# IEEE 1394 SERIAL BUS CONTROLLER

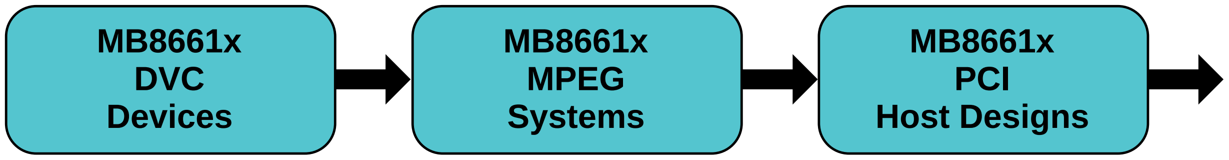
November 1996

## DESCRIPTION

Fujitsu has developed a 1394 controller LSI core that combines the physical (PHY) and link layers which are the two fundamental components of all 1394 controller LSIs. The Fujitsu implementation is an all CMOS design and is fabricated in standard cell process to facilitate quick turn-around time for derivative products. Fuijtsu intends to use the basic elements of this 1394 core to produce application specific controllers for the key emerging markets. These include digital cameras, DVD, set-top-box, file systems, and video editing.

The first commercially available LSI to be based on the MB8661x core will  be designed for DVC applications using the AV/C protocol and the second product will be designed for MPEG applications. In both products the PHY and LINK layers remain almost identical, with the exception being the size of the send / receive FIFO (The larger MPEG packets require the FIFO to be three times larger than is required for the DVC mode). The principal difference between devices will be the on-chip logic and firmware that will execute protocol specific operations such as time stamping CIP header management. The AV/C support function is implemented via a bank of dedicated registers that are activated through the setting of a mode pin. When the AV/C mode is not selected, the chip is controlled through a bank of default registers (Register Bank 0) that provide low level control of both Asynchronous and Isochronous operations. This flexibility enables the device to be used in applications other than AV/C including audio only applications, video only and those applications with very unique programming requirements.

The MB8661x core is presently designed to run at 200Mbit/sec, however, the first commercially available versions will be targeted toward low cost applications such as digital cameras and digital VCRs. Fujitsu will introduce 200Mbit/sec and 400Mbit/sec products later in 1997. Included in the mix will be products designed for set-top-boxes, and PCI based host applications that will address both motherboard and adapter card requirements.



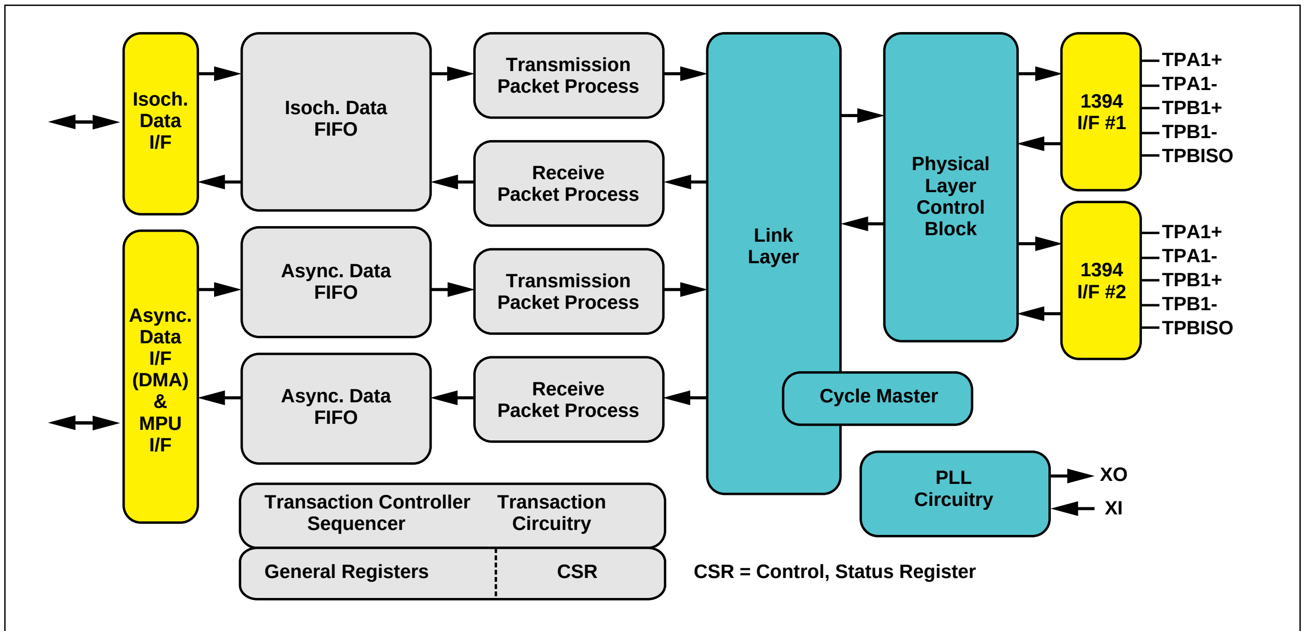
# IEEE 1394 SERIAL BUS CONTROLLER

## FEATURES

Standard features that can be found in the MB8661x core and derivative products include:

- On Chip transaction sequencer and application specific firmware
- Multiple CSRs for Isochronous Resource Manager (IRM)
- 32-bit CRC generation and check (disable function)
- Separate Asynchronous and Isochronous ports
- Separate Asynchronous and Isochronous FIFOs
- Internal clock generator via on-chip PLL
- 3.3V operation

The block diagram below illustrates the primary functional blocks of the MB8661x core. The buffer and FIFO capacities may be modified depending on the target application. In addition, the number of ports supported may also be increased from two to three.



**MB8661x BLOCK DIAGRAM**

Printed in the U.S.A.

FUJITSU MICROELECTRONICS, INC.
3545 North First Street, San Jose, California 95134-1804
Customer Response Center (CRC)
CRC Tel: 1-800-866-8608, CRC Fax: 408-922-9179
http://www.fujitsumicro.com

# *What is IEEE Standard 1394-1995?*

IEEE 1394-1995 is a *Standard for a High Performance Serial Bus*. The bus provides a high-speed, low-cost, versatile means of interconnecting a wide variety of computer peripherals and consumer electronic devices. It supports a component structure and bus topology that permits ease of installation for all forms of multimedia equipment. The interface consists of a six-wire shielded cable containing three sets of twisted-pair conductors: two sets for information transfer operating in differential mode, and one set for power and ground, nominally 8–40 VDC at 1.5 A.

With processor speeds operating at 200 MHZ and fixed-disk drives containing multiple gigabytes, I/O requirements for personal computers, workstations, and servers are increasing rapidly. High-speed I/O processor interfaces that have been developed for mainframes and supercomputers are too expensive and too bulky for smaller computer systems. Accordingly, much effort has been expended in developing a low-cost alternative for a small computer system interface—the result is the IEEE 1394-1995 Serial Bus Interface.

The bus multiplexes a variety of different types of digital signals, such as, compressed video, digitized audio, and device control commands over a high-bandwidth serial interface. With a single, standardized multimedia connector, the bus structure enables progressive evolution and cost effectiveness to meet key emerging multimedia demands.

The 1394 standard specifies a set of three layers of protocol which standardize the method in which the host system interacts with peripheral devices. The *physical layer* defines the electrical signals of the various serial bus media and provides bus arbitration; the *link layer* describes the transmission of data packets and provides addressing, data checking, and data framing; the *transaction layer* defines the request-response protocol.

The serial bus architecture is defined in terms of nodes. A node is an addressable entity attached to the bus. A module, which is defined as a replaceable device, contains one or more nodes; a node contains one or more units, such as, a processor, an I/O device, or a memory unit. Each node has one or more ports, in which the port connects to either a cable or backplane and provides one end of a physical connection with another node. Modules are a physical packaging concept, whereas, a node is a logical addressing concept. The cable topology allows nodes to be connected in either a branch or a daisy-chain configuration.

The 1394 serial bus architecture limits the number of nodes on any bus to 63, but supports a multiple bus hierarchy by means of bus bridges. With multiple buses, approximately $2^{16}$ nodes can be addressed by the serial bus addressing structure.

Data transfer services supports two basic modes for transferring data between nodes: asynchronous and isochronous. Asynchronous mode transfers a variable amount of data and several bytes of transaction layer information as a packet to an explicit address and waits for an acknowledgment. The transmission of a packet and receipt of an acknowledgment is called a subaction. If the packet was received correctly, then the process continues and the next packet is sent. If an error occurred during transmission or an acknowledgment was not received, then the packet is retransmitted or an error procedure is invoked. Asynchronous transmission is used for devices that have no fixed data rate requirement.

Asynchronous transfer can be concatenated without requiring a bus arbitration sequence if the node already controls the bus as it would if the subaction is the transaction response corresponding to the immediately preceding acknowledgment. The time for a subaction transaction may vary depending on bus activity during bus arbitration and upon the time to receive an acknowledgment, which is device dependent. Therefore, the timing for transferring packets and receiving acknowledgments is not predictable. Since packet transmission does not recur at exactly the same periods, this mode is called asynchronous.

Isochronous mode transfers a variable amount of data in fixed-size packets at regular intervals. This mode uses simplified addressing and requires no acknowledgment. Since transmission recurs at regular intervals, the architecture supports a guaranteed high bandwidth. Isochronous transfer is used primarily in applications requiring real-time data transfer capability for time-sensitive applications, such as, audio and video devices. Isochronous transmission increases system throughput by not requiring an acknowledgment and also by generating a smaller gap between packet transmissions than a corresponding asynchronous subaction gap. In addition to guaranteed bandwidth, isochronous transmission can reduce packet overhead by transmitting concatenated packets with a single command. Since packet transmission is uniform in time and recurs at regular intervals, this mode is called isochronous.

Bus bridging is an important concept of the 1394 interface. The serial bus limits the number of nodes to 63; however, using bus bridging to produce a multiple bus system increases the addressing capability to approximately $2^{16}$ nodes. A bus bridge is a unit that provides a physical connection between two buses in the bus hierarchy. That is, the 1394 serial bus interface can be used to bridge together other 1394 buses, or buses with different architectures. For example, a bus bridge can bridge together the 1394 serial bus with a Future Bus +; a second bus bridge can bridge together the same 1394 serial bus with a Scalable Coherent Interface. Each bus bridge monitors the highest bus in the hierarchy to detect a transaction for a node on its bus. When a bridge receives a packet addressed to a node on its bus, it forwards the packet to the appropriate node.

## History

The IEEE 1394 bus architecture was developed in 1986 by Apple Computer as a means to simplify the network of interconnecting cables in a personal computer system and to provide a mechanism to transfer blocks of real-time data at high speed. The proposed standard (P1394), which was derived from Apples's original *FireWire* design, was adopted as an industry standard by the Institute of Electrical and Electronics Engineers (IEEE) in December, 1995.

Apple eventually licensed the architecture to several other personal computer and consumer electronics companies. In 1994, these companies formed the 1394 Trade Association to support the development of computer and consumer electronics systems that could be easily interconnected using the 1394 architecture. The group is chartered with the promotion of 1394 technology into the consumer, computer, and peripherals markets to provide a uniquely coherent I/O structure.

The 1394 bus has gained wide acceptance by several major manufacturers, including Adaptec, AT&T, Apple, Compaq, Fujitsu Microelectronics, Intel, IBM, Matsushita, Microsoft, Philips, Seagate, Sony, Texas Instruments, and Yamaha, among others.

## Advantages

Data transfer rates of 100, 200, and 400 megabits per second (Mbps) are currently supported by the bus architecture. Proposed future data rates will exceed 1 gigabits per second (Gbps). Additionally, all future speed increases will be downward compatible with current speeds. This will permit a scalable topology capable of supporting multiple data transfer rates over the same physical interface. All devices shall be capable of operating at 100 Mbps. Also, all devices capable of operating at 200 Mbps shall also be capable of operating at 100 Mbps and all devices capable of operating at 400 Mbps shall be capable of operating at both 200 and 100 Mbps.

The 1394 interface provides a low-cost, entry-level architecture for computer peripherals while maintaining easy adaptability for future peripheral upgrades. One of the inherent benefits of the 1394 interface is the use of serial transmission, which greatly reduces the cost of cables and connectors. Parallel interfaces require more wires, which means larger, more expensive connectors with a corresponding increase in required shielding to prevent crosstalk between signals. Skew between signals is also a problem on a multiconductor parallel cable. Skew between signals on the 1394 interface, however, is minimized by using only two signals to transfer information between units. Although computers are becoming physically smaller, they are expanding in terms of computing power and I/O requirements. Handheld, pocket-size, and laptop computers have little room for connectors, yet require high data rates.

The plugs and sockets are mechanically identical at both ends of the cable, providing ease of installation for additional devices. This single connection is a major attribute of the 1394 interface, providing simple connectivity with high bandwidth for multimedia applications. This global interconnect feature eliminates the need for multiple I/O interconnects enabling I/O port integration and board space consolidation.

Of equal importance is the *plug-and-play* characteristic which allows live connection and disconnection of devices without loss of data integrity. This makes it possible to connect and disconnect peripherals without having to power down or reconfigure the computer system. The 1394 protocol automatically reconfigures the network whenever a node is added or removed from the interface.

The serial bus complements a parallel bus by allowing modules to communicate even though they are physically dispersed in the same enclosure. A serial bus also permits communication between different modules in a system that are located in different enclosures and connected to different parallel bus backplanes. A redundant data path produced by a serial bus increases fault tolerance. The system can use the serial bus to isolate and diagnose errors without recourse to the failed parallel bus. Complementing a parallel bus with a serial bus provides a data path for low-cost system modules that do not need the full bandwidth capability of a parallel bus.

The serial bus can also be used as an effective, low-cost peripheral interconnect. the compact cable and connector allow data transfer bandwidths comparable to existing bus structures. An added advantage is architectural compatibility with parallel bus architectures, yielding a lower communica tions overhead than I/O-specific interfaces.

## *Applications*

The 1394 serial bus can integrate diverse subsystems into a single cohesive unit. It can link together previously incompatible systems, such as, Asynchronous Transfer Mode (ATM) devices, Digital Video Consortium (DVC) devices, Digital Video Disks (DVD), Motion Picture Expert Groups (MPEG) systems, Musical Instrument Digital Interface (MIDI) devices, Peripheral Control Interface (PCI) systems, direct access storage devices (fixed-disk drives and CD-Rom disk drives), sequential access storage devices, printers, scanners, and other multimedia devices.

Applications that benefit from the 1394 interface include desktop publishing, document imaging, video editing, video conferencing, multichannel digital audio mixing, keyboard synthesizing, and other multimedia applications.

The 1394 interface provides an alternative bus structure for a parallel backplane bus and adds increased flexibility to the bus architecture. Although the many modules (containing CPU's, I/O devices, memory, or other units) that comprise a computer system might operate on different backplane bus standards, they must operate coherently for total system performance.

## ***Addressing***

The 1394 serial bus follows the control and status register (CSR) architecture as defined in IEEE Standard 1212-1994 for 64-bit fixed addressing. The high-order 16 bits of each address represents the node identification. The node identification is partitioned into two fields: the high-order 10 bits specify a bus identification field and the low-order 6 bits specify a node identification field. Each field reserves the value of all 1s for special purposes. Therefore, addressing capability is specified for 1023 buses with 63 nodes per bus, providing 64k independently addressable nodes.

The low-order 48 bits (256 terabytes) provide address space for:

- Initial memory space
- Private space: reserved for node-local uses
- Initial register space: 2048 bytes allocated as follows:
    - 512 bytes reserved for core CSR architecture resources
    - 512 bytes reserved for serial bus registers
    - 1024 bytes reserved for a ROM ID area
- Initial units space: reserved for node-specific resources providing 256 MB of CSR address space

## Packet Formats

The 1394 standard defines byte and multiple-byte sizes as follows:

|  |  |
|---|---|
| 8 bits: | byte (Bits 0-7 LSB) |
| 16 bits: | doublet (Bits 0-15 LSB) |
| 32 bits: | quadlet (Bits 0-31 LSB) |
| 64 bits: | octlet (Bits 0-63 LSB) |

For 32-bit quadlet registers, byte 0 is the high-order byte; for a 64-bit quadlet-register pair, the left quadlet is the high-order quadlet. The high-order field is transmitted first, and within a field, the high-order bit is transmitted first. Serial bus packets consist of a sequence of quadlets, in which the high-order quadlet is transmitted first.

The cable physical layer sends and receives a series of short packets at 100Mbps (base rate) which are used for bus management. Each packet consists of 64 bits (two quadlets). The second quadlet is the 1s complement of the first quadlet. If the first quadlet does not match the logical inverse of the second quadlet, then the packet is ignored. The cable physical layer packets are the self-identification packet, the link-on packet, and the physical configuration packet.

***Self-Identification Packet*** One to four self-identification packets are transmitted at the base rate of 100 Mbps during the self-identification phase of bus arbitration. Information contained in these packets consists of the self-identification packet identifier, the physical node identifier of the sender (source), the speed capabilities of the sender, the worst-case repeater data delay, power consumption and source characteristics, port status, and other pertinent information regarding the characteristics of the sending node, including an indication whether another self-identification packet will immediately follow the current packet. The number of packets is proportional to the number of ports on the bus.

***Link-On Packet*** The link-on packet is used by those nodes that do not automatically power-on their link layer circuitry. Upon receipt of a link-on packet, the node will apply power to its link layer. Information contained in the link-on packets consists of the link-on packet identifier and the physical node identifier of the destination node.

***Physical Configuration Packet*** This packet is used to optimize serial bus performance for particular configurations. This packet can set the gap size to a smaller value for higher system throughput and force a particular node to be the root node. Both actions operate on remote nodes. Information contained in the physical configuration packet consists of the physical configuration packet identifier, the physical node identifier of the destination node, and a gap count field.

A primary packet contains at least two quadlets: a packet header and an optional data block. The packet header consists of one or more header quadlets followed by a header cyclic redundancy check (CRC) character for error detection. The header CRC is calculated on the packet header only. A node will not initiate any action if the header CRC indicates a transmission error.

Cyclic redundancy checking is used primarily for serial data transmission. A CRC character is calculated by the sending device during data transfer, then appended to the data. The same algorithm is used by the receiving device to calculate a CRC character. If the resulting CRC character is different than the one received, then an error has occurred during transmission. Primary packets with a data block contain one or more quadlets of data followed by a data CRC. The data CRC is calculated on the data block quadlets only and does not include any of the packet header quadlets.

The first quadlet of every packet header contains a 4-bit transaction code which specifies the packet format and the transaction to be performed such as, write request, write response, read request, and read response. The transaction code field also indicates whether the subaction is asynchronous or isochronous.

## Asynchronous Packets

An asynchronous packet is a sequence of quadlets that are aligned on a quadlet boundary. All asynchronous packets contain a packet header followed by an optional data block and conform to the format of a primary packet. The length of an asynchronous packet is at least four quadlets.

## Packet Header

The major fields (components) contained in the packet header of asynchronous packets are summarized below. A transaction is defined as a request and a corresponding response.

***Destination Identification*** A 16-bit field that specifies the node identification of a receiving node. The destination identification field is partitioned into two smaller fields: the upper 10 bits specify a destination bus identification and the lower 6 bits specify a node identification.

***Destination Offset*** This field specifies the low-order 48 bits of the destination node identification (address).

***Source ID*** A 16-bit field that specifies the source node. This component is also partitioned into two subfields with the same designations as the destination identification field.

***Transaction Label*** A 6-bit field specifying a unique tag for each outstanding transaction for a node. The transaction label sent in a request subaction is also used as the transaction label in the response subaction. Each transaction is uniquely identified by the source identification of the responding node, the destination identification of the requesting node, and the transaction label.

***Retry Code*** A 2-bit retry code that specifies whether this packet is a retry attempt and the protocol to be followed by the destination node. The retry protocols define the procedure used to respond to an inbound packet when the node is busy. The retry protocols apply to both request packets and response packets.

***Transaction Code*** A 4-bit field that specifies the packet format and the type of transaction to be performed. The transaction codes indicate whether the transaction is asynchronous or isochronous, and whether the subaction is a request or response for a read or write operation.

***Priority*** A 4-bit field that applies only to the backplane environment and specifies the arbitration priority. A value of $0000_2$ corresponds to the fair arbitration mechanism; a value of $1111_2$ corresponds to the highest priority and disables fair arbitration.

***Data Length*** A 16-bit field that specifies the number of bytes to be transferred during a subaction.

***Header CRC*** This 32-bit field contains the CRC character for the header quadlets. The CRC character is generated and checked using a standard algorithm as defined in IEEE standard 802.

## *Data Block*

The fields in the data block are summarized below.

***Data Block Payload*** This field contains the data to be transferred during a subaction. If the data length specifies a length that is not a multiple of four, then the transmitting device will pad the data field with bytes containing $00_{16}$ so that the data field is composed of full quadlets.

***Data CRC*** This 32-bit field contains the CRC character for the data block payload. The CRC character is generated and checked using a standard algorithm as defined in IEEE standard 802.

## *Packet Types*

The types of asynchronous packets are listed below for different operations. Each packet contains an appropriate CRC character.

***Asynchronous Packet with no Data Payload*** This packet contains only a packet header as previously described; no data block follows the header.

***Read Request for a Data Quadlet*** This packet requests a single data quadlet from the specified destination address.

***Read Response for a Data Quadlet*** This packet is sent only in response to a read request for a data quadlet. The packet contains a packet header only, and the following main fields: destination identification, source identification, and the requested data quadlet located in the fourth quadlet of the packet. A 4-bit response code is also included in the packet header. The response code indicates the result of the command request such as, successful completion of the command, a hardware error, incorrect address in the destination node, and other status information.

***Read Request for a Data Block*** This packet requests a block of data from a specific destination address. The packet specifies the destination identification, the destination offset, the source identification, and the data length.

***Read Response for a Data Block*** This packet is sent only in response to a read request for a data block. The packet contains a packet header consisting of the following main fields: destination identification, source identification, and data length. A 4-bit response code is also included in the packet header as previously defined. The data block follows the packet header and contains the requested data with padding bytes of $00_{16}$, where necessary.

***Write Request for a Data Quadlet*** This packet requests that a data quadlet be written to the specific destination address. The packet contains the following main fields: destination identification, destination offset, and source identification. The data quadlet to be written to the destination address is located in the fourth quadlet.

***Write Request for a Data Block*** This packet requests that a data block be written to the specified destination address. The packet contains a packet header consisting of the following main fields: destination identification, destination offset, source identification, and data length. The data block follows the packet header and contains the data field for the destination padded with bytes of $00_{16}$, where necessary.

***Write Response*** This packet is a response to a write request for a data quadlet or for a data block. The packet contains the following main fields: destination identification, source identification, and a response code.

## *Isochronous Packets*

Isochronous packets are a sequence of aligned quadlets and conform to the rules for a primary packet as discussed previously. An isochronous packet consists of a packet header and an optional data block. A *cycle start* packet initiates the start of an isochronous cycle on the bus and is sent only by a cycle master. The cycle start packet contains the following primary fields: destination identification, destination offset, source identification, transaction code which specifies a request to start an isochronous cycle for a quadlet or block of data, and other fields that are specific to isochronous data transfer.

Only one type of isochronous packet is defined for the serial bus. The isochronous packet contains a packet header and a data field. The packet header consists of two quadlets; the first contains a data length field, a channel number field, and a transaction code field; the second is the header CRC character. The data field contains the data to be transferred. If the data length specifies a length that is not a multiple of four, then the sending device pads the remaining bytes with $00_{16}$, so that the data field is composed of full quadlets.

## *Acknowledge Packets*

An acknowledge packet is sent by a destination device in response to a non-isochronous primary packet. The acknowledge packet is one byte in length. The high-order four bits contain the acknowledge code; the low-order four bits contain the acknowledge parity which is the 1s complement of the upper four bits. The 16 acknowledge codes specify conditions that occurred during a packet transmission such as, a packet accepted with or without a response subaction, a packet not accepted and the reason for non acceptance, a data CRC error, or the number of bytes in the data block did not match the number of bytes indicated in the data length field.

## Physical Layer

The physical layer has three primary functions: data transfer, arbitration, and provision for the electrical and mechanical interface. The physical layer translates the data transfer requirements from the link layer into electrical signals to drive the bus and receive information from the bus. It also provides an arbitration service to permit a node to gain access to the bus and to guarantee that only one node at a time is transferring data. The physical layer also provides the mechanical interface for the serial bus to the backplane media. Although the cable and backplane environments have different physical layers, both provide a *data strobe* concept for encoding data bits and both provide a method to ensure fair access to the bus.

### Data Transfer

During packet transmission, only one node is transmitting data on the bus, allowing the entire media to operate in half-duplex mode, which allows transmission in either direction, but not at the same time.

***Data Encoding*** During data transfer, two signals are active on the bus: Data and Strobe. Information on the data line is encoded in the nonreturn-to-zero (NRZ) method, where a binary 1 is represented as a high level and a binary 0 is represented as a low level. For long strings of consecutive 1s or 0s, the NRZ encoding method is not self-clocking, because transitions occur only on 0-to-1 and 1-to-0 boundaries.

***Data-Strobe Encoding*** The Data signal is accompanied by the Strobe signal which changes state whenever two consecutive bits on the Data line are the same. This ensures that a transition occurs for each bit cell on either the Data signal or the Strobe signal. Thus, the Data-Strobe encoding method is inherently self-clocking. A clock signal, which provides a transition for each bit period can be derived by Exclusive-ORing the Data and Strobe signals. This encoding scheme increases the skew tolerance for data transferred on the serial bus.

### Cable Physical Layer

A node is an addressable device on the serial bus. The cable environment is a network of nodes connected by point-to-point (non-bussed) links. The physical connection to a node consists of a port on each node’s physical layer and the associated cable. A node can have multiple ports which allows branching between nodes; however, nodes must be connected together in an acyclic manner; that is, with no loops. The electrical components in the cable physical layer takes data received on one port, resynchronizes it to a local clock, and transmits the data from all of its other ports.

The cable physical layer consists of four logical units: ports, which provide the cable interface; arbitration logic, which provides access to the bus; a resynchronizer, which receives the data and resynchronizes it to a local clock; and an encoder, which encodes the data in the data-strobe format.

***Tree Topology*** Cable configuration is accomplished in three phases: bus initialization, tree identification, and self identification. During cable configuration, a tree topology is constructed in which each node is assigned a physical node identification. Whenever a new node is connected to the bus, a bus reset signal forces all nodes into a special state that clears all previous topology information. The connection status of each port is stored internally by the physical layer. Nodes with no connected ports are *isolated*; nodes with a single connected port are *leaves*; nodes with two or more connected ports are *branches*.

After bus initialization is complete, the tree-identification process begins and translates the network into a tree topology, where one node is designated as the *root* node. A port that is connected closer to the root is called a *parent*; a port that is connected farther from the root is called a *child*. For example, assume that a branch node has three ports, where one port is connected to a port of the root node and two ports are connected to two different leaf nodes through their respective ports. The port that is connected to the root node is designated as a parent; each of the two ports that connect to the leaf nodes are designated as a child. Unconnected nodes do not participate in the tree-identification process. When the tree-identification process is complete, the structure will have each connected port labeled as either a parent or a child.

Selection of the root node is not topology dependent?any node can be a root node. The root that has all of its connected ports identified as children becomes the root. The only requirement is that the isochronous cycle master must also be the root, since the root node has the highest priority.

Self-identification occurs next in which each node selects a unique physical identification, or node address. The self-identification process uses a deterministic procedure in which a node selects a physical identification number depending upon its location in the tree structure. The root node gives control of the bus to the node attached to its lowest numbered connected port. When the node on that port and all of its children have been identified, a signal is sent to the root node indicating that the identification process is finished. The root then passes control to its next highest numbered connected port and the process repeats. The physical identification number is simply the count of the number of times that a node passes through the state of receiving self-identification information from other nodes before sending its own identification number. For example, a branch node has a higher identification number than any of its leaf nodes, because each leaf node, as a child, has the opportunity to establish its physical identification before the branch node, as a parent.

The cable architecture supports data transfer rates of 100, 200, and 400 Mbps, where 100 Mbps is designated as the base rate. Each node broadcasts its speed capabilities as part of its self-identifica tion packet. Each node exchanges speed information with its parent at the end of the node's self-identification process. This speed capability is recorded by both nodes. Since a node has already received self-identification information from each of its children before sending its own self-identification information to its parent, the node has a complete record of the speed capabilities of all the nodes attached to each of its connected ports.

***Cable Media Interface*** The cable media interface is the implementation of a physical connection and contains three major parts: the electrical interface, the cable connectors, and the cable media. The electrical interface contains three sets of well-shielded, twisted-pair conductors. Two sets are used for low voltage, low current bidirectional differential signals labeled TPA, TPA*, TPB, and TPB*, where TPA* and TPB* are the opposite phase of TPA and TPB, respectively . The third set of conductors is labeled VP and VG and provides the power and ground needed by the physical layer (ports) to receive, resynchronize, and transmit signals. The cable connectors are small and rugged, providing six electrical contacts plus a shield.

## Media Attachment

A module consists of one or more nodes (addressable devices) with one physical layer per node and one port per module. If more than one node occupies a module, the nodes share the same bus transceivers. The number of modules on a backplane will not exceed the maximum number specified for the host backplane and the number of nodes on a backplane will not exceed 63. Modules should be designed to support fault detection. Within a module, the bus transceivers should be packaged separately from the parallel backplane.

## Gap Timing

A gap is a period of time during which the bus is idle. There are four types of gaps: a subaction gap, an acknowledge gap, an arbitration reset gap, and an isochronous gap. A subaction refers to a link layer operation consisting of arbitration, packet transmission, and acknowledgment. A *subaction gap* separates all nonconcatenated asynchronous subactions and has a duration of eight arbitration clock times (approximately 182.76 ns). An *acknowledge gap* occurs between the end of a packet and the start of an acknowledge and has a duration of 20 arbitration clock times (approximately 406.9 ns). An *arbitration reset gap* occurs when the fairness interval starts and has a duration of 32 arbitration clock times (approximately 651.04 ns). An *isochronous gap* occurs before the start of arbitration for an isochronous subaction. An isochronous cycle begins after a cycle start packet is sent, and ends when a subaction gap is detected. An isochronous cycle begins approximately every 125 ms.

## Arbitration Sequence

The arbitration sequence uses a unique 6-bit arbitration number for each module, which is the same as the node's physical identification. The arbitration number is preceded by a 4-bit priority level for use by asynchronous-mode devices; priority levels are not used in isochronous arbitration. The lowest priority ($0000_2$) is used for fair arbitration and the highest priority ($1111_2$) is used for cycle start requests. The remaining 14 priority levels ($0001_2$–$1110_2$) are used for urgent arbitration, which allow nodes to obtain a larger portion of the bus bandwidth. All nodes arbitrating for fair arbitration are allowed one fair access to the bus during each fairness interval. A *fairness interval* is a group of contiguous transfers during which each competing node using the fairness protocol obtains a single transfer. A fairness interval consists of one or more subactions separated by subaction gaps and followed by an arbitration reset gap. The delimiters of the fairness interval are the arbitration reset gaps.

## Arbitration

It is possible for more than one node to attempt to access the bus simultaneously. Therefore, a node must arbitrate for the bus in order to gain access. When a node returns an acknowledge packet, however, arbitration is not necessary since there is no bus contention—the node simply waits for an acknowledge gap, then begins transmission. A node begins arbitration after the bus remains idle for a predetermined amount of time, then transmits its priority level and arbitration number during the arbitration sequence. This sequence determines which node gains control of the bus for packet transmission.

***Fair Arbitration*** The physical layer arbitration method guarantees that only one node will have bus control at the end of the arbitration period. However, the node with the highest priority will always win?this is the node with the highest arbitration number and is physically closest to the root node. To provide equal bus access to all competing nodes, the asynchronous arbitration procedure adds a fairness protocol scheme. The fairness protocol is based upon the concept of a fairness interval.

Using fair arbitration, an active node can initiate an arbitration sequence to send an asynchronous packet exactly once during each fairness interval. When a subaction gap is detected, a node can begin arbitration only if its arbitration enable flag is set. The flag is set (1) at the arbitration reset gap (beginning of the fairness interval) and is reset (0) when the node successively accesses the bus through fair arbitration. This disables any additional arbitration requests by that node for the remainder of the fairness interval. A fairness interval ends when all nodes attempting fair arbitration have successfully accessed the bus. When the subaction by the final node is finished, an arbitration reset gap is generated, since all nodes now have their arbitration enable flags reset and cannot compete for bus control. The arbitration reset gap reenables arbitration by setting the arbitration enable flags in the nodes, thereby starting the next fairness interval.

***Urgent Arbitration*** The backplane physical layer enhances the fair arbitration algorithm by dividing access opportunities among nodes based on two priority arbitration classes: urgent and fair. Nodes using an urgent arbitration priority class may use up to 75% of the access opportunities in the fairness interval, with the remaining 25% equally shared among the nodes using the fair arbitration priority. The fair/urgent priority arbitration method is described below.

If the bus is idle for longer than an arbitration reset gap, then a fairness interval begins and all nodes set their arbitration enable flags. Those nodes implementing urgent priority set their urgent counters to a value of three. A fair arbitration node begins arbitration upon detecting a subaction gap if its arbitration enable flag is set; if the flag is reset, the node will wait for the arbitration reset gap before attempting arbitration. A fair arbitration node transmits a packet without the urgent label, then resets its arbitration enable flag. An urgent arbitration node sets its urgent counter to three whenever a fair arbitration packet is transmitted or received, including packets that are addressed to other nodes.

An urgent arbitration node decrements its urgent counter by one whenever a packet with the urgent label is transmitted or received by either itself or any other urgent arbitration node. This ensures that there will be no more than three urgent arbitration packets for each fair arbitration packet. The highest priority urgent node will always obtain bus control before other urgent nodes with lower priorities.

When fair arbitration and urgent arbitration nodes are both present on the serial bus, a fairness interval ends when the final fair arbitration node and up to three urgent arbitration nodes have successfully accessed the bus. Since all fair nodes now have their arbitration enable flags reset and all urgent nodes have their urgent counters decremented to zero, none of the nodes can access the bus. The bus remains idle until the next arbitration reset gap, which reenables arbitration on all nodes and begins a new fairness interval.

***Cycle Master Arbitration*** This arbitration class is used by the cycle master to arbitrate for the transmission of a cycle start packet. The cycle start packet is a primary packet sent by the cycle master that indicates the start of an isochronous cycle. This arbitration class is similar to the urgent arbitration class, except that the priority level is all ones, which is the highest priority. Arbitration begins when a subaction gap is detected.

***Isochronous Arbitration*** This arbitration class is used by nodes arbitrating to send isochronous packets. Isochronous services can be provided without disrupting the basic arbitration protocol by assigning the highest priority level to the cycle master. The cycle master must be the root, since the root node has the highest priority. Nodes transmitting isochronous data respond to cycle starts by arbitrating for the bus without waiting for a subaction gap. Arbitration begins when an acknowledge gap is detected regardless of the state of the arbitration enable flags or the urgent count flags. An isochronous packet is sent as soon as arbitration succeeds. This generates smaller minimum gaps between isochronous packets than for asynchronous packets. Because an acknowledge gap is shorter than a subaction gap or an arbitration reset gap, nodes arbitrating for isochronous transmission will obtain bus control before nodes waiting to send fair, urgent, or cycle master packets. The priority field is not used in this arbitration class.

***Immediate Arbitration*** This arbitration is used by nodes sending an acknowledge for a received packet. The acknowledge is sent when an acknowledge gap is detected. This arbitration is called immediate because an arbitration sequence is not necessary to obtain access to the bus.

## Link Layer

The link layer provides acknowledged data transfer services between a source node and a destination node. The basic protocol data unit is the packet and its corresponding acknowledge, both of which comprise a subaction. There are two types of subactions: asynchronous and isochronous.

### Asynchronous Subaction

An asynchronous subaction transmits a variable amount of data and several bytes of transaction layer information to a specific address. An acknowledgment is returned to the source node. All asynchronous subactions are separated by subaction gaps during which the bus is idle. Between the end of a transmitted packet and the corresponding acknowledgment by the destination node is a period of bus idle called an acknowledge gap. The acknowledge gap length may vary depending on the location on the bus of the receiving node with respect to the transmitting node. However, the maximum length of the acknowledge gap is significantly less than a subaction gap to ensure that other nodes on the bus will not begin arbitration before an acknowledge has been received.

The sequence for a nonconcatenated asynchronous subaction is: arbitration by the source node, packet transmission, acknowledge gap, and acknowledgment by the destination node. The sequence for a concatenated asynchronous subaction is: arbitration by the source node, packet transmission, acknowledge gap, and acknowledgment immediately followed by the second packet (without arbitration), the acknowledge gap, and acknowledgment.

An arbitration sequence occurs when a node is ready to transmit a packet of information to a destination node. The source node requests its physical layer to gain control of the bus. When bus control has been obtained for an asynchronous subaction, the source node sends the following packet information: a data prefix that may contain speed information; the source and destination address; a transaction code; a transaction label; a retry code; a data quadlet or data block; a header CRC character; a data block CRC character, if applicable; and a packet termination code.

### Isochronous Subaction

An isochronous subaction, also called a channel, transmits a variable amount of data at regular intervals with simplified addressing; no acknowledgment is required from the destination node. The source node transmits data every nominal cycle time of 125ms, which is also the guaranteed maximum latency. The maximum packet size is set by the bus management layer. All isochronous subactions are separated by an isochronous gap, which is a bus idle period before the start of arbitration for an isochronous subaction. The channel numbers 0 through 63 are available for assignment.

The sequence for a nonconcatenated isochronous subaction is: arbitration by the source node and packet transmission. The sequence for a concatenated isochronous subaction is: arbitration by the source node and packet transmission immediately followed by the next packet (without arbitration). Isochronous subactions have guaranteed access to the available bus bandwidth; asynchronous subactions use the remaining bandwidth.

An isochronous subaction includes a channel identifier rather than source and destination address. An isochronous channel is the connection between the source node and one or more destination nodes. One packet for each channel is transmitted during each isochronous cycle. A subaction acknowledgment is a packet sent from the destination node to the source node in response to most primary packets. An acknowledgment packet is always one byte in length and indicates the action taken by the destination node. Isochronous and asynchronous broadcast packets do not have acknowledgments.

## Link Layer Operation

The link layer is the center layer in a stack of three protocol layers that provides service to the transaction layer for asynchronous subactions. The link layer also provides isochronous subactions directly to the application.

***Sending an Asynchronous Packet*** The link layer sends an asynchronous packet when requested by the transaction layer. The link layer assembles the appropriate packet, then requests the physical layer to begin bus arbitration. When the node obtains control of the bus, the link layer sends the packet to the physical layer, which in turn, transmits the packet over the serial bus. The packet transmission ends when the link layer sends a data end indication to the physical layer. After the packet has been sent, an acknowledge packet of one byte is sent to the source node by the destination node. If the acknowledge is invalid or is not received within a specific time limit, then the link layer assumes that the acknowledge is missing. An acknowledge packet is considered invalid if the low- order four bits of the acknowledge byte is not the 1s complement of the high-order four bits. The acknowledge gap specifies the maximum time duration to receive an acknowledge packet. There will be no acknowledgment if the packet was a broadcast packet.

***Receiving an Asynchronous Packet*** The link layer receives an asynchronous packet from the physical layer upon receiving an indication of data availability. When the complete packet has been received, the physical layer sends an end-of-packet indication to the link layer. The link layer then examines the packet components (fields) to determine the validity of the packet. If the packet is invalid due to a CRC error or an incorrect address, the link layer ignores the packet. If the packet is valid, the link layer transmits the packet components to the transaction layer and returns an acknowledge packet to the source node via the physical layer.

***Sending an Isochronous Packet*** When a cycle start packet has been executed and the source node has isochronous data in its queue, the link layer requests the physical layer to begin arbitration for an isochronous arbitration class. When the source node obtains bus control, the packet is constructed in the appropriate format and sent to the physical layer for transmission on the bus.

If the link layer has isochronous packets to send to more than one channel, it will arbitrate for bus control and send the first packet, then concatenate subsequent packets until all channels have received their appropriate packets. This precludes the necessity to arbitrate for bus control for each packet. The link layer concatenates isochronous packets by sending a data prefix (indicating concatenation) to the physical layer at the end of a packet instead of a data end. The final isochronous packet is indicated by a data end symbol.

***Receiving an Isochronous Packet*** The link layer will receive isochronous packets if it has been configured as a *listener* by a source node. A listener is a node that receives an isochronous subaction for an isochronous channel. There may be more than one listener (destination node) for an isochronous channel. The link layer detects the end of a packet when the physical layer communi cates an ending symbol to the link layer. The link layer then examines the packet components to determine if the packet is valid and is addressed to a channel to which this node is listening. If the packet is invalid due to a CRC error or an inactive channel, the link layer ignores the packet. If the packet is valid and is addressed to a channel to which this node is listening, the link layer sends the packet components to the application. The link layer does not return an acknowledge packet to the source node.

## Transaction Layer

The transaction layer is one layer in a stack of three protocol layers that provides service to the link layer and the serial bus management layer. The link layer defines a request-response protocol to perform bus transactions for read, write, and lock. During a write transaction, data is transferred to an address in a different node. During a read transaction, data is retrieved from an address in a different node. A lock transaction transmits data parameters from a source node to a destination node. The destination node performs an indivisible function on the data and returns the updated data to the source node. The transaction layer does not support isochronous transactions.

### Read and Write Transactions

Read and write transactions provide services that are analogous to that for link layer subactions, except that the transaction layer does not support isochronous subactions. The transaction layer begins a read transaction by communicating a data request to the link layer in the source node. The request contains the destination address and the data length and initiates the appropriate request-response protocol between the source and destination nodes.

The response subaction begins when the destination node completes the requested read or write operation and initiates a data transaction response. The read response contains the read data, the data length, the source node identification, the transaction label, and response code. The write response contains the source identification, the transaction label, and the response code. A read or write transaction may be completed without a response subaction if an error occurred in the request subaction.

### Lock Transactions

For a lock transaction, the request contains the destination address, data, any arguments required for the lock operation, the data length of the arguments and data, and the lock function to be performed. The transaction layer initiates a lock transaction by sending a data request to the link layer which causes a data indication to be sent to the transaction layer of the destination node.

The response subaction begins when the destination node completes the requested lock operation. The response contains the previous data value, the previous data length, the source node identification, the transaction label of the corresponding request, and the response code. A lock transaction may be completed without a response subaction if there was an error in the request subaction.

## Error Handling

The transaction layer handles some error conditions so that invalid information is not passed on to the application. Error conditions detected by the transaction layer are:

***Response Data Error*** A data error acknowledge was sent to this node for a transaction response as a result of a CRC error, or the data length did not match the data length field.

***Response Format Error*** A type error was sent to this node because a field in the request packet header contained an unsupported or incorrect value, or an invalid transaction was attempted; for example, a write command to a read-only device.

***Acknowledge Missing*** An acknowledge packet was not sent to this node for a transaction response.

***Request Data Error*** A data error was detected in a transaction request received by this node.

***Unsolicited Response*** A transaction response was received for which there was no corresponding transaction request.

***Response Retry Limit*** A transaction response was not received within the specified number of retries.

## Split Transactions

A split transaction is a transaction that begins with an acknowledged request subaction, and is followed by an acknowledge response subaction, but not immediately. The destination node (responder) releases control of the bus after sending an acknowledge packet. At some time later, the node arbitrates for the bus to initiate a response subaction. Meanwhile, other subactions may occur between the request and the response subactions.

### *Unified Transactions*

A unified transaction is a request subaction with a corresponding acknowledge subaction, but is not followed by a response subaction. Only write operations may qualify as unified transactions.

### *Broadcast Transactions*

A broadcast transaction is a request subaction with no corresponding response subaction. Only write operations may be broadcast transactions.

### *Pending Transactions*

A pending transaction is a transaction that is not yet completed. For example, a unified transaction is completed when the request subaction has been acknowledged; a broadcast transaction is completed when the request subaction has been transmitted; all other transactions are complete when the response subaction has been acknowledged. A transaction is also complete when a response has not been received within a specified timeout period.

# Transaction Layer

The transaction layer is one layer in a stack of three protocol layers that provides service to the link layer and the serial bus management layer. The link layer defines a request-response protocol to perform bus transactions for read, write, and lock. During a write transaction, data is transferred to an address in a different node. During a read transaction, data is retrieved from an address in a different node. A lock transaction transmits data parameters from a source node to a destination node. The destination node performs an indivisible function on the data and returns the updated data to the source node. The transaction layer does not support isochronous transactions.

## Read and Write Transactions

Read and write transactions provide services that are analogous to that for link layer subactions, except that the transaction layer does not support isochronous subactions. The transaction layer begins a read transaction by communicating a data request to the link layer in the source node. The request contains the destination address and the data length and initiates the appropriate request-response protocol between the source and destination nodes.

The response subaction begins when the destination node completes the requested read or write operation and initiates a data transaction response. The read response contains the read data, the data length, the source node identification, the transaction label, and response code. The write response contains the source identification, the transaction label, and the response code. A read or write transaction may be completed without a response subaction if an error occurred in the request subaction.

## Lock Transactions

For a lock transaction, the request contains the destination address, data, any arguments required for the lock operation, the data length of the arguments and data, and the lock function to be performed. The transaction layer initiates a lock transaction by sending a data request to the link layer which causes a data indication to be sent to the transaction layer of the destination node.

The response subaction begins when the destination node completes the requested lock operation. The response contains the previous data value, the previous data length, the source node identification, the transaction label of the corresponding request, and the response code. A lock transaction may be completed without a response subaction if there was an error in the request subaction.

## Error Handling

The transaction layer handles some error conditions so that invalid information is not passed on to the application. Error conditions detected by the transaction layer are:

***Response Data Error*** A data error acknowledge was sent to this node for a transaction response as a result of a CRC error, or the data length did not match the data length field.

***Response Format Error*** A type error was sent to this node because a field in the request packet header contained an unsupported or incorrect value, or an invalid transaction was attempted; for example, a write command to a read-only device.

***Acknowledge Missing*** An acknowledge packet was not sent to this node for a transaction response.

***Request Data Error*** A data error was detected in a transaction request received by this node.

***Unsolicited Response*** A transaction response was received for which there was no corresponding transaction request.

***Response Retry Limit*** A transaction response was not received within the specified number of retries.

## Split Transactions

A split transaction is a transaction that begins with an acknowledged request subaction, and is followed by an acknowledge response subaction, but not immediately. The destination node (responder) releases control of the bus after sending an acknowledge packet. At some time later, the node arbitrates for the bus to initiate a response subaction. Meanwhile, other subactions may occur between the request and the response subactions.

### *Unified Transactions*

A unified transaction is a request subaction with a corresponding acknowledge subaction, but is not followed by a response subaction. Only write operations may qualify as unified transactions.

### *Broadcast Transactions*

A broadcast transaction is a request subaction with no corresponding response subaction. Only write operations may be broadcast transactions.

### *Pending Transactions*

A pending transaction is a transaction that is not yet completed. For example, a unified transaction is completed when the request subaction has been acknowledged; a broadcast transaction is completed when the request subaction has been transmitted; all other transactions are complete when the response subaction has been acknowledged. A transaction is also complete when a response has not been received within a specified timeout period.

## *Future Considerations for the 1394 Serial Bus*

The 1394 architecture will continue to meld current and emerging technologies into cohesive computing systems. The architecture represents the cornerstone for the fusion of many new modes of information media. The speed will undoubtedly increase well beyond one gigabits per second and improved data compression algorithms will further enhance the speed capabilities.

The flexible architecture of the serial bus will play a pivotal role in the computer industry which is changing more rapidly than ever before. The cost-effectiveness, speed capability, and ease of installation will tend to expand the applications of computers to ever increasing domains.

The small, low-cost physical connectors and versatile topology will find utilization in newly-emerging applications such as, personal digital assistants (PDAs), network computers, and information appliances. Many of these will have limited stand-alone capability and will derive most of their operational performance by interacting with server computers.

The confluence of technology and market requirements may well be the catalyst which allows the 1394 interface to integrate computer technology into societal requirements. The interface may potentially be an integral part of future wireless ATM networks where packet transmission occurs between a stationary host and a mobile host.

Fixed-disk drive manufacturers such as, IBM, Maxtor, and Western Digital are developing disk subsystems in anticipation of the imminent emergence of host systems. Interactive video capability and TV linking are also future possibilities in which the serial bus is ideally suited. The bus may conceivably attain prominence as an integral component in the internet infrastructure.

The future of the 1394 high performance serial bus is both promising and realizable.