

# Audio Distribution and Control using the IEEE 1394 Serial Bus

*Bob Moses  
Greg Bartlett*

*PAVO, Inc.  
Seattle, Washington, USA*

## Abstract

This paper describes a system for transmitting hundreds of channels of high quality digital audio, control & monitoring protocols, and digital video over an IEEE 1394 serial bus. The paper provides a brief tutorial on the IEEE 1394 standard, paying specific attention to digital audio formats and synchronization methodologies. Recommendations are made for the audio industry to standardize audio protocols above the IEEE 1394 standard to enable future interoperability between a variety of devices.

## Introduction

The world needs a multichannel digital audio interconnect standard. The majority of today's digital audio systems are connected using the AES3 and IEC958 (a.k.a. "AES/EBU" and "SPDIF") interfaces. These standards were designed to carry a single stereo audio stream in a fixed point-to-point connection between two devices, and were optimized for the Red Book CD audio format. Each pair of audio channels must be transferred over an individual cable, adding significant cost and complexity to multichannel systems. Their asynchronous nature complicates the process of synchronizing multiple interfaces, generally requiring expensive sample rate conversion. Control information is typically passed along separate proprietary paths, such as MIDI (via System Exclusive commands), RS-232/422, or infrared transmission. These standards lack the data-carrying capacity and flexibility to support tomorrow's sophisticated digital media systems comprised of multichannel digital audio, digital video, sophisticated control & monitoring protocols, and TCP/IP.

Because of the inherent limitations of AES/EBU and SPDIF interfaces, developers of multichannel digital audio equipment have invented a number of proprietary interconnects for exchanging digital audio data between multiple devices. For example, the Alesis ADAT Optical

Digital Interface carries 8 channels of 24-bit PCM digital audio over a daisy-chained plastic optical fiber bus between as many as 16 devices. Time code-based synchronization and MIDI control data are transferred over a separate proprietary 9-pin D-sub connector interface. Similarly, the TEAC Digital Interface Format (TDIF) exchanges 8 channels of 24-bit PCM digital audio over a parallel point-to-point connection between two devices. Separate physical interfaces (terminated in D-sub and BNC connectors) provide the necessary synchronization signals between devices.

Both of these interfaces have enjoyed widespread commercial success on a variety of professional digital tape recorders and digital signal processing equipment. However, they lack the scalability, extensibility, ease of use, and compatibility with the variety of streaming data types required by modern digital media equipment. Furthermore, since these interconnects are privately licensed, they are not available as a ubiquitous public domain interface.

To ensure that future digital media devices can interoperate effectively, a new standard is required which meets the following criteria:

- ⌚ Supports the data formats and bandwidth requirements of current and future multichannel audio standards (e.g., Dolby Pro Logic, Dolby Digital/AC-3, DTS, MPEG-II, and DVD).
- ⌚ Carries digital video and control & monitoring protocols in the same cable to reduce system complexity and cost.
- ⌚ Guarantees precise synchronization between video and audio signals, especially after arbitrary amounts of processing in the system.
- ⌚ Is widely adopted in the computer and A/V industries – i.e., open to everyone – preferably standardized by an international standards body.
- ⌚ Promotes interoperability between all types of A/V and computing devices: personal computers, DVD players, DBS and DSS systems, cable modems and set top boxes, surround sound decoders, tuners, amplifiers, speakers, video projectors, disk drives, digital camcorders and still cameras, D-VHS digital VCRs, video teleconferencing systems, portable computer docking stations, personal digital assistants, gaming platforms, color printers and scanners, musical instruments, and so on. *The world needs fewer standards, not more.*

- ⌚ Allows devices to be added to or removed from the system by a lay person without attention to bus termination, node identification, and other complicated technical details.
- ⌚ Does not *require* a computer to set up or operate the system, but does allow one or more computers to be peer nodes in the system.
- ⌚ Provides an integrated human interface to the system (e.g., using wall-mounted controls, on screen GUI, or remote controls) with a minimum number of controls, preferably at multiple locations.
- ⌚ Is cost effective.
- ⌚ Is extensible, supporting higher levels of performance in the future.
- ⌚ Is robust and reliable.

Fortunately, a standard called *IEEE 1394* exists which fulfills these requirements[1]. Originally invented by Apple Computer (and named “FireWire”), IEEE 1394 has been adopted by over 120 companies in the audio, video, and computer industries and hailed as the interconnect of choice for future multimedia systems. This paper describes the authors’ experiences implementing audio devices for the IEEE 1394 bus, and proposes several areas for further standardization.

## 1394 Glossary

**IEEE 1394** – The official name of the standard is “IEEE 1394-1995 Standard for a High Performance Serial Bus.” It is published by the Institute of Electrical and Electronics Engineers (IEEE). Planned enhancements (supporting speeds up to 3.2Gb/s) to the IEEE 1394 standard are typically referred to with a revision letter suffix (e.g., 1394a, 1394b).

**IEC 1883** – International Electrotechnical Commission (IEC) standard for a Digital Interface for Consumer Electronic Audio/Video Equipment, based on IEEE 1394.

**FireWire** – Apple Computer’s trademarked name for its implementation of IEEE 1394. FireWire is the most commonly used synonym for IEEE 1394.

**1394 Trade Association** – The 1394 Trade Association was formed in 1994 to facilitate adoption of 1394 by consumer electronics and personal computer manufacturers. This trade association boasts over 120 member companies, representing the world’s most powerful OEMs.

**SC100C** – The IEC subcommittee that is responsible for developing five separate standards that embrace 1394: a general digital interface for multimedia, three languages for use with digital VCRs, and a language for Moving Picture Experts Group (MPEG) data transmission.

**FDIS** – The SC100C’s Final Draft International Standard (FDIS), to be published late in 1997.

**mLAN** – Yamaha’s Music LAN, based on the 1394 bus. In 1997, a portion of mLAN was approved as a 1394 Trade Association specification, under the name “Audio and Music Protocol.”

**AV/C** – The AV/C Digital Interface Command Set, created by the A/V Working Group of the 1394 Trade Association, defines a protocol for controlling and monitoring simple consumer electronics audio/video devices.

**FCP** – Function Control Protocol, defined by IEC 1883, defines a means to carry control & monitoring protocols (such as the AV/C Digital Interface Command Set) within 1394 asynchronous block write transactions.

**DV** – Sony’s proprietary digital video compression system, incorporating IEEE 1394 as its standard digital video interface.

**DVCPRO** – Pioneered by Panasonic, this variant of the DV format was specifically designed for broadcast TV and industrial videographers. DVCPRO uses a somewhat larger cassette than the DV cassette.

**Asynchronous** – Asynchronous transactions provide a fully acknowledged guaranteed delivery mechanism (i.e., always delivered). At least 20% of the bus bandwidth is reserved for asynchronous transfers. This mode is useful for transmitting data with non-critical timing properties (e.g., control & monitoring protocols).

**Isochronous** – Isochronous transactions provide a constant latency within a specified bandwidth (i.e., always on-time). Up to 80% of the bus bandwidth is available for isochronous transmissions. This mode is useful for time-critical data such as audio and video signals.

**Isochronous Cycle** – The period of time in which isochronous packets may be transmitted on to the bus. The isochronous cycle is initiated when the Cycle Master node transmits a Cycle Start packet, and ends after all nodes have sent their isochronous data. Isochronous cycles are guaranteed to occur every 125 $\mu$ s.

**Hot-Pluggable** – The characteristic of IEEE 1394 that allows a device to be connected and disconnected to the bus while powered, without disrupting other devices on the bus.

**Physical Layer** – Performs the analog cable interface, arbitration, and bus initialization functions required by IEEE 1394.

**Link Layer** – Data is addressed, error-checked, and formatted into packets in the Link Layer. It provides a layer of abstraction between the host application (e.g., a digital signal processor handling digital audio streams) and the lower level bus arbitration in the Physical Layer.

**Transaction Layer** – Defines a request-response (Read, Write, and Lock) protocol to perform asynchronous bus transactions in accordance with the IEEE 1212 Control and Status Register (CSR) architecture.

**CYCLE\_TIME Register** – Every node supporting isochronous transactions contains a 32-bit register which increments at a nominal rate of 24.576MHz. The Cycle Master copies its CYCLE\_TIME register to all other isochronous nodes every 125 $\mu$ s to provide a common isochronous timing reference.

**Cycle Master** – The Cycle Master is a node chosen automatically during bus initialization that maintains a common clock reference for all nodes on the bus. The Cycle Master transmits a *Cycle Start* packet every 1250s and copies its CYCLE\_TIME register to all other isochronous-capable nodes, thereby synchronizing all nodes to a common time base. Devices that do not support isochronous transactions are not required to operate as a Cycle Master.

**Isochronous Resource Manager** – The Isochronous Resource Manager, chosen automatically during bus initialization, provides a known location for all nodes to verify the availability of isochronous resources, both channels and bandwidth. Devices that do not support isochronous transactions are not required to operate as an Isochronous Resource Manager.

**Bus Manager** – During bus initialization, one node may be selected to perform power management and bus optimization functions; this is the Bus Manager. Devices are not required to be Bus Manager-capable. If no Bus Master is available, the bus operates without one.

**Node** – A device on the 1394 bus.

**Quadlet** – A 32-bit binary data word.

## IEEE 1394 History

### 1986: Apple Computer Begins Work on its Proprietary FireWire Bus

In 1986, after an heroic attempt at carrying digital audio over the Apple Desktop Bus failed, a group of engineers at Apple Computer refocused their attention on creating a fast, inexpensive, and simple serial bus that would replace many of the ports on personal computers.

Understanding that their new bus would have to compete with every new multimedia digital bus showing up in the trade journals, the team knew that speed and flexibility were the keys to their success.

In the interest of time, the Apple engineers adapted a flexible, inexpensive serial bus called IEEE 1394; at that time, IEEE 1394 was an unimplemented standard bus for diagnosing errors in backplane parallel data buses. To support multimedia, a technique was developed to stream data in realtime, something no other bus could do at the time.

Over several years, innovations were introduced to increase speed and reduce costs. Analog functions were tweaked using affordable CMOS chips operating at a modest 50Mbit/s. Data strobe encoding was added, doubling the speed to 100Mbit/s. An adaptive arbitration scheme was developed, simplifying bus topology and allowing the bus to configure itself. Using time-division multiplexing (TDM), fixed-rate multimedia streams began sharing the bus with traditional computer datagrams. To reduce cost, an optical fiber physical layer was replaced with a twisted-pair copper solution, with the help of Hosiden and Molex.

## 1990: Apple Gives FireWire to the World

Originally intended to be a proprietary Apple technology to replace the Apple Desktop Bus, FireWire became the subject of an important 1990 announcement: Apple would begin adapting the old IEEE 1394 standard as FireWire. An IEEE working group was established, including delegates from Apple, Texas Instruments (the first company to develop a 1394 Physical Layer chip), Stewart Connector, Molex, Adaptec, and Western Digital. In 1992, the IBM PC Company joined the working group, and the standards discussions reached critical mass.

## 1993: Apple Demonstrates FireWire

In 1993, Apple demonstrated FireWire to the public. In the demo, QuickTime video (stored on a hard drive) was simultaneously displayed with real time, full-motion formatted video delivered directly to the display memory over the IEEE 1394/FireWire bus. For the first time, streaming data and time-sensitive control information were demonstrated on the bus.

## 1994: Trade Association is Formed; the First IEEE 1394 Chipset Arrives

In 1994, early adopters of IEEE 1394 formed the 1394 Trade Association (1394 TA) (<http://www.1394ta.org>) to support the development of computer and consumer electronics systems that can be easily connected via 1394. The 1394 TA is currently comprised of over 120 of the leading companies in electronics, computers, and consumer products, including:

- |                                      |  |                                    |
|--------------------------------------|--|------------------------------------|
| Ⓢ 3A International, Inc.             | Ⓢ 3Com                                   | Ⓢ Acer Laboratories, Inc.          |
| Ⓢ Adaptec                            | Ⓢ Advanced Technology and Systems        | Ⓢ AirBorn, Inc.                    |
| Ⓢ AMD                                | Ⓢ AMP                                    | Ⓢ Apple Computer, Inc.             |
| Ⓢ AST Research, Inc.                 | Ⓢ Award Software International, Inc.     | Ⓢ Belkin Components                |
| Ⓢ Berg Electronics                   | Ⓢ Boston Optical Fiber                   | Ⓢ C & M Corporation                |
| Ⓢ Canon Information Systems          | Ⓢ Cirrus Logic, Inc.                     | Ⓢ CMD Technology, Inc.             |
| Ⓢ Compaq Computer Corporation        | Ⓢ Computer & Communications Research Lab | Ⓢ Cypress Semiconductor            |
| Ⓢ Daewoo Electronics                 | Ⓢ Dell                                   | Ⓢ Diamond Multimedia Systems, Inc. |
| Ⓢ Digital Vision Laboratories        | Ⓢ Eastman Kodak Company                  | Ⓢ Fast Electronic U.S., Inc.       |
| Ⓢ Firefly, Inc.                      | Ⓢ Framatome Connectors USA               | Ⓢ Fuji Photo Film Co., Ltd.        |
| Ⓢ Fuji Xerox Co., Ltd.               | Ⓢ Fujitsu, Ltd.                          | Ⓢ Future Techno Designs PTE, Ltd.  |
| Ⓢ General Instrument                 | Ⓢ Hamamatsu Photonics K.K.               | Ⓢ Harman International Industries  |
| Ⓢ Hewlett Packard                    | Ⓢ Hitachi                                | Ⓢ Hosiden Corporation              |
| Ⓢ IBM Microelectronics               | Ⓢ In-System Design                       | Ⓢ Innovative Semiconductors, Inc.  |
| Ⓢ Integrated Silicon Solutions, Inc. | Ⓢ Intel Corporation                      | Ⓢ Interex, Inc.                    |
| Ⓢ JVC                                | Ⓢ KC Technology, Inc.                    | Ⓢ Kenwood Corporation              |
| Ⓢ Kobe Steel, Ltd.                   | Ⓢ Lexmark International, Inc.            | Ⓢ LG Electronics, Inc.             |
| Ⓢ LSI Logic                          | Ⓢ Lucent Technologies                    | Ⓢ Macro Designs, Inc.              |

Ⓢ Matrox Electronic Systems	Ⓢ Matsushita Electric Industrial Co., Ltd.	Ⓢ Maxtor
Ⓢ Micrel Semiconductor	Ⓢ Micron Semiconductor Products, Inc.	Ⓢ Microsoft Corp.
Ⓢ Mitsubishi Electric Corp.	Ⓢ Molex, Inc.	Ⓢ Motorola Semiconductor Products
Ⓢ Naltec, Inc.	Ⓢ Nanao Corporation	Ⓢ National Instruments
Ⓢ National Semiconductor Corp.	Ⓢ NEC Electronics, Inc.	Ⓢ NEC Systems Laboratory
Ⓢ Newnex Technology Corp.	Ⓢ Nihon Synopsys Co., Ltd.	Ⓢ Nikon Corp.
Ⓢ Nippon Columbia Company, Ltd.	Ⓢ Nippon Steel Corp.	Ⓢ Nokia Multimedia Network Terminals
Ⓢ O2Micro, Inc.	Ⓢ Opti, Inc.	Ⓢ O. R. Technology
Ⓢ PAVO, Inc.	Ⓢ Phase One Denmark A/S	Ⓢ Philips Semiconductors
Ⓢ Phoenix Technologies, Ltd.	Ⓢ Pioneer Electronic Corp.	Ⓢ QLogic Corp.
Ⓢ Quantum Corp.	Ⓢ Ricoh Company, Ltd.	Ⓢ Rockwell Semiconductor Systems
Ⓢ Roland Corp.	Ⓢ Samsung Information Systems America	Ⓢ Sand Microelectronics, Inc.
Ⓢ Sanyo Electric Co., Ltd.	Ⓢ Seagate Peripherals	Ⓢ Seiko Epson Corp.
Ⓢ SGS-Thomson Microelectronics	Ⓢ Sharp Corp.	Ⓢ Sican GmbH
Ⓢ Silicon Integrated Systems Corp.	Ⓢ Silicon Systems	Ⓢ Sony Electronics, Inc.
Ⓢ Standard Microsystems Corp.	Ⓢ Sun Microsystems	Ⓢ Symbios Logic
Ⓢ Synopsys, Inc.	Ⓢ SystemSoft Corp.	Ⓢ Technology Rendezvous, Inc.
Ⓢ Tekom Technologies, Inc.	Ⓢ Texas Instruments	Ⓢ Thomas and Betts
Ⓢ Thomson Consumer Electronics R&D France	Ⓢ Toshiba	Ⓢ Toshiba America Information Systems, Inc.
Ⓢ Unibrain SA	Ⓢ U. S. Robotics	Ⓢ VIA Technologies, Inc.
Ⓢ VLSI Technology Inc.	Ⓢ Vobis Microcomputer AG	Ⓢ Western Digital Corporation
Ⓢ Winbond Electronics Corp.	Ⓢ Yamaha Corp.	Ⓢ Yokogawa Electric Corp.

### **1995: The First Product; the IEEE 1394 Standard is Approved**

Late in 1994, Texas Instruments released the first compliant IEEE 1394 chipset. In July of 1995, Sony used this chipset to create the first 1394-based commercial product—a digital video camcorder. Sony’s camera boasted a “1394-compliant DV Connector,” based on their proprietary implementation of digital video streaming on the standard IEEE 1394 bus.

In December of 1995, after years of intense negotiations among the converging computer and audio/video industries, the IEEE 1394 standard was approved.

### **1996: The First Public Demo of Audio over 1394**

One month after the 1394 standard was approved, Microsoft, Texas Instruments, and PAVO joined forces, determined to demonstrate audio streaming over 1394 within three short months. PAVO, a digital audio product development company, developed the audio implementation;

Texas Instruments worked to integrate their fledgling chipset into PAVO's design; and Microsoft modified their Windows operating system to support the audio hardware.

The world's first public demonstration of a 1394 audio system – Bill Gates' keynote address at the 1996 Windows Hardware Engineering Conference (WinHEC '96) – was conducted with the resulting 1394 Digital Audio Bridge. In the demo, a 60-second Dolby Pro Logic-encoded WAV file (16-bit, 44.1kHz) was transmitted from a PC hard disk to a Texas Instruments PCI/1394 host adapter card, over the 1394 bus to PAVO's Digital Audio Bridge. The Digital Audio Bridge extracted the audio packets, retransmitting them in real time over an SPDIF link to an audiophile surround-sound system installed in the auditorium. Over 2000 computer hardware engineers attending the conference were introduced to the concept of dramatically improved sound for the personal computer platform; Microsoft's vision of an entertainment computer was unleashed.

### **1997: Multiple 1394 Products are Demonstrated**

1394 audio/video product announcements and demonstrations became commonplace in 1997. Among them, JVC showed a 1394 Digital Video Cassette Recorder (DVCR). Sony introduced its third 1394-compliant camcorder model, while Panasonic released its first. 3A International offered a 1394 bus analyzer and bus extender nodes. PAVO introduced the first 1394 legacy audio adapter, selected by Microsoft as the audio reference design for Win32 Driver Model (WDM) drivers used in their upcoming Memphis and NT 5.0 operating systems. Harman Kardon, in conjunction with IBM, Toshiba, Microsoft, and Symbios Logic, demonstrated a 200-disc dual DVD transport. Several companies have introduced DV-decoder/1394 PCI-bus cards for use in digital video editing applications. Maxtor (hard drive), Toshiba (DVD drive), Texas Instruments (laptop computer, PCI adapter), Adaptec (PCI adapter), Yamaha (mLAN interface), Molex and Hosiden (connectors), NEC (plastic optical fiber transceivers), Kenwood/Firefly (bus analyzer), and others have now demonstrated 1394 products at trade shows.

Perhaps the truest indicator of the perceived future of 1394 in audio and video systems, nearly every semiconductor manufacturer has now announced plans for 1394 chipsets.

## **1394 Anatomy**

### **Connector and Cabling**

The childproof 1394 connector was originally created for the Nintendo Gameboy, and has proven to be reliable in the harshest environments (e.g., in the hands of children). The cable, with a diameter of less than 1/4in, is very flexible and durable. Two variants of this connector/cable have been standardized. One has 6 conductors – 4 for carrying data, and 2 for power (8 to 40

VDC at up to 1.5A). The other has only the 4 data conductors; it is intended for miniature battery-powered devices.

The power supplied in the 6-conductor cable enables the Physical Layer of each node to remain operational even if node's local power is off. In fact, an entire node can be powered from the 1394 bus if its power requirements are modest.

Maximum cable length depends on the cable characteristics. The most common 1394 cable contains 28AWG wires and can carry data up to 4.5m. Cables with 24AWG conductors can run up to 14m, and several companies (e.g., NEC, Sony) have announced plastic optical fiber solutions that allow for 1394 cable lengths of 100m.

## **Electrical Isolation**

Annex A of the IEEE 1394 standard [1] establishes strict electrical isolation requirements, as summarized here:

1. The cable shield must be connected to the external enclosure (i.e., chassis ground) and the cable power ground conductor must be isolated from the node's internal power ground. Typically, the cable power ground is connected to an isolated regulation circuit to power the PHY chip. The PHY chip is electrically isolated from the LINK chip using transformer, capacitive, or optical coupling.
2. The signal lines in the cable must be DC isolated from the system power distribution ground.
3. The cable power conductor and its ground must be isolated from the system power distribution ground. In all cases, the isolation must prevent any part of the external cable from becoming a low impedance electrical connection between different power domains.
4. At all times, the algebraic sum of all DC currents in the six cable conductors at every connector must be less than 50mA.

Of course, the actual implementation of these requirements could vary from vendor to vendor (and device to device). Therefore, it is possible for grounding problems typical to audio systems to occur. Exact implementation of the IEEE 1394 standard, however, should eliminate most electrical isolation problems.

## **Topology**

The topology of a 1394 system can be a daisy chain, tree, star, or a combination of these. The 1394 standard specifies that two devices should not have more than 16 cable hops between them,

using normal cables. Up to 63 devices may be connected in one bus, and up to 1023 buses can be interconnected to create a very large network with over 64,000 devices. Each node may have up to 256 terabytes of memory addressable over the 1394 bus. A fair bus access mechanism guarantees that all nodes have equal access to the bus.

### **Automatic Configuration**

During system initialization, each node in a 1394 bus carries out a process of bus initialization, tree identification, and self identification. These functions are carried out quickly and automatically with no human intervention.

No master controller is required in a 1394 system, eliminating the communications bottleneck and single point of failure common to many other systems. Owing to 1394's support for "hot plugging," new devices may be added to or removed from the network without disrupting other nodes.

### **Data Rate**

The data rates defined for the 1394 cable environment are 98.304Mbit/s, 196.608Mbit/sec, and 393.216Mbit/s (referred to as *S100*, *S200*, and *S400*). Devices with different data rates may be freely interconnected; communications will automatically be performed at the highest rate supported by the lower-rate devices.

### **Asynchronous and Isochronous Operation**

IEEE 1394 supports two primary means of transferring data; *asynchronous* and *isochronous*. Isochronous streams are transmitted with predictable latency, guaranteed bandwidth, and on-time delivery. Isochronous transmission allows a multitude of audio and video streams to be transmitted through the system. IEEE 1394 supports up to 63 independent isochronous "channels," each of which could contain an unlimited number of logical audio or video channels (limited by bandwidth, of course). In a multimedia system, for example, one isochronous channel could carry a 5.1 channel surround sound audio signal and an uncompressed digital video signal.

Isochronous operation is divided into 125 $\mu$ s time segments, called *isochronous cycles*. The cycle begins when the bus Cycle Master (any isochronous-capable node, automatically selected during bus initialization) arbitrates for the bus and transmits a special asynchronous packet, called a *Cycle Start* packet. Within this packet is the value of the Cycle Master's clock counter. All devices on the bus receive this value and update their own local clock counter value, guaranteeing that the bus operates to a common time reference.

Asynchronous transfers can take place any time the bus is free of isochronous traffic. The 1394 protocol guarantees that at least 20% of bus time is reserved for asynchronous data transfers. A short acknowledge packet is returned to the sender for every asynchronous packet, providing a mechanism for guaranteeing delivery.

Asynchronous transactions comply with the IEEE 1212 Control and Status Register (CSR) architecture [2]. This standard defines a protocol for reading, writing, and locking memory locations in a node on the bus. Additional protocols have been defined for controlling and monitoring parameters on remote nodes, and for establishing virtual connections in the system.

## **Implementing an IEEE 1394 Bus Interface**

Figure 1 shows the layers in a typical IEEE 1394 implementation. The Physical Layer provides low level access to the 1394 bus. IEEE 1394 devices may have 1, 2 or 3 connectors, each connected to a single "PHY Chip." Cable power is typically extracted from the bus and used to power the PHY, maintaining the integrity of a daisy-chained bus while a 1394 device is turned off. A device may optionally supply power to the bus via a diode-OR circuit. The PHY chip exchanges raw data and system clock signals with the Link Layer Controller chip.

The Link Layer is responsible for properly formatting isochronous and asynchronous packets for transmission, and buffering incoming packets for processing by the Application Layer. The IEEE 1394 standard allocates an undefined block of payload data to isochronous packets, and does not dictate specific audio or video data formats. Such formats are determined at the Application Layer. Asynchronous messages provide read/write access to shared memory on each node. Typical Link Layer Controller chips provide host interfaces compatible with common microprocessors, PCI buses, and synchronous serial buses.

The Application Layer formats packets for specific audio, video, and control applications. Typically, a DSP or RISC processor is employed as a host processor to the Link chip. If the 1394 interface is intended for a computer, the Application Layer is often implemented in operating system drivers and application software. The Application Layer is not standardized today, and has been implemented differently in a variety of different devices. For example, Microsoft is developing a driver model (WDM) and API for future versions of the Windows operating system.

## Audio Distribution over IEEE 1394

An IEEE 1394 system is a bus of peer nodes. Although one node may optionally be designated as the Bus Master, Cycle Master, or Isochronous Resource Manager by low level arbitration, no node is required to assume the role as overall master controller for the bus. All operations are performed in a distributed peer-to-peer manner. This architecture lends itself well to audio systems, since audio devices are traditionally connected in a peer-to-peer fashion. (There is no concept of a master controller in analog systems.) The hot-pluggable feature of 1394 allows audio devices to be added to or removed from the bus while the system is operating. This is especially important in live public address systems in which interruption of audio signals is detrimental to the overall experience.

IEEE 1394 isochronous transmissions are broadcast on to the bus with a channel identifier, in a connectionless architecture. Any isochronous-capable node may listen to any isochronous channel, and if a priori knowledge is available indicating which audio streams are transmitted over which isochronous channels, it is straightforward for receiving devices to dynamically tune in to any desired audio signal on the bus.

However, it is often desirable to assign specific connections between I/O *plugs* on bus nodes to establish proper signal routing. In a connectionless bus such as IEEE 1394, these plugs are virtual (i.e., they are software vectors, not physical connections). The plugs may represent an actual hardware I/O port, or buffers in the heart of a DSP. A comprehensive *plug control* model has been defined [3] for establishing virtual connections between source and sink points on the bus. The details of this plug control model are beyond the scope of this paper, but a brief summary is provided in the following section.

### Audio Routing using Plug Control

Under the plug control model, any device capable of transmitting or receiving isochronous data implements virtual plugs for its inputs and outputs. A *connection* is made between an output plug on one node and one or more input plugs on other nodes. *Plug Control Registers* (PCR) on each node specify the isochronous channel, device specific I/O port numbers, and other parameters necessary for managing virtual connections across the bus. Asynchronous transactions are used to communicate plug control register settings, called Connection Management Procedures (CMP). PCR settings may or may not be persistent when a device's power is cycled, depending on the implementation.

Figure 2 shows an example of two devices implementing this plug control model. Node A has 4 output plugs with associated plug control registers. Node B has 4 input plugs and plug

control registers. A virtual connection is established by writing the identity of the source and sink nodes into each respective PCR, represented in this example as [node ID – plug #]. Once the PCRs are written, the source node (node A) may transmit isochronous data to the sink node (node B).

A personal computer is typically used to manage plug control, although it is possible for audio devices to establish connections with each other directly without the intervention of a computer.

### Formatting Audio Packets for IEEE 1394

The IEEE 1394 standard allocates 80% of bus bandwidth to isochronous transactions. This enables data throughput of 80Mbits/s, 160Mbits/s, or 320Mbits/s, depending on the bus data rate (S100, S200, or S400). Dividing these data rates by audio sample rates and word sizes reveals the number of audio channels that can be carried in a single 1394 bus (ignoring overhead and bus arbitration inefficiencies). Table 1 illustrates several possible scenarios for utilizing this bandwidth at the S400 rate with typical audio formats; several popular video formats are shown for comparison. Dividing these figures by 2 and 4 will provide an estimate of channel capability at the S200 and S100 rates, respectively.

<b>Data type</b>	<b>Bandwidth per channel</b>	<b>Number of channels supported at S400 rate</b>
16-bit, 44.1kHz audio	705.6Kbits/s	453
24-bit, 48kHz audio	1.152Mbits/s	277
Dolby Digital/AC-3 5.1 channel audio	384Kbits/s	833
DTS 5.1 channel audio	384Kbits/s	833
<b>High quality video</b> <i>30fps, 640x480 pels, 24-bit color</i>	221Mbits/s	1
<b>Reduced quality video</b> <i>15fps, 320x240 pels, 16-bit color</i>	18Mbits/s	17
<b>MPEG-2 video</b> <i>“NTSC quality”</i> <i>“PAL quality”</i> <i>complex image</i>	<b>2 - 15Mbits/s</b> <i>3Mbits/s</i> <i>4Mbits/s</i> <i>5-10Mbits/s</i>	<b>21 – 160</b> <i>106</i> <i>80</i> <i>32 - 64</i>
DV compressed video	20Mbits/s	16

*Table 1: Bandwidth Requirements for popular audio and video formats (ignores protocol overhead and bus inefficiencies)*

Most bus implementations carry a mix of audio and video formats. Although it would be theoretically possible to format a single packet containing a mixture of data from different A/V sources, this would be prohibitively complicated to manage in an open system. Therefore, one isochronous channel (of the 63 available) is typically allocated to each format. Since only one

node may transmit data on a particular isochronous channel, the total number of transmitting nodes multiplied by the total number of different data types can not exceed 63 per bus. This is not typically a problem since only a few nodes transmit data at one time, and only a few different data types are commonly supported within one system. Dynamic plug control allows the system to allocate isochronous channels to nodes on an as-needed basis.

The IEEE 1394 standard establishes the basic mechanism for transferring isochronous data from one node to another, but does not dictate specific formats for audio or video data. Virtually the only limitation on isochronous payload data is that it is limited to between 0 and 256 quadlets at the S100 base rate (256 32-bit quadlets every 125 $\mu$ s results in a data rate of 65Mbits/s, which falls within the 80Mbits/s available). This limit is proportionally higher at the S200 and S400 bus rates. Because the isochronous payload format is undefined, a protocol for formatting audio and video data must be added above the IEEE 1394 standard in the Application Layer. Several such protocols have been implemented to date [6] [7]; a brief summary of these follows.

### **Early Unformatted Packet Formats**

The first public demonstration of audio transmission over 1394 was made in a keynote address – delivered by Microsoft Chairman Bill Gates – at the 1996 Windows Hardware Engineering Conference (WinHEC 96). The demo system, developed in a partnership between PAVO, Microsoft, and Texas Instruments, carried a stereo 16-bit 44.1kHz digital audio stream on one isochronous 1394 channel. Microsoft wrote a simple application program that read 16-bit Dolby Pro Logic-encoded WAV files from a PC hard disk and inserted the PCM audio samples into isochronous packets. These packets were transmitted from the PC over a 1394 bus via a Texas Instruments 1394 PCI Host Adapter card. PAVO's standalone 1394 Digital Audio Bridge received the isochronous packets, converted the payload data into a continuous stream of audio samples, and output the stereo audio signal over an SPDIF interface to a Pro Logic decoder and conference audio system. Gates used the system to evangelize Microsoft's concept of an entertainment computer (called a "Simply Interactive PC/SIPC") capable of processing high quality audio and video in a home living room environment.

Each isochronous packet carried a payload of raw PCM audio samples with no overhead information. Since the 1394 isochronous cycle rate is 8kHz, and the audio sample rate was 44.1kHz, each isochronous packet carried an average of 5.5125 samples. This average was maintained by alternating between 5 or 6 samples per packet, with an extra 6-sample packet inserted into the sequence every 40 isochronous cycles. This resulted in a total of 39 5-sample packets, and 41 6-sample packets within 80 isochronous cycles, for an average of 5.5125 samples

per isochronous cycle. Sample rate recovery in the PAVO device was implemented by a phase locked loop referenced to this average rate of data transfer.

Although simple, this method does not tell the receiving node anything about the format of the received payload. Both the sender and receiver must have a priori knowledge of what the data means, what its sample rate is, what isochronous channel is being used, and so on. This works in a closed system. However, in order to promote industry-wide interoperability, it is desirable to standardize a common isochronous packet format; only then can source and sink nodes effectively communicate using the myriad data formats available.

### **Common Isochronous Packet Format**

The first commercial implementations of 1394 were for digital video applications [4]. To facilitate interoperability between different digital video devices manufactured by different vendors, developers of these early video products devised a *common isochronous packet* (CIP) format. This CIP format was written down and published as the IEC 1883 standard [3]. Engineers at Yamaha extended CIP to support a variety of common audio formats in their Music LAN (mLAN) specification [6]. CIP is general enough to describe the format of virtually any type of audio or video stream. It is highly recommended that the CIP format be adopted by all 1394 systems likely to carry audio or video signals.

The CIP format (see Figure 3) specifies a 2-quadlet header, followed by a block of payload data. The meaning of the payload data is completely specified by the fields of the CIP header. These fields are briefly described in Table 2. The FMT field specifies the type of data carried by the isochronous packet; to date, only video formats have been standardized.

Yamaha's mLAN specification designates several new values for the FMT field for audio and MIDI data [6]. The 1394 Trade Association has approved a subset of this specification [7]. In addition, mLAN will be submitted to the AES Standards Committee Working Group SC-06-02 on IEEE 1394 for consideration.

The authors encourage audio developers to adopt the CIP format, and use the FMT audio fields defined in the mLAN specification.

### **Synchronization**

The need for synchronization in audio systems requires that each node accurately recovers an audio stream's sample rate, and provides accurate time alignment between the appropriate audio channels. Several methods have been implemented and tested to perform these functions.

## **Sync Method #1: Sample Rate Recovery Based on Cycle Timer Rate**

Every isochronous-capable node on a 1394 bus has a 32-bit counter called the CYCLE\_TIME register. The CYCLE\_TIME register increments at a rate of 24.576MHz, and is updated by the bus Cycle Master with a common value every 1250s (via Cycle Start packets). This guarantees that the CYCLE\_TIME register on every bus node increments at an identical frequency, and therefore provides a universal timing reference for generating sample rate clocks (e.g., via a phase locked loop).

The IEEE 1394 standard allows each node's 24.576MHz master clock to have a 100ppm frequency tolerance, which could result in a (worst-case) 200ppm frequency differential between any two nodes. Since each node's CYCLE\_TIME register is essentially free running between Cycle Start packets, each CYCLE\_TIME register could lead or lag the Cycle Master by up to 25ns at the end of each isochronous cycle. This translates directly to jitter in the CYCLE\_TIME register. If audio sample rate clocks are referenced to this register, jitter can occur in the audio clocks. In practice, this jitter has been filtered to less than 1ns by a proprietary phase locked loop design.

Since every node's CYCLE\_TIME register counts at the same average rate, this method assumes – indeed requires – that every audio sample rate on the bus be referenced to a common clock: the Cycle Master's 24.576MHz master clock. In professional audio systems that strive for one common sample rate, this is considered an asset.

However, this method does not easily accommodate the variable or "free running" sample rates you might find in a system carrying multiple asynchronous data streams. Sample rate conversion can be employed to resynchronize free running PCM data streams to the master clock, but "non-audio" data streams (such as Dolby Digital/AC-3) can not be altered by sample rate conversion. Therefore, they can not be placed on the bus using this synchronization method (unless, for example, the Cycle Master slaves its CYCLE\_TIME register to the clock rate of a received Dolby Digital/AC-3 signal, which is impractical). For this reason, this synchronization method is useful only to systems that allow every node to slave its sample rate clocks to the Cycle Master clock.

## **Sync Method #2: Sample Rate Recovery Based on Buffer Fill**

A second method of recovering sample rate clocks is to allow a transmitter to send audio samples at a free running rate, requiring the receiver to remove them from an input buffer at such a rate that the buffer fill level remains constant. Under this condition, the receiver has matched the sample rate of the transmitter. This method requires a relatively large buffer, which increases the latency through the system. In practice, this method has been implemented using 32-sample

buffers, adding 726Os latency to the signal. Depending on the application, this added time delay could be considered imperceptible or unacceptable.

The advantages of this method are its simplicity, low synchronization overhead (compared to time stamps), immunity to bus jitter, and support of multiple coexisting sample rates.

### **Sync Method #3: Sample Rate Recovery Based on Time Stamps**

A third method of recovering sample rate clocks utilizes time stamps. Each packet of audio data is stamped with a presentation time as it is transmitted. Each receiving node outputs the packet's first sample at the specified presentation time. Remaining samples in the packet are then output at the nominal sample rate.

Time stamps are based on the least-significant 16 bits of the CYCLE\_TIME register, as indicated in Figure 4. (Remember that this register is equalized every 125Os by the Cycle Master.) Therefore, all nodes have a common timing reference to generate outgoing or process incoming time stamps.

Time stamps occupy the 16-bit FDF field in the CIP header. The algorithm for generating a time stamp is nontrivial. The CYCLE\_TIME register's 12-bit *cycle offset* field increments at a 24.576MHz rate (see Figure 4). This field is modulus 3072, which results in a rollover every 125Os. The CYCLE\_TIME register's *cycle count* field increments every time the *cycle offset* field rolls over, at the 125Os (8kHz) rate. This, of course, corresponds to the isochronous rate of the bus; indeed, isochronous cycles are initiated every time the *cycle count* field gets bumped. Once the *cycle count* field counts up to 8000, exactly 1 second has elapsed ( $8000 \times 125\text{Os} = 1\text{s}$ ). At this instant in time, the 7-bit *second count* field increments.

Since the time stamp carries the least significant 16 bits of the CYCLE\_TIME register, its 12 least significant bits increment at a rate of 24.476MHz, while the 4 most significant bits increment at a rate of 8kHz. This provides a time stamp resolution of 40.69ns, with a range of 0 to 2ms.

Jitter under this clock recovery method is dependent on the worst-case difference between the value of the CYCLE\_TIME registers of the sender and receiver. As with the first sync method, this error is approximately 25ns over one isochronous cycle. In practice, this jitter has been filtered to less than 1ns by a proprietary phase locked loop design.

### **Time Alignment**

Time alignment involves the accurate delay of audio signals in the system. This can be implemented in several ways. Audio signals can be manually time aligned using traditional acoustical measurement tools (e.g., a TEF analyzer) and delayed using digital delay lines. Automatic time alignment requires knowledge of each audio signal's latency through the system,

adoption of a common latency (i.e., the longest latency experienced by all audio signals), and the ability to adjust digital delay lines to maintain this latency.

Of the three synchronization methods discussed, the time stamping method is best suited for implementing automatic time alignment. Since each time stamp indicates the presentation time of a packet's first sample, and since time stamps range from 40.69ns to 2ms, audio signals can be automatically time aligned within this range. If the maximum latency of an audio signal exceeds 2ms, additional information must be communicated between nodes for a coarse time delay control. It is straightforward to use asynchronous messages to write a coarse time delay value to a well-known latency register mapped in each node.

## **Video Synchronization over 1394**

Video streams on the 1394 bus are time stamped with the same process described above, allowing all audio and video signals on a bus to be synchronized within 40.69ns of each other without an external house sync [4].

## **MIDI over 1394**

The Musical Instrument Digital Interface (MIDI) is a widely used protocol used in musical instruments and multimedia computers. MIDI messages are transferred asynchronously at a data rate of 31.25kbits/s. They consist of one or more 8-bit words, with one start bit and one stop bit. MIDI data can be easily encapsulated into either asynchronous or isochronous packets. To date, no standard has emerged for bridging MIDI to the 1394 bus, however several methods are possible.

### **MIDI Method #1: Sending MIDI using Asynchronous Block Writes**

The MIDI input of a MIDI-to-1394 bridge can buffer incoming MIDI data and then write the data to another node, using an asynchronous block write transaction, whenever the node wins bus arbitration (normally once every isochronous cycle). Since 1394 asynchronous transactions are not guaranteed to have a specific latency, an unknown amount of time delay will be introduced to the MIDI data. However this latency will be approximately 1250s under normal conditions. This is equivalent to a delay of 4 MIDI bytes, which is acceptable in most situations.

### **MIDI Method #2: Sending MIDI using Isochronous Transactions**

MIDI data can be buffered and transmitted over a single isochronous channel. Since the MIDI data rate is 31.25kbits/s, and the isochronous rate is 1250s, a MIDI isochronous packet will contain 0-4 bytes of MIDI data (one quadlet). The 2-quadlet CIP header adds 200% overhead to a 1-quadlet payload, so this is an extremely inefficient method of transmitting MIDI through the

system. (Indeed, it is arguably a waste of a precious isochronous channel.) On the other hand, isochronous data is transmitted with known latency, so the MIDI data can be exchanged between nodes with minimal jitter.

## Control & Monitoring

As discussed in the previous sections, the IEEE 1394 serial bus is capable of streaming many channels of audio and video with high performance. The 1394 bus also provides an asynchronous communications channel between nodes for exchanging control & monitoring messages. Isochronous transport is not used for control & monitoring messages because these messages do not typically require guaranteed time of delivery. Even at the slowest rate possible (20% of the S100 base rate), asynchronous messages are transmitted 640 times faster than MIDI.

IEEE 1394's asynchronous communication method allows one node to read from or write to memory in another node. Therefore, the technique for controlling and monitoring parameters in a node is based on the simple concept of reading and writing public variables. Several protocols have been devised for locating parameters and arbitrating reads and writes to them. The protocol most widely used in commercial 1394 products is called *Function Control Protocol (FCP)*.

### Function Control Protocol (FCP)

The Function Control Protocol (FCP) defined by IEC 1883 – the proposed *Standard for Digital Interface for Consumer Electronic Audio/Video Equipment* [3] – provides a simple means to encapsulate device commands and responses within 1394 asynchronous block write transactions. It provides a basic mechanism for transferring command messages from one node to another.

The semantics of the messages (e.g., the commands themselves) are outside the scope of FCP, although an *Audio/Video Control Digital Interface Command Set (AV/C)* has been published by the 1394 Trade Association for use in controlling simple consumer electronics audio/video devices [5].

FCP is extensible and has thus far proven capable of providing the underlying transport services identified by the AESSC SC-10-2 Working Group on Application Layer Protocols for the AES24 protocol [8]. Further work is needed to fully test this assertion, but the authors believe FCP is very complimentary with AES-24.

A complete treatment of the FCP protocol is beyond the scope of this paper, however we will provide a brief introduction below.

## FCP Terminology

An FCP node that controls another node is called the *controller*; a node that is controlled is called a *target*. An FCP message is called a *frame*. A frame sent from a controller to a target is called a *command frame*. A frame sent from a target to a controller is called a *response frame*.

FCP commands are sent and received via memory mapped registers at well-known addresses. The register that receives a command frame is called the *command register*, and the register that receives a response frame is called the *response register*. These registers are mapped into a node's memory space as follows:

Offset base (top address of "initial register space")	FFFF F000 0000 <sub>16</sub>
Top address of command register (offset)	000 0B00 <sub>16</sub>
Top address of response register (offset)	000 0D00 <sub>16</sub>

## FCP Packet Format

FCP frames are inserted into the data field of a standard 1394 asynchronous block write packet with the format shown in Figure 5. The 4-bit CTS field specifies the Command/Transaction Set. The values defined for CTS are given in Table 3. The data payload of the FCP frame is dependent on the specific command, and may vary from 0 to 512 bytes. Since 1394 packets must be formatted into quadlets, up to 3 bytes of zero padding may be necessary in the quadlet.

## The AV/C Digital Interface Command Set

Currently, the primary command/transaction set used in 1394 products is the AV/C Digital Interface Command Set [5]. This command set has been designed for controlling typical consumer electronics devices such as video cameras, monitors, disk and tape recorder/players, and tuners. The commands are "strongly typed" for the specific functions found on these types of devices (e.g., play, record, track time).

AV/C does not provide a generic (i.e., "weakly typed") protocol for discovering and controlling arbitrary parameter sets in unknown devices. Such a protocol has been under development by the AESSC SC-10-2 Working Group on Application Layer Protocols, called AES-24 [8]. It is proposed that a reserved CTS code be allocated to the AES-24 protocol, allowing FCP to carry messages formatted under the AES-24 protocol to a wide variety of audio/video devices.

The AV/C protocol also defines the Connection Management Procedures (CMP) for plug control. These messages provide a means for the system to dynamically allocate isochronous resources to nodes wishing to send or receive audio/video signals on the bus. Implementing CMP is optional, but is recommended for implementing robust systems that adapt to changing conditions on the bus.

## Case Study: The “Papaya” Digital Audio Adapter

The concepts discussed in this paper have been implemented and tested in a device available from PAVO, affectionately called “Papaya.” Selected by Microsoft as the reference design for the Win32 Driver Model (WDM) 1394 audio class driver, Papaya provides a Plug and Play interface between audio devices and the Windows 95, Windows NT, and Memphis operating systems via IEEE 1394. Papaya is also being used as the reference design for BeOS, the “digital media” operating system developed by Be, Inc.

Papaya can be used as a stand-alone node on the 1394 bus, streaming audio to other Papayas, or to other compatible 1394 audio devices. It is essentially an IEEE 1394 legacy audio adapter, and can be used to quickly create 1394-compliant prototypes of audio equipment for demonstration and evaluation purposes.

### Papaya Functional Blocks

Figure 6 shows a block diagram of Papaya. The IEEE 1394 bus interface is implemented using Texas Instruments’ 200Mbps TSB21LV03 PHY and TSB12C01A Link chipset. An Analog Devices ADSP21061 SHARC digital signal processor performs all Application Level operations (i.e., those above the IEEE 1394 Transaction Layer). The SHARC is ideally suited for interfacing to the TSB12C01A Link IC due to its high-speed 32-bit external memory interface, with acknowledged hardware wait states.

Sample clock recovery and other synchronization operations are implemented in a XILINX XC5206 field programmable gate array (FPGA). An AKM AK4520A 20-bit stereo codec performs A/D and D/A conversion on stereo analog input and output signals. Crystal Semiconductor’s CS8401 and CS8411 SPDIF transmitter/receiver chips perform digital audio I/O. Papaya also has an on-board debug monitor, accessed via a 19.2Kbaud RS-232 interface.

Papaya’s stereo audio inputs and outputs stream over two isochronous channels on the 1394 bus. Both 44.1kHz and 48kHz sample rates are supported, and any sample word length between 16 and 32 bits (including 32-bit IEEE floating point) can be used. Data from the A/D converter and SPDIF input are multiplexed on to one isochronous channel; data received from the second isochronous channel is fed to the D/A converter and SPDIF transmitter. In this fashion, Papaya streams up to 8 channels of full duplex digital audio over the 1394 bus.

Papaya’s SPDIF I/O supports both “audio” and “non-audio” formats, allowing it to handle PCM audio data as well as encoded data (e.g., Dolby Digital/AC-3). The SPDIF input can be sample rate converted (through an on-board Analog Devices AD1890 sample rate converter) to resynchronize the incoming SPDIF PCM data stream to the system clock.

## **Papaya Synchronization**

Papaya supports all three modes of synchronization discussed earlier. Because the time stamping method is preferred by most 1394 developers, it is the default synchronization mode. Approximately 10% of the FPGA's resources are required for implementing the synchronization circuitry. Exact details of this implementation are proprietary.

## **Papaya Packet Formats**

Papaya supports two primary isochronous streaming protocols: zero-overhead and CIP. The zero-overhead format transmits raw PCM samples in an isochronous packet with no additional header information. As discussed earlier, this method is the most efficient, but demands that both the sender and receiver have a priori knowledge of the exact content of the packet. The CIP format supported by Papaya is compatible with the Audio and Music Protocol developed by the 1394 Trade Association, as well as Yamaha's mLAN specification.

## **Control & Monitoring**

Papaya device parameters are controlled with FCP commands using the AV/C command set. FCP commands may be sent/received by Papaya while the node is streaming isochronous (audio) data.

At the time this paper was submitted for publication, efforts were underway to implement the AES-24 application protocol on Papaya over FCP.

## **Summary**

IEEE 1394 offers the digital media industry a robust, cost effective, extensible high speed interconnect for a wide variety of applications. The authors have successfully implemented several prototype audio systems, and have proven the viability of the concepts presented in this paper.

Efforts undertaken by the Audio Engineering Society AES-10 committee to standardize control & monitoring protocols for audio equipment are complimentary to IEEE 1394, and all indications are that the AES-24 protocol can be encapsulated within standard 1394 FCP asynchronous packets.

In 1997 the AES Standard Committee established the SC-06-02 Working Group on IEEE 1394 to establish standards for transmitting audio over the 1394 bus. The authors encourage all interested professionals to join this group and participate in the process of creating a unified standard for the industry.

## Acknowledgements

The authors gratefully acknowledge Brian Karr of PAVO for his review and critique of early drafts of this paper. We have also been engaged in many thought-provoking conversations with Burke Henehan of Texas Instruments, and Yoshiyuki Sawada and Jun-ichi Fujimori of Yamaha.

## References

- [1] IEEE Std 1394-1995, Standard for a High Performance Serial Bus
- [2] ISO/IEC 13123:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses
- [3] IEC 1883, proposed standard for Digital Interface for Consumer Electronic Audio/Video Equipment
- [4] HD Digital VCR Conference, Specifications of Consumer-Use Digital VCR's using 6.3 mm magnetic tape (December 1995)
- [5] 1394 Trade Association, AV/C Digital Interface Command Set, Version 2.0, (April 1997)
- [6] Yamaha Corporation, proposal for Audio and Music Protocol, Draft Version 0.32, (August 1996)
- [7] 1394 Trade Association, Specification for Audio and Music Data Transmission, Version 0.91a, (March 97)
- [8] AES24-1-1995, AES standard for sound system control-- Application protocol for controlling and monitoring audio systems-- Part 1: architecture

## World Wide Web Sites

**1394 TA:** <http://www.firewire.org/>

**IEEE Standards:** <http://standards.ieee.org/catalog/it.html>

**PAVO:** <http://www.pavo.com/ieee1394/ieeehome.htm>

**Apple Computer:** <http://www.devworld.apple.com/dev/FireWire/>

**Sony:** <http://www.sel.sony.com/semi/firewire.html>

**Molex:** <http://www.molex.com/other/news1394.html>

**LTK Cable:** <http://www.ltkcable.com/1394.html>

**Adaptec:** <http://www.adaptec.com/firewire/>

**Philips:** <http://www.semiconductors.philips.com/multimedia/designercorner/bussystems/ieee/>

**Texas Instruments:** <http://www.ti.com/sc/docs/msp/1394/1394.htm>

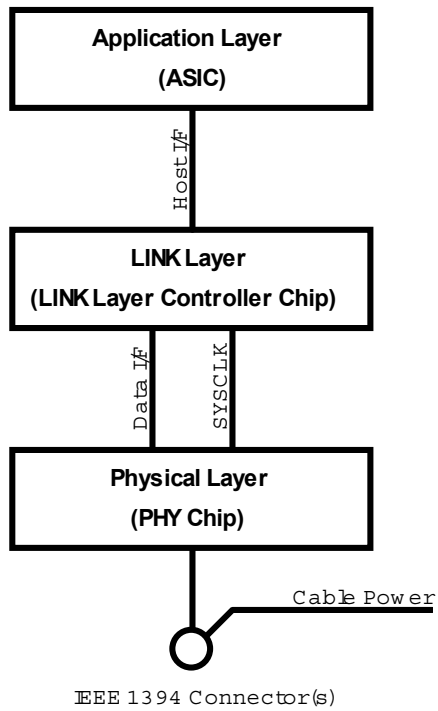


Figure 1: Layers in an IEEE 1394 Implementation

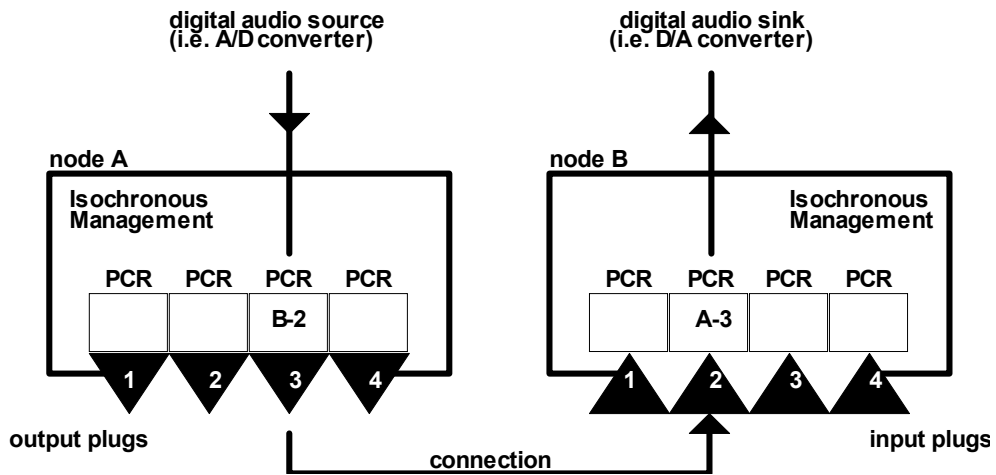
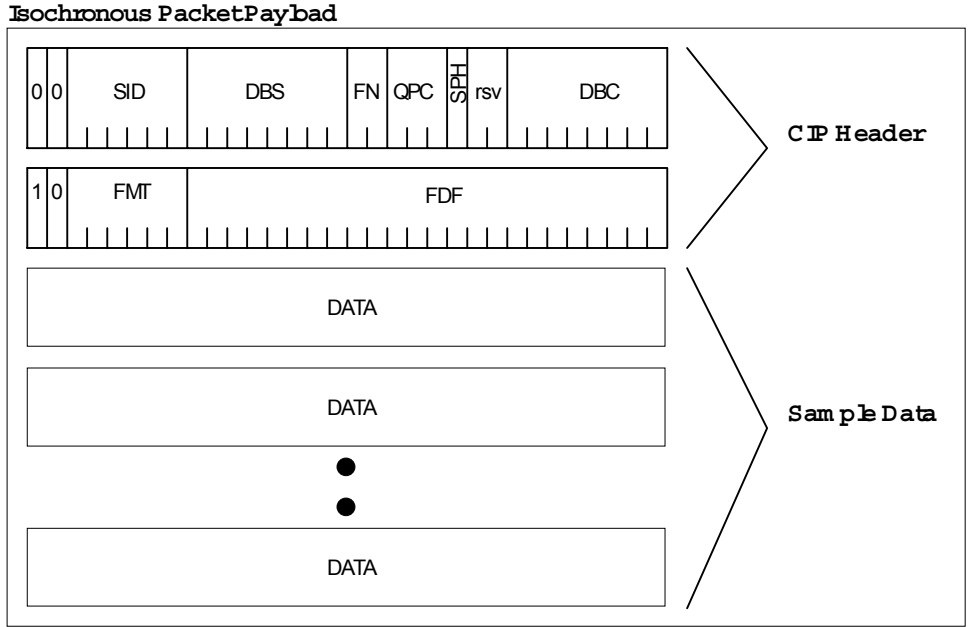


Figure 2: Plug Control Model



**Figure 3: Common Isochronous Packet (CIP) Format**

SID	<i>Source ID</i>	Source node ID of transmitter (packet originator).
DBS	<i>Data Block Size</i>	Size of data block in quadlets. Can not exceed 256 quadlets per packet.
FN	<i>Fraction Number</i>	Source packet subdivision is optionally used to improve bus utilization time. A data block may be split into 1, 2, 4, or 8 packets corresponding to FN = 00, 01, 10, 11 respectively.
QPC	<i>Quadlet Padding Count</i>	Quadlet padding is used when FN does not equal 00.
SPH	<i>Source Packet Header</i>	Flags that the source packet has its own header.
rsv	<i>Reserved</i>	
DBC	<i>Data Block Count</i>	This field is incremented every data block, providing a sequence number for the data block. When packet division is employed (FN > 00), the lower bits of this field indicate the offset value of the first data block in the CIP packet.
FMT	<i>Format ID</i>	Specifies the type of data contained in the data block. Values have been defined for DVCR, MPEG-2, audio, etc.
FDF	<i>Format Dependant Field</i>	The meaning of this field is dependant on the value of format ID. For many audio and video formats, this field carries a time stamp based on the lower 16 bits of the node's CYCLE_TIME register when the data is to be presented to a D/A converter.

**Table 2: CIP Fields**

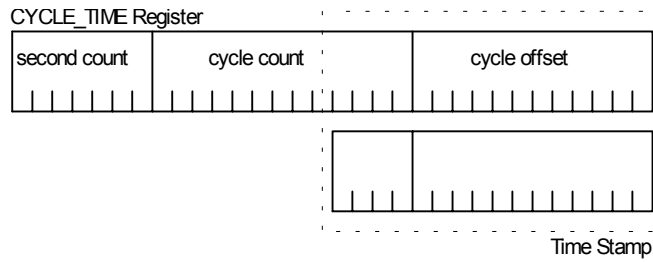


Figure 4: Time Stamp derived from CYCLE\_TIME Register

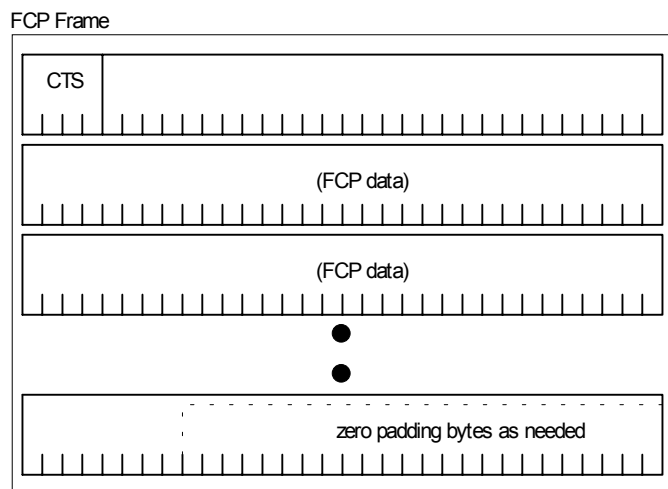
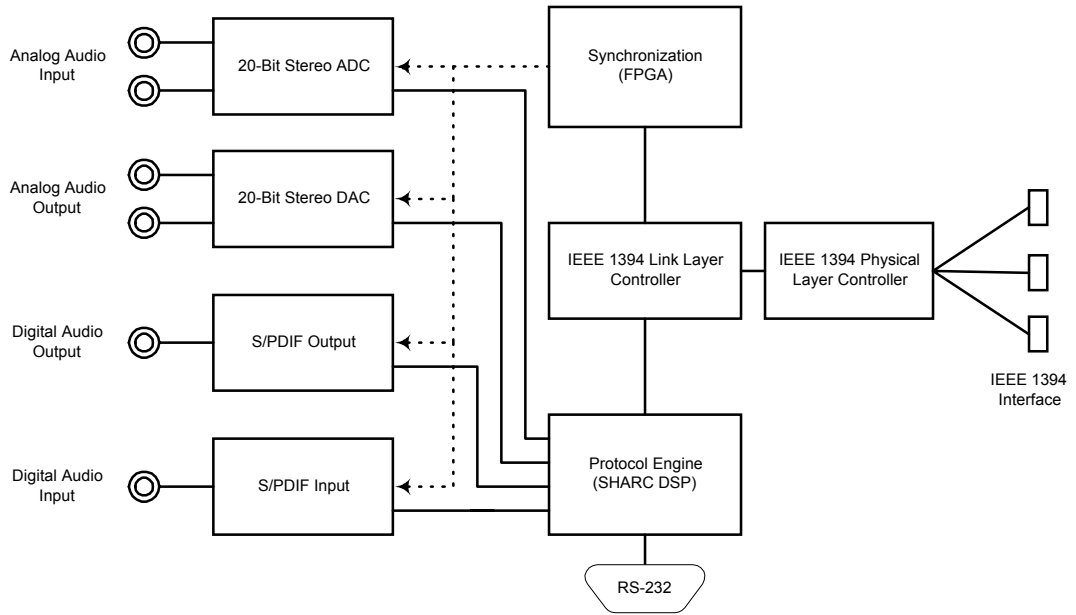


Figure 5: FCP Frame

CTS Code		CTS		
msb	lsb	msb	lsb	CTS
0	0	0	0	AV/C
0	0	0	1	Reserved for CAL
0	0	1	0	Reserved for EHS
0	0	1	1	(reserved)
1	1	0	1	
1	1	1	0	Vendor Unique
1	1	1	1	Extended CTS

Table 3: CTS Command/Transaction Set



*Figure 6: Block Diagram of Papaya Audio Adapter*