Audio Support in HP MPower

Multimedia capability promises to enhance the communication and presentation of information through the use of real-world data types such as audio and video. Compact-disk-quality audio is the first of such data types to be offered as a standard feature on all of HP's new workstations.

by Ellen N. Brandt, Thomas G. Fincher, and Monish S. Shah

HP MPower provides the hardware and software to allow recording and playing of audio files over a network, incorporating audio in email, adding audio annotations to system files, and recording and playing to external devices like tape recorders, CD players, and VCRs. With these capabilities audio-enabled applications can add voice annotation to documents ranging from spreadsheet rows and columns to CAD drawings. Programmers might add audio comments to their programs. Error messages could take the form of spoken messages, or even distinctive sounds that convey more information than a simple beep. Finally, background music could be added to presentations.

This article describes HP MPower's audio functionality, application development tools, and audio hardware and software architecture.

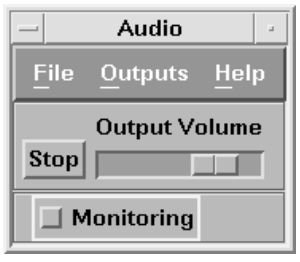
Audio Tools

The audio tools provided in HP MPower allow users to record, edit, and play audio data in a variety of file and data formats. HP MPower also provides tools for converting between audio formats. All of these tools are built on the audio library, which defines the application program interface to HP MPower's client/server audio implementation. The application program interface, libraries, widgets, and header files are available to third-party software developers who wish to use audio in their applications.

The Audio Editor. The audio editor is based on OSF/Motif widgets and audio library (alib) functions. The audio editor enables the user to record, play, and edit audio files in a variety of file and data formats. It displays a waveform representation of the data to make editing easy and supports basic editing tasks like selecting, cutting, and pasting. The main screen for the audio editor is shown in Fig. 1a.

The audio editor can be invoked or redisplayed by clicking on the audio icon on the HP MPower media bar. Dropping a file from the HP VUE file manager onto the audio icon will bring up the audio editor with the file already loaded and

(a)



(b)

Fig. 1. The audio user interface in HP MPower. (a) The audio editor. (b) The audio control panel.

displayed. A file can be loaded into an already visible audio editor by dropping the file icon onto the editor screen.

Audio Control Panel. The audio control panel is an OSF/Motif interface to global audio parameters such as volume and output device selection (see Fig. 1b). All cooperating audio applications can use the control panel so the user always knows where to go to control these attributes. The audio control panel also provides a Stop button that will stop the current play operation from any cooperating application.

The audio control panel allows the user to turn monitoring on or off. Monitoring involves listening to the audio input signal. In a conventional tape recorder, monitoring allows the user to listen to the audio signal being recorded. On a workstation that has HP MPower, monitoring can be used whether or not anything is being recorded. For example, a user can monitor the workstation's line inputs from a CD player or VCR even when recording is not in progress without using the CPU or other system resources.

Audio Playback. An audio file can be played by simply double-clicking on the file's icon in the file manager or in an audio-enriched mail message. The file will begin playing according to the settings of the audio control panel. A file can also be played by dragging its icon from the file manager to the speaker icon on the HP VUE front panel.

Other Functionality. In addition to the graphical interfaces to audio functionality mentioned above, there are some other capabilities shipped with HP MPower that are accessible from a command line. In the directory /usr/audio/bin executables for the audio editor (audio_editor), the audio control panel (AudioCP), the double-click function (send_sound), and the convert and attributes programs are provided. The convert program converts audio files from any supported file format, data format, and rate and number of channels to any other format and rate. The attributes program tells everything that it can determine or guess about any audio file including file format, data format, sampling rate, number of channels, data length, and header length.

Audio Data and File Types

A number of audio data and file types exist in the industry today, making it difficult for audio files to be shared in a heterogeneous environment.

The lack of standards for audio is currently being addressed by groups such as the IMA (Interactive Multimedia Association). This organization and others are trying to develop standards so that someday there will be a clearer picture as to how to store audio information in a format that is accessible to everyone and can be easily incorporated with other aspects of multimedia.

In the meantime, we chose to support two of the existing file formats and to develop conversion utilities that allow us to support sampling rates, data formats, and byte ordering methods that are not supported directly by our audio hardware.

File Format. A file format is a structure in which there is information about the data as well as the data itself. This information may reside exclusively in a header at the beginning of the file or may be interspersed in "chunks" throughout the file. In the latter case, it is possible that only certain types of chunks are pertinent to audio.

Pertinent information for audio files includes the sampling rate, data format, number of channels, compression technique, and number of samples. Sometimes there is additional information such as loop points or edit markers.

The two audio file formats we chose to support include Microsoft® RIFF/Waveform, which is chunk-based, and the NeXT/Sun audio format, which is a file header followed by data.

We also support audio files that contain only the raw samples. Although this is a popular method of storing audio data and it can be useful in a heterogeneous environment, we recommend using a file format with a header whenever possible because the attributes of the audio data cannot be determined from the samples themselves.

Data Format. Data format defines the method in which audio samples are stored. The most basic method is linear PCM (pulse code modulation). The signed amplitude of the audio waveform is quantized at fixed intervals of time. Each step of the quantization has equal size. This format is very easy to work with and is preferred by most audio editing and mixing routines. One of the formats that our audio hardware supports is the 16-bit linear PCM, which is used by compact disks and is popular on UNIX*-system-based workstations.

An unsigned version of 8-bit linear PCM is very popular among Macintosh and PC users because instead of having an amplitude range of −128 to +127, unsigned (or offset) 8-bit linear has an amplitude range of 0 to 255. All samples are offset by 128. For example, silence, which is normally quantized as 0, is recorded as 128.

We also support the CCITT (International Consultative Committee for Telephone and Telegraph) μ-law and A-law standards. These are companded PCM formats. In these formats the straightforward linear scale is replaced with a base-eight logarithmic scale such that there are small step sizes at low signal levels and large step sizes at high signal levels. The result is a better signal-to-noise ratio and the ability to represent the dynamic range of 13-bit or 14-bit samples with only 8 bits. μ-law and A-law both specify 8-bit samples at 8000 samples per second. μ-law is very common on UNIX-system-based workstations. An overview of the A-law and μ-law data formats is provided on page 65.

Sample Rate. The sample rate refers to the number of digital samples used to represent one second of analog audio. The greater the number of samples, the more accurately the audio signal will be reproduced. A sample rate of 8 kHz (8000 samples/s) can reproduce human voice with adequate clarity, but it does a very poor job on music. For music, 44.1 kHz (44,100 samples/s) works well. The music on all compact disks is recorded at this sample rate. Digital audio tapes (DAT) have a 48-kHz sample rate, producing slightly better audio quality. The audio hardware (described later) supports all of these sample rates plus a few others.

The sampling rate is also subject to the Nyquist criterion, which dictates that the sampling rate must be at least twice the rate of the highest frequency being recorded. Higher frequencies will typically be filtered out by the recording hardware.

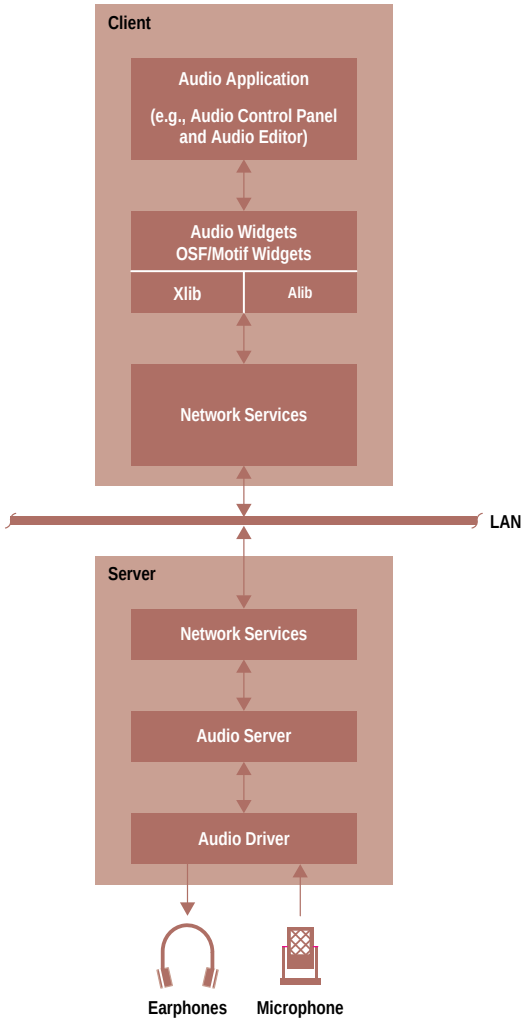


Fig. 2. Client/server architecture for the audio software.

Multiple Channels. Although audio files with more than two channels do exist, most are either mono (one channel) or stereo (two channels). The two channels in a stereo file are typically interleaved on a sample by sample basis. This means there will be a sample for the left channel, followed by one for the right, followed by the next one for the left and so on.

Byte Ordering. One problem with supporting files across a heterogeneous environment is that the byte ordering of the local hardware may be different. In our audio structure we supply information about the byte ordering of the audio hardware connected to the system. Unfortunately, there is no easy way to determine the byte ordering of the audio data in a file that is imported from elsewhere. Therefore we are forced to make assumptions. We assume all RIFF/Waveform files use least-significant-byte-first order and that all other files use most-significant-byte-first order. This applies even to the files created by our audio tools.

Client/Server Architecture

The audio system software uses a client/server architecture that is modeled after the X Window System (see Fig. 2). The client side provides a consistent interface to application programs and handles the connection to the server, which may be local or remote. The server side provides a consistent interface to the client side, handles control of the audio device, and performs data I/O.

The audio server (aserver) interfaces multiple clients to the audio hardware, allowing simultaneous play and record, queuing of multiple play and record transactions, priority preemption, and dynamic buffering.

The audio server also supplies information to the client side about the attributes of the connected audio hardware. The audio library provides the capability to convert various audio attributes to ones that are supported by the connected hardware. Therefore, the hardware attributes (sampling rates, data formats, byte ordering, etc.) do not need to be the same on the client and the server.

Support for Application Developers

Since our audio software subsystem is closely modeled after the X Window System and OSF/Motif, we provide a library, a server, widgets, and a toolkit that are very similar to their X counterparts. When appropriate, we followed the X conventions in our implementation of the various components. For example, the naming convention, function argument convention, and event and error handling of the audio library all follow the conventions used in Xlib. An application developer who is familiar with X and OSF/Motif should find it easy to incorporate audio functionality.

The audio server handles the interface to the audio driver and provides control of audio transactions. This isolates the audio client from hardware-specific device calls and allows a higher level of transaction control than would otherwise be possible.

The audio library is conceptually much like the X library, or Xlib. The audio library provides the low-level application program interface for audio. The audio library provides functions that allow an application to connect to one or more audio servers, to manipulate the configuration of the audio hardware controlled by the servers, to control recording from a server to a file or data stream, and to control playback from a file or data stream to a server. The audio library also provides the capability for applications to play audio from any of several popular file and data formats and to save audio in any of those formats.

The audio widgets provide high-level access to record and play functionality. The application developer can use the widgets without having to learn the lower-level audio library calls. The audio widgets don't export all the functionality of the audio library, but they do provide some flexibility through the use of resources that can be specified by the client application.

The audio toolkit provides callbacks for audio events. Since it is not desirable for audio events to interfere with other events when an application is using the X toolkit, the audio toolkit allows the X toolkit to detect audio events and call the audio toolkit event handler.

Several example programs are also provided for developers. These include the source code and Makefiles. Some of the examples use the widgets and toolkit, and others use only the low-level audio library calls.

(continued on page 66)

Overview of A-law and μ-law Data Formats

An analog audio signal is continuous in time and amplitude (Fig. 1a). In contrast, a digital audio signal is discrete in time and amplitude. An analog-to-digital converter is used to convert an analog audio signal to digital format. This conversion consists of two separate steps: sampling and quantization. Sampling converts the analog signal from continuous time to discrete time by capturing the value of the analog signal at regular intervals in time. Theoretically, the sampled signal only has a value at those sampling times. Fig. 1b shows the sampled version of the signal in Fig. 1a.

The process of quantization maps each sample value to a number. These numbers are represented with a fixed number of bits, giving them limited precision. The quantization process picks the number that best approximates the amplitude of the sample. Essentially, each step in the digital code represents a quantum of increase (or decrease) in the amplitude. Hence the name, quantization. Fig. 1c shows the signal after quantization, along with the digital codes assigned to each quantization level. A digital audio stream is simply a sequence of such digital codes. Although Fig. 1c shows only a few quantization levels, actual implementations use thousands of levels.

The most straightforward form of quantization uses fixed-size quanta of amplitude. In other words, each step in the digital code represents the same step size in amplitude. In this scheme, the mapping from amplitude to digital codes can be represented with a linear function (known as linear quantization). Fig. 2a shows a linear mapping function.

A-law and μ-law define a kind of mapping in which the digital code is roughly equal to the logarithm of the amplitude. Although both laws use the same basic idea, the actual mapping equations are somewhat different in the two laws. Also, to simplify the conversion to or from linear quantization, a piecewise linear approximation is used. Fig. 2b shows a logarithmic mapping function and its piecewise linear approximation.

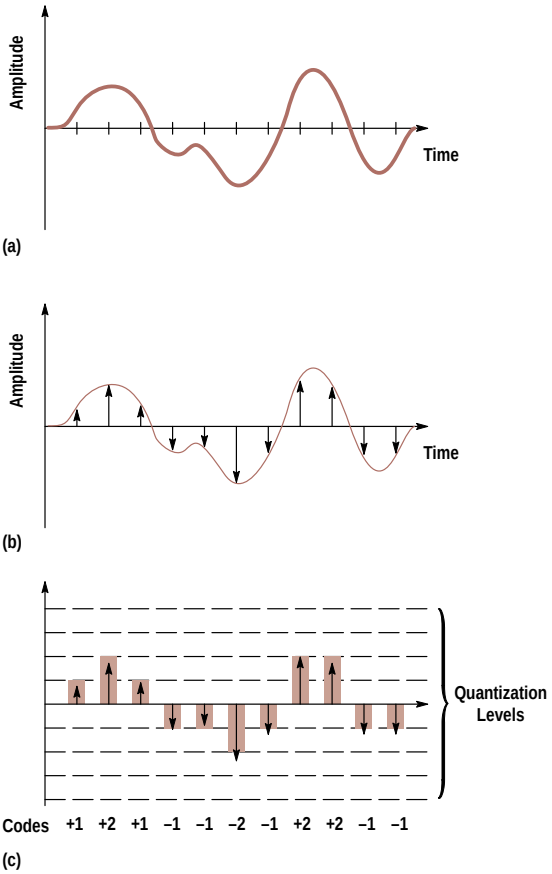


Fig. 1. The stages of converting an audio analog signal from analog to digital format. (a) Original analog signal. (b) Sampled version of the analog signal. (c) Quantization levels assigned to the sample points.

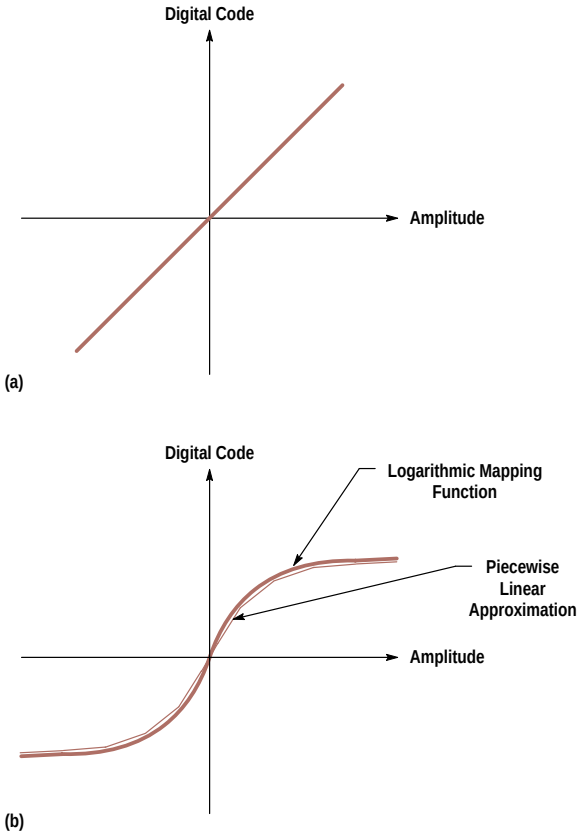


Fig. 2. Methods of quantization. (a) Linear mapping, or linear quantization. (b) Logarithmic mapping function and its piecewise linear approximation.

Telephone companies use A-law and μ-law to carry long distance conversations because these formats save bandwidth. For telephone conversations, signal strength may vary as much as 30 dB because some callers have softer voices than others, or some microphones are more sensitive than others. It is necessary to maintain a 35-dB signal-to-noise ratio (SNR) over the entire range. With linear mapping, quantization noise remains independent of the amplitude level, so one must design for 65-dB signal-to-noise ratio (30 dB + 35 dB) to meet the requirements. As the signal strength varies over the 30-dB dynamic range, the SNR varies between 35 dB and 65 dB. This solution delivers a higher SNR than required at most amplitudes to meet the SNR requirement at minimum amplitude. The price for exceeding the SNR specification in a linear mapping scheme is that 12 bits per sample would be required. However, using A-law or μ-law, 35-dB signal-to-noise ratio can be maintained over a 30-dB amplitude range with just eight bits. This works because logarithmic mapping has the property that larger signal amplitudes result in larger quantization steps. Thus, quantization noise is proportional to the amplitude, maintaining a reasonable SNR across a broad dynamic range. This is a more efficient way to use the quantization levels, making eight bits per sample adequate. Thus, the use of A-law or μ-law results in a 33% reduction in telephone audio transmission. The same benefits are realized in computer audio.

The differences between A-law and μ-law are minimal. The U.S.A., Canada, and Japan use μ-law for telephone transmission, whereas most other countries use A-law for telephone transmission. While A-law is somewhat easier to implement, μ-law provides slightly better quality at low amplitudes. Note that A-law and μ-law work well with voice audio only. For high-fidelity music reproduction, linear mapping with 16 bits per sample is typically used.

Audio Hardware

While audio components such as microphones and speakers work with analog signals, the workstation CPU manipulates data in digital form. Thus, to record audio on a workstation, the analog audio signal must be converted to digital form using an analog-to-digital converter. Similarly, audio playback requires a digital-to-analog converter. The addition of these components turns a workstation into a versatile tape recorder—one that can provide rapid access to a large number of audio clips and associate them with other data types within the workstation.

By supporting several options for sample rates and data formats and offering a choice of stereo or monophonic sound, the audio hardware used on HP 9000 workstations allows the user to make the appropriate trade-offs between quality and storage requirements. For example, higher sample rates result in better quality, but also require more storage. The user has the freedom to choose the appropriate quality.

The audio inputs and outputs are compatible with most consumer equipment, which allows easy connectivity. Two types of inputs are offered on our audio hardware: microphone and line in (for VCRs and compact disks). Only one of these may be selected at any given time. Three outputs are available: speaker, headphones, and line out. Any combination of outputs may be activated simultaneously, but they will all output the same signal. Although the audio design supports these five types of inputs and outputs, some workstations do not contain all five connectors because of space restrictions. Input and output may be activated simultaneously, which means that simultaneous recording and playback is allowed.

Audio under the UNIX Operating System

Audio presents a special challenge to a UNIX system because audio is an isochronous data type. Isochronous means constant with respect to time. This implies that the system must process audio samples at exactly the specified sample rate. If it slows down or speeds up, the listener will perceive distortion in the audio. Unfortunately, the UNIX operating system was not designed to work with isochronous data types. In fact, the UNIX system supports multitasking, which means that the CPU splits its effort between any number of tasks that might be active. Obviously, when there are more tasks outstanding, each task gets less time from the processor. This makes it impossible to guarantee that the audio hardware will get as much processor time as it needs.

Even the hardware infrastructure of the workstation can interfere with audio operation. Just as the CPU cannot guarantee adequate attention to audio, the system bus cannot guarantee adequate bandwidth for audio. The audio hardware design team's challenge was clear: overcome these difficulties while maintaining low cost.

The Solution

The problem description above makes it clear that a solution that guarantees isochronous operation under all conditions does not exist. The designers instead chose an approach that achieves isochronous operation under most conditions. That approach calls for putting a reasonable, not worst-case, upper bound on the delay for any given operation and computing how much audio data could be consumed or produced in that time. A FIFO buffer of that size is required to cover that delay. For example, a heavily loaded CPU may shift its attention away from the audio tasks for one or two seconds. Therefore, more than two seconds of audio is typically buffered in system memory. (Actually, the user can vary the size of that buffer if necessary. A system that is often heavily loaded might need a larger buffer.)

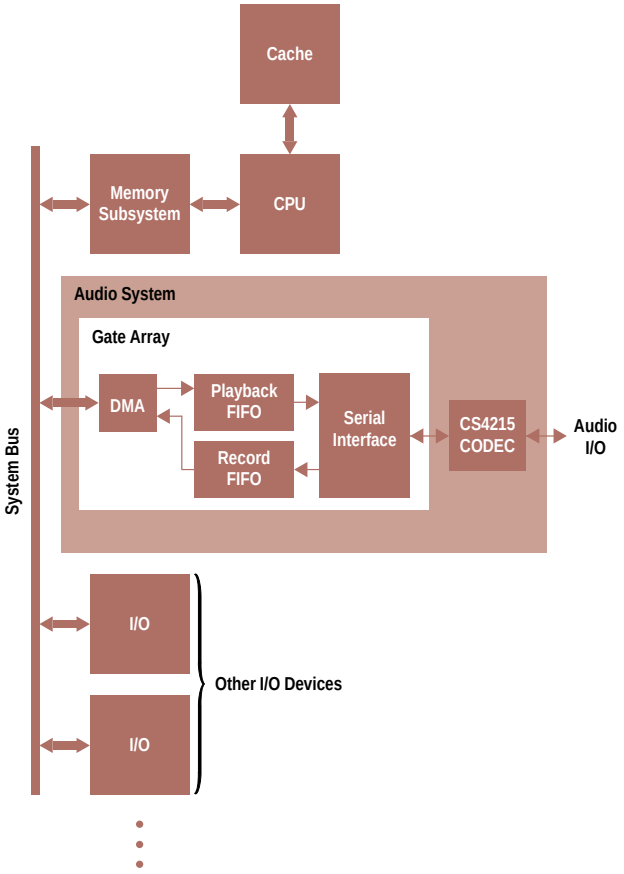


Fig. 3. Block diagram of the audio I/O hardware within a simplified representation of a portion of a workstation.

Two options exist for moving audio data between system memory and the audio hardware. Either the CPU can move it in response to an interrupt, or the audio hardware itself can move it. Since the CPU often turns its attention to other things, it might take several milliseconds to respond to an interrupt. Under the design philosophy described above, the audio hardware would need to buffer enough data to cover that delay. To provide adequate storage space, a separate memory chip would be required, making the audio hardware somewhat more expensive. On the other hand, the audio hardware could access the data using direct memory access (DMA). To perform DMA, the audio hardware must become the master of the system bus. The delay for getting mastership is on the order of tens of microseconds, and the buffer must be able to cover that delay. The DMA approach reduces the size of the buffer by two orders of magnitude. In fact, in the final design, the required buffer size is small enough to implement inside a simple gate array (see Fig. 3). The gate array is required to interface to the bus anyway, so the cost increase was negligible. Of course, the DMA logic added design complexity. Thus, cost was reduced through increased R&D effort. To make the audio hardware inexpensive enough to offer it as a standard feature, the engineering trade-offs had to favor lower cost.

Physical or Virtual DMA?

All modern workstations employ a virtual memory system that allows the CPU to work with virtual memory spaces that are larger than the physical memory available in the system. All programs access memory using a virtual address, and the CPU hardware translates it to a physical address. In contrast, DMA accesses do not go through that same hardware, so the hardware initiating a DMA request must supply the physical address to the bus. In that sense, HP workstations do physical DMA, not virtual DMA. Still, an I/O device could include the hardware necessary to do virtual-to-physical translation, effectively giving that device the ability to do virtual DMA. Since programs work with virtual addresses, they could communicate with I/O hardware more easily if that hardware did virtual, rather than physical DMA.

Unfortunately, virtual-to-physical translation hardware adds complexity and cost to I/O hardware. For that reason, our audio hardware does not do virtual DMA. Instead, the driver software assumes the responsibility of presenting the hardware with physical addresses. Again, the trade-off was made in favor of lower system cost.

Hardware Components

Fig. 3 shows a block diagram of the audio hardware components within a simplified representation of a workstation. As shown in the figure, the audio hardware connects to the workstation's system bus. The CPU uses the system bus to initialize the audio hardware with the desired parameters such as sample rate, volume level, and so on. The DMA block uses the system bus to read audio data for playback and to write audio data for recording. It writes the playback data into the playback FIFO and reads the recorded data from the record FIFO. Each FIFO's size is 8 words by 32 bits. The other ends of the FIFOs connect to the serial interface block. This hardware converts audio data from parallel form, which the FIFOs use, to serial form, which the audio CODEC requires. The term CODEC is an abbreviation for coder/decoder. In this context, analog-to-digital converters are called coders and digital-to-analog converters are called decoders. The audio CODEC implements two converters of each type in a single chip, allowing stereo operation. Some of the CODEC inputs and outputs are buffered with analog amplifiers. In some cases, the amplifiers provide more gain, while in others they help match input or output impedance. The CODEC also implements the logic required to support the various sample rates and data formats discussed earlier. The analog-to-digital and digital-to-analog converters inherently operate with 16-bit linear data. So, if A-law or μ-law mode is selected, the CODEC converts playback data from the selected mode to 16-bit linear and recorded data from 16-bit linear to the selected mode.

The CODEC is a commercially available part. A single gate array implements the rest of the logic in the audio hardware. Some salient features of this HP-designed gate array are:

- 4,953 gates
- 1.0-micrometer technology
- 120 PQFP (plastic quad flat pack) package.

The process used for this gate array allows finer-pitch I/O pads than most similar processes. The finer pitch is better matched to the particular ratio of gates to pins of this chip. If manufactured in another process, this chip would have cost more because of a larger die area.

Conclusion

For audio to be useful on the desktop, audio capabilities must be pervasive. The HP 9000 Series 700 workstations have kept the cost of audio hardware and application development low by avoiding special-purpose hardware like digital signal processors in favor of using the power of the PA-RISC processor to handle digital audio signals. This allows HP to ship high-quality audio with every workstation. Our audio offering is completed with an audio server and basic audio tools to get users started with audio, and a library containing audio widgets, a toolkit, and program examples for application developers.

Bibliography

1. John Bellamy, *Digital Telephony, Second Edition*, John Wiley and Sons, 1991, pp. 110-119.