method comprising the steps of:
 - (a) accessing said first patch of wavetable data from said first location of said memory;
 - (b) deriving, from said first patch of wavetable data, a patch of wavetable data which has a third sample frequency greater than said first sample frequency, wherein said derived patch of wavetable data comprises
 40 said frequency compensated patch; and
 - (c) storing said frequency compensated patch in a second location of said memory for use by said digital audio synthesizer in generating digital audio signals having said second sample frequency.
 2. A method as claimed in claim 1, wherein said frequency compensated patch comprises a digitally filtered version
 45 of said first patch of wavetable data.
 3. A method as claimed in claim 1, wherein said frequency compensated patch comprises a fraction of data samples from said first patch of wavetable data.
 4. A method as claimed in claim 1, 2 or 3, further comprising the step of calculating a ratio of said second sample frequency to said first sample frequency, and wherein said frequency compensated patch is derived only if said
 50 ratio is greater than a predetermined value.
 5. A method as claimed in claim 4, wherein said predetermined value is greater than about 2.0.
 - 55 6. A method as claimed in claim 4, wherein the ratio of said second sample frequency to said third sample frequency of said frequency compensated patch is less than or equal to about 2.0.

Patentansprüche

1. Verfahren zum Erzeugen einer frequenzkompensierten Version eines ersten Patch von Wellendiagrammdaten, die eine erste Tastfrequenz aufweisen und an einer ersten Stelle eines Speichers gespeichert sind, wobei das frequenzkompensierte Patch in dem Speicher gespeichert wird und in dem Speicher entweder auf das erste Patch oder das frequenzkompensierte Patch durch einen Digitalwellendiagramm-Audio-Synthesizer zugegriffen wird und das Patch dazu verwendet wird, digitale Audio-Signale mit einer zweiten Tastfrequenz zu erzeugen, die höher ist als die erste Tastfrequenz, wobei das Verfahren die folgenden Schritte umfaßt:

(a) Zugreifen auf das erste Patch von Wellendiagrammdaten aus der ersten Stelle des Speichers;

(b) aus dem ersten Patch von Wellendiagrammdaten, Ableiten eines Patch von Wellendiagrammdaten, das eine dritte Tastfrequenz hat, die größer ist als die erste Tastfrequenz, wobei das abgeleitete Patch von Wellendiagrammdaten das frequenzkompensierte Patch aufweist; und

(c) Speichern des frequenzkompensierten Patch an einer zweiten Stelle des Speichers zur Verwendung durch den Digital-Audio-Synthesizer beim Erzeugen digitaler Audio-Signale, die die zweite Tastfrequenz aufweist.

2. Verfahren nach Anspruch 1, bei dem das frequenzkompensierte Patch eine digital gefilterte Version des ersten Patch von Wellendiagrammdaten aufweist.

3. Verfahren nach Anspruch 1, bei dem das frequenzkompensierte Patch einen Bruchteil von Daten-Abtastwerten aus dem ersten Patch von Wellendiagrammdaten aufweist.

4. Verfahren nach Anspruch 1, 2 oder 3, ferner mit dem Schritt des Berechnen des Verhältnisses der zweiten Tastfrequenz zu der ersten Tastfrequenz, wobei das frequenzkompensierte Patch nur dann abgeleitet wird, falls das Verhältnis größer als ein vorbestimmter Wert ist.

5. Verfahren nach Anspruch 4, bei dem der vorbestimmte Wert größer als ungefähr 2,0 ist.

6. Verfahren nach Anspruch 4, bei dem das Verhältnis der zweiten Tastfrequenz zu der dritten Tastfrequenz des frequenzkompensierten Patch kleiner als oder gleich ungefähr 2,0 ist.

Revendications

1. Procédé pour fournir une version compensée en fréquence d'une première correction de données de tableaux d'ondes présentant une première fréquence d'échantillonnage et enregistrée dans une première position de mémoire, dans lequel ladite première correction compensée en fréquence est stockée dans ladite mémoire et dans lequel on accède, soit à ladite première correction, soit à ladite correction compensée en fréquence à partir de ladite mémoire par un synthétiseur numérique audio de tableaux d'ondes et utilisé pour générer des signaux audio numériques présentant une seconde fréquence d'échantillonnage qui est plus élevée que ladite première fréquence d'échantillonnage, ledit procédé comprenant les étapes consistant à :

(a) accéder à ladite première correction de données de tableaux d'ondes à partir de ladite première position de ladite mémoire ;

(b) déduire, à partir de ladite première correction de données de tableaux d'ondes, une correction de données de tableaux d'ondes qui présente une troisième fréquence d'échantillonnage plus élevée que ladite première fréquence d'échantillonnage, dans lequel ladite correction déduite des données de tableaux d'ondes comprend ladite correction compensée en fréquence ; et

(c) enregistrer ladite correction compensée en fréquence dans une seconde position de ladite mémoire pour être utilisé par ledit synthétiseur audio numérique en générant des signaux audio numériques présentant ladite seconde fréquence d'échantillonnage.

2. Procédé selon la revendication 1, dans lequel ladite correction compensée en fréquence comprend une version filtrée numériquement de ladite première correction de données de tableaux d'ondes.

3. Procédé selon la revendication 1, dans lequel ladite correction compensée en fréquence comprend une fraction

d'échantillons de données venant de ladite première correction de données de tableaux d'onde.

4. Procédé selon la revendication 1, 2 ou 3, comprenant, de plus, l'étape consistant à calculer un rapport de ladite seconde fréquence d'échantillonnage à ladite première fréquence d'échantillonnage, et dans lequel ladite correction compensée en fréquence est déduite seulement si ledit rapport est supérieur à une valeur prédéterminée.
5. Procédé selon la revendication 4, dans lequel ladite valeur prédéterminée est supérieure à 2,0 environ.
6. Procédé selon la revendication 4, dans lequel le rapport de ladite seconde fréquence d'échantillonnage à ladite troisième fréquence d'échantillonnage de ladite correction compensée en fréquence est inférieur ou égal à 2,0 environ.

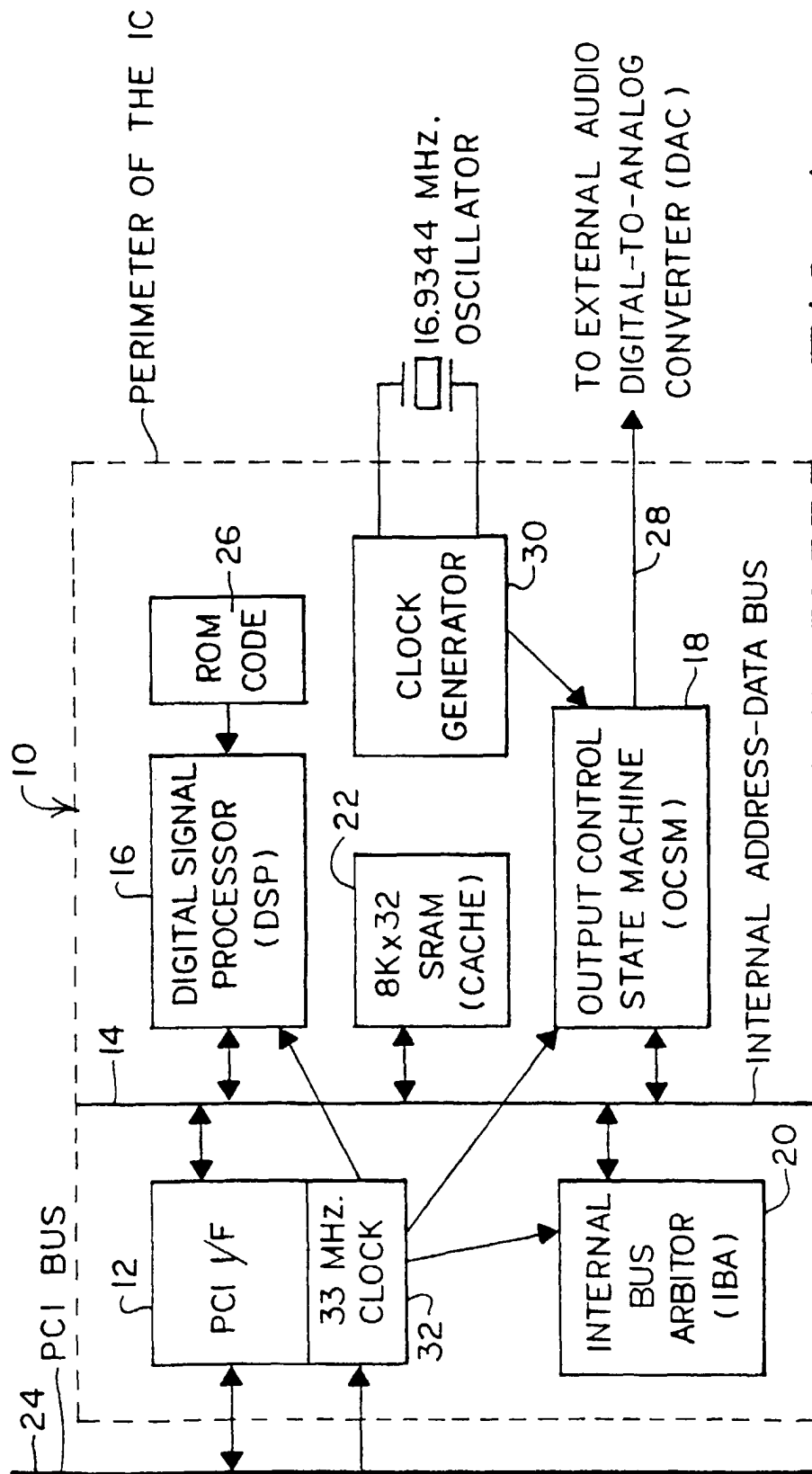


FIG. 1

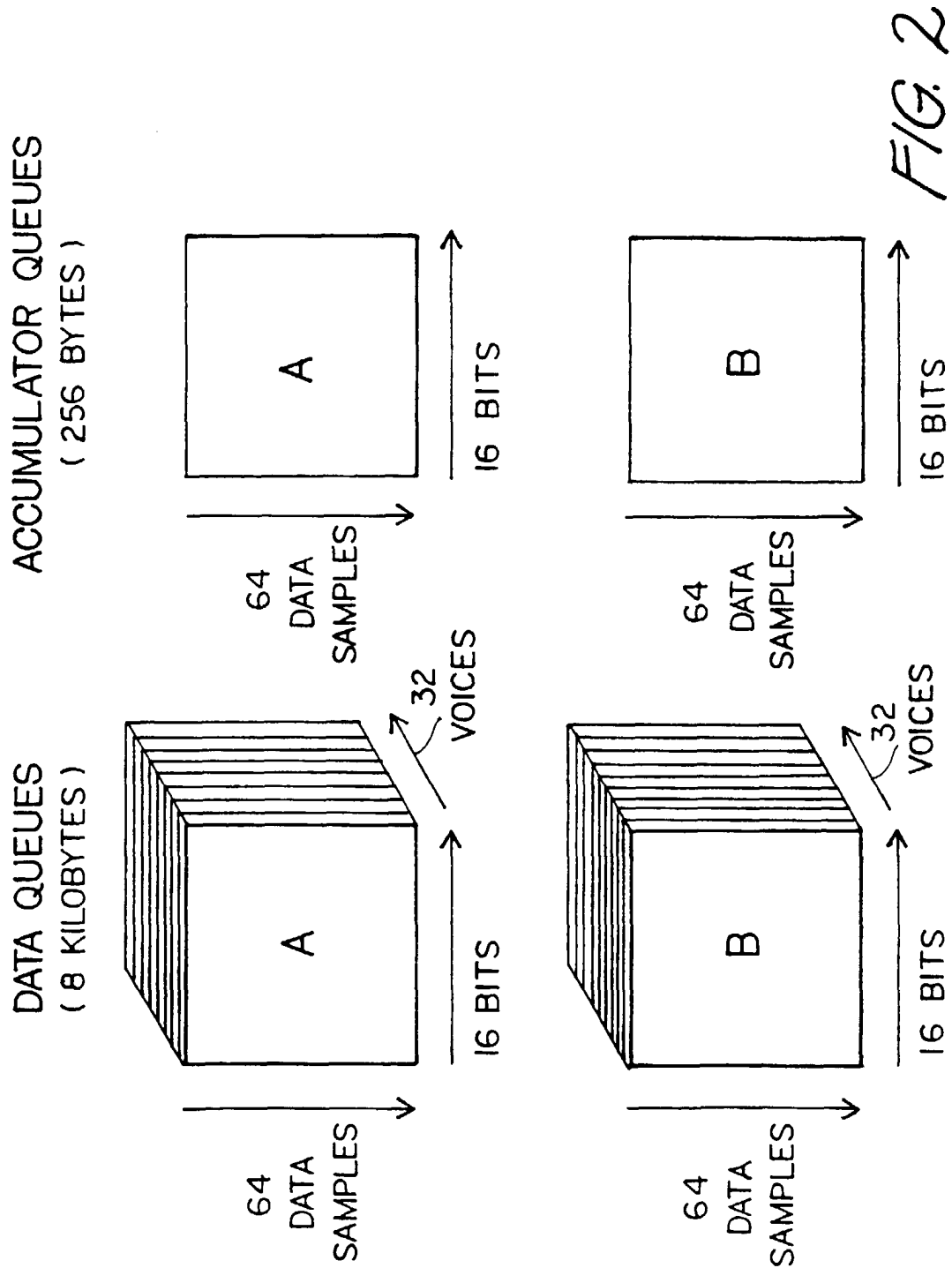


FIG. 2

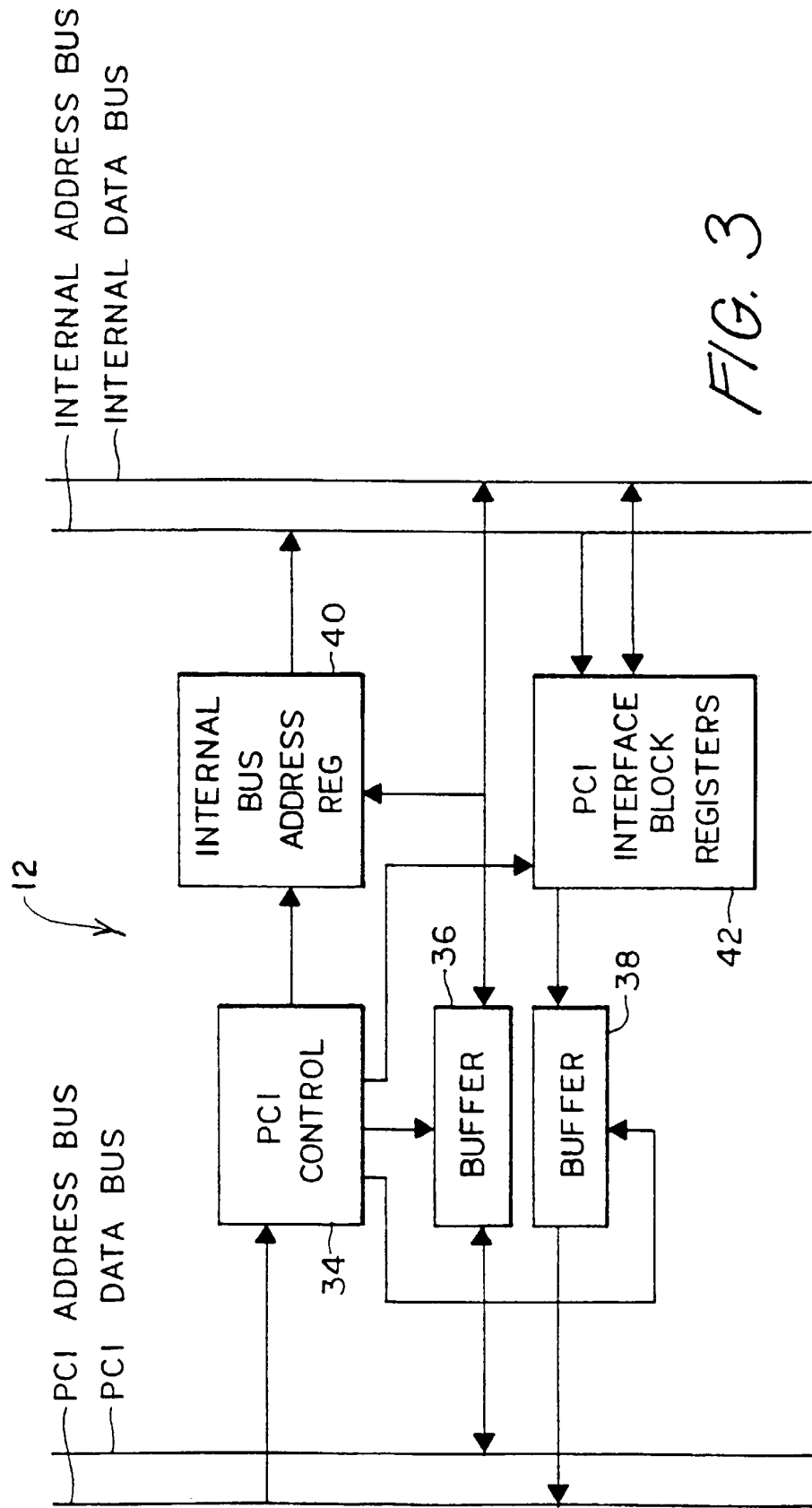


FIG. 3

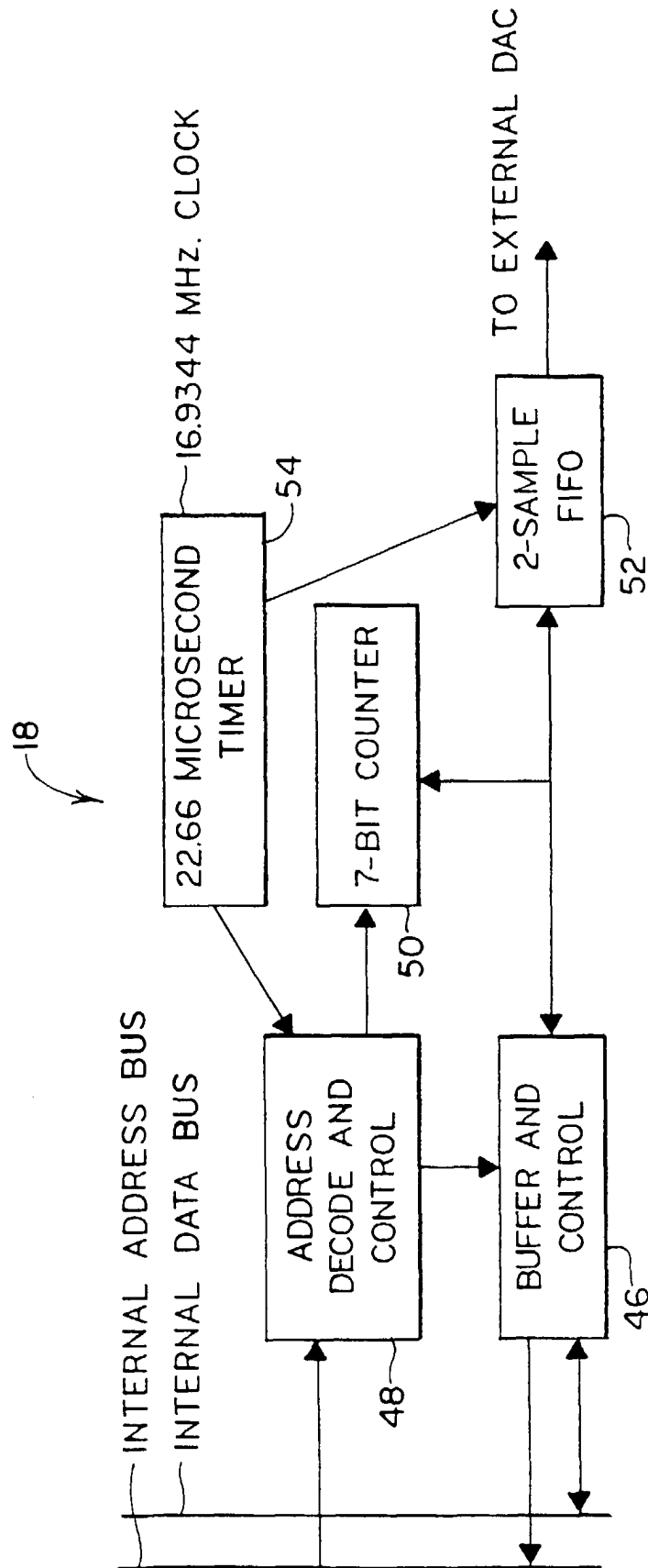


FIG. 4

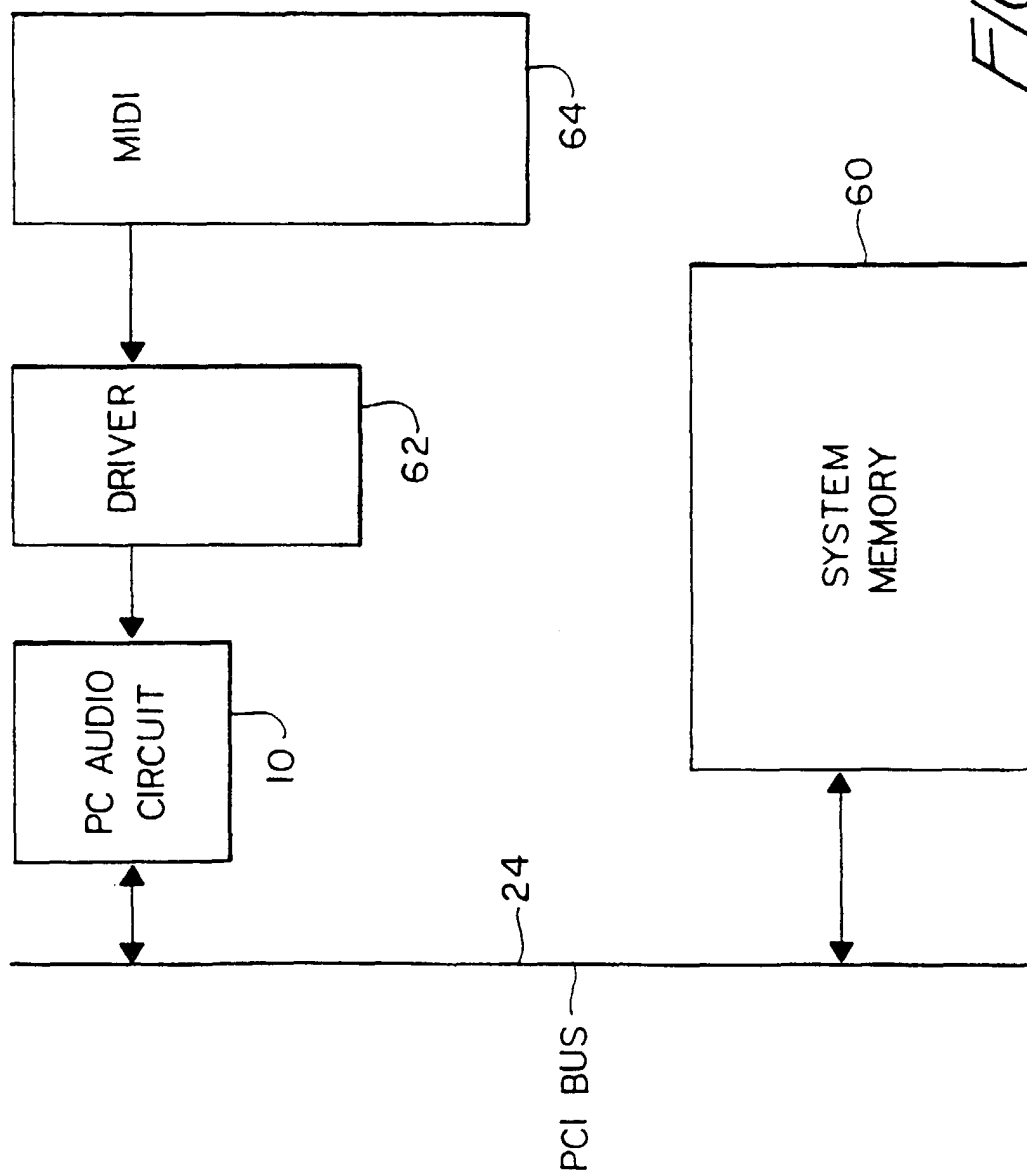


FIG. 5