A Bus on a Diet – The Serial Bus Alternative
An Introduction to the P1394 High Performance Serial Bus

Michael Teener, Plumbing Architect
Apple Computer, Inc. MS:60-AR, 20705 Valley Green Dr, Cupertino, CA95014
(Version 1.4— Thursday, April 22, 1993
This paper is largely based on one of the same title presented to CompCon '92 in February, 1992.)

ABSTRACT

This paper briefly discusses the justifications for the use a serial bus in computer systems. It then describes a leading proposal for such an interconnect: the proposed IEEE P1394 High Performance Serial Bus.

Note: A previous version of this paper was presented at IEEE CompCon '92, in February 1992. This version has been updated with the latest changes to the P1394 specification.

1. INTRODUCTION: WHY SERIAL?

Traditionally, computer systems are tied together with parallel backplane busses. Although this has served well in the past, designers are looking at narrower alternatives for three primary reasons:

1) Physical constraints. Systems are getting smaller and the space available is decreasing faster than connector technology can cope.
2) Cost. Semiconductor technology is still on a fairly fast evolutionary path, while connector and inter-connect technology is relatively mature. For a fixed level of performance, the cost of an interconnect drops much faster if its implementation is semiconductor intensive (narrow) instead of connector intensive (wide).
3) Reliability. The primary point of failure in an interconnect is usually its physical connection. The fewer the signals, the simpler (and more reliable) its connector.

1.1. Memory or Channel

There are two primary ways to view an interconnect: as some kind of extended memory space or as a more abstract channel interface. In a memory space interconnect, all resources are viewed as memory or registers that are accessed by "read", "write" and other processor-to-memory transactions. In channel interfaces, the transactions are at a higher level; typically some kind of command such as "read logical block" or "print page."

This paper confines itself to the memory model interconnects, so channel interfaces such as FibreChannel are not discussed.

1.2. Some Serial Bus Alternatives

There are a variety of serial designs that may serve as examples of serial busses. Three of these are listed below:

1) IEEE 1596 SCI. The Scalable Coherent Interface has three physical layers, and two of the three are 1250 Mbaud serial designs: one using coax and one using optical fiber. SCI is a pure 64-bit address memory-model interconnect that follows the IEEE 1212 Command and Status Register (CSR) Architecture. It has a sophisticated and powerful cache coherence mechanism that supports large multiprocessor systems with up to 65k nodes. This comes at the cost of significant complexity, but all that complexity is hidden in the implementation of the interface: something that scales well with developing semiconductor technology. Right now, this is the ultimate in serial bus performance.
2) Apple Desktop Bus(ADB). This is a proprietary minimalist serial interface that provides a simple read-write protocol to up to 16 devices, each with only eight eight-bit registers. Its data rate of 90 kbits/sec is very low, really only adequate for a CPU to communicate with keyboards, pointing devices or other desktop devices. ADB is extremely simple to implement, requiring less than 1000 gates.
3) IEEE P1394 High Performance Serial Bus. The P1394 serial bus lies somewhere between the previous examples. Like SCI, it follows the IEEE 1212 architecture and has a 64-bit address space. It does not, however, support cache coherency and its latency and transfer rate are not adequate for use in tightly-

coupled multiprocessing. On the other hand, as a low-cost interconnect for loosely-coupled systems it may find considerable use.

Since the P1394 serial bus lies in the fuzzy middle ground between an ultimate serial bus like SCI and the very modest ADB, it makes a good candidate for further examination.

2. THE P1394 HIGH PERFORMANCE SERIAL BUS[1]

The P1394 specification describes a high speed serial bus designed for low cost yet providing the data transfer speed and low latency needed for a peripheral bus or as a backup to a traditional parallel backplane bus. The highlights of the Serial Bus[2] include:

1) Automatic assignment of node addresses – no need for address switches.
2) Variable speed data transmission from over 24 Mbit/sec for TTL backplanes to almost 50 Mbit/sec for BTL backplanes to almost 400 Mbit/sec for the cable medium.
3) The cable medium allows up to 16 physical connections (cable hops) each of up to 4.5 meters, giving a total cable distance of 72 meters between any two devices. Bus management recognizes smaller configurations to optimize performance.
4) Bus transactions that include both block and single quadlet reads and writes, as well as an “isochronous” mode which provides a low-overhead guaranteed bandwidth service.
5) A Physical Medium Dependent layer supporting both cable media and backplane busses.
6) Both fair and priority bus access mechanisms that guarantee all nodes at least partial access, regardless of priority.
7) Consistent with the IEEE 1212 Control and Status Register Architecture Specification.

2.1. Topology

The physical topology of the Serial Bus is divided into two parts as shown in figure 1. The first part is called the “backplane environment” which is specific to the particular host parallel backplane. The other part is called the "cable environment" and is completely specified in the P1394 document. Nodes on a single bus may reside in different backplane environments, or directly in the cable environment. There is no requirement that the Serial Bus have any particular set of environments. All nodes may reside strictly in a single backplane, or they all may be directly attached to the cable, or any combination of backplanes or cable.

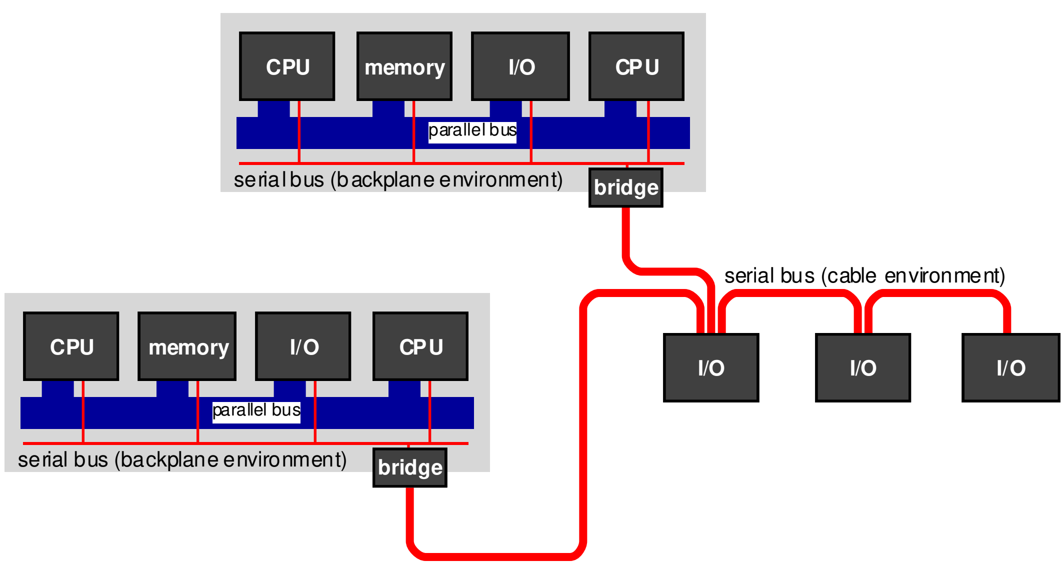

Figure 1. Serial Bus Physical Topology

[1]Note: much of this description of the P1394 Serial Bus is extracted from draft 6.0 of the proposed specification.

[2]In the rest of this paper, the P1394 High Performance Serial Bus will be referred to simply as the Serial Bus.

2.1.1. Cable Environment

The physical topology for the cable environment is a non-cyclic network with limited branches and extent. The medium consists of 2 shielded twisted pairs for signals and one pair for power and ground which connect together ports on a node. Each port consists of terminators, transceivers and simple logic. The cable and ports act as bus repeaters between the nodes to simulate a single logical bus.

Since each node must continuously repeat bus signals, the separate power and ground wires within the cable enable the physical layer of each node to remain operational even if node local power is off. The pair can even power an entire node if its requirements are modest. The Serial Bus cable provides 8 to 40 VDC at up to 1.5 A. The actual current available is system-dependent.

2.1.2. Backplane Environment

The Serial Bus can be extended within each physical device as a two-signal electrical bus. This can provide a simple and direct way for internal processing resources to communicate with appropriate peripherals.

The backplane interface to the Serial Bus will make use of the pins reserved for a Serial Bus by the various ANSI/IEEE bus standards. Drivers and receivers for these signals follow the conventions established by the appropriate parallel bus standard: e.g., Futurebus using BTL and Fastbus and SCI using ECL.

2.2. Node and Board Architectures

The Serial Bus architecture is defined in terms of entities called nodes. A node is an addressable entity, which can be independently reset and identified. More than one node may be co-located on a single physical module, and more than one functional unit may be co-located on a single node.

2.3. Addressing

The Serial Bus follows the IEEE 1212 standard for 64-bit fixed addressing, where the upper 16 bits of each address is the node_id. This allows a system of up to 64k nodes. The Serial Bus divides the node_id into two smaller fields: the higher order 10 bits specify a bus_id and lower order 6 bits specify a node_offset. Each of the fields reserves the value of all “1”s for special purposes, so this addressing scheme provides for 1023 busses each with 63 nodes to be accessible in a simple manner. This standardization is continued within the node, with 2^{48} bytes (256 terabytes) divided between CSRs (command and status registers – core IEEE 1212 resources, registers specific to the Serial Bus, a ROM ID area, and node-specific resources) and general memory space.

2.4. Protocol Architecture

The serial bus protocols are described as a set of three stacked layers as shown in figure 2:

1) The Transaction Layer defines a complete request-response protocol to perform the bus transactions required to support the IEEE 1212 architecture (the operations of read, write, and lock). Note that the Transaction Layer does not add any services for isochronous data, although it does provide a path for isochronous management data to get to the Serial Bus Management via reads from and writes to the isochronous control CSRs.
2) The Link Layer provides an acknowledged datagram[3] service to the Transaction Layer. It provides addressing, data checking, and data framing. One Link Layer transfer is called a “subaction”. The Link Layer also provides an isochronous data transfer service directly to the application.
3) The Physical Layer translates the logical symbols used by the Link Layer into electrical signals on the different Serial Bus media. It also guarantees that only one node at a time is sending data. The Physical Layer also defines the mechanical interfaces for the Serial Bus. There is a different Physical Layer for each environment: cable and backplane.

[3] one-way data transfer with confirmation of reception.

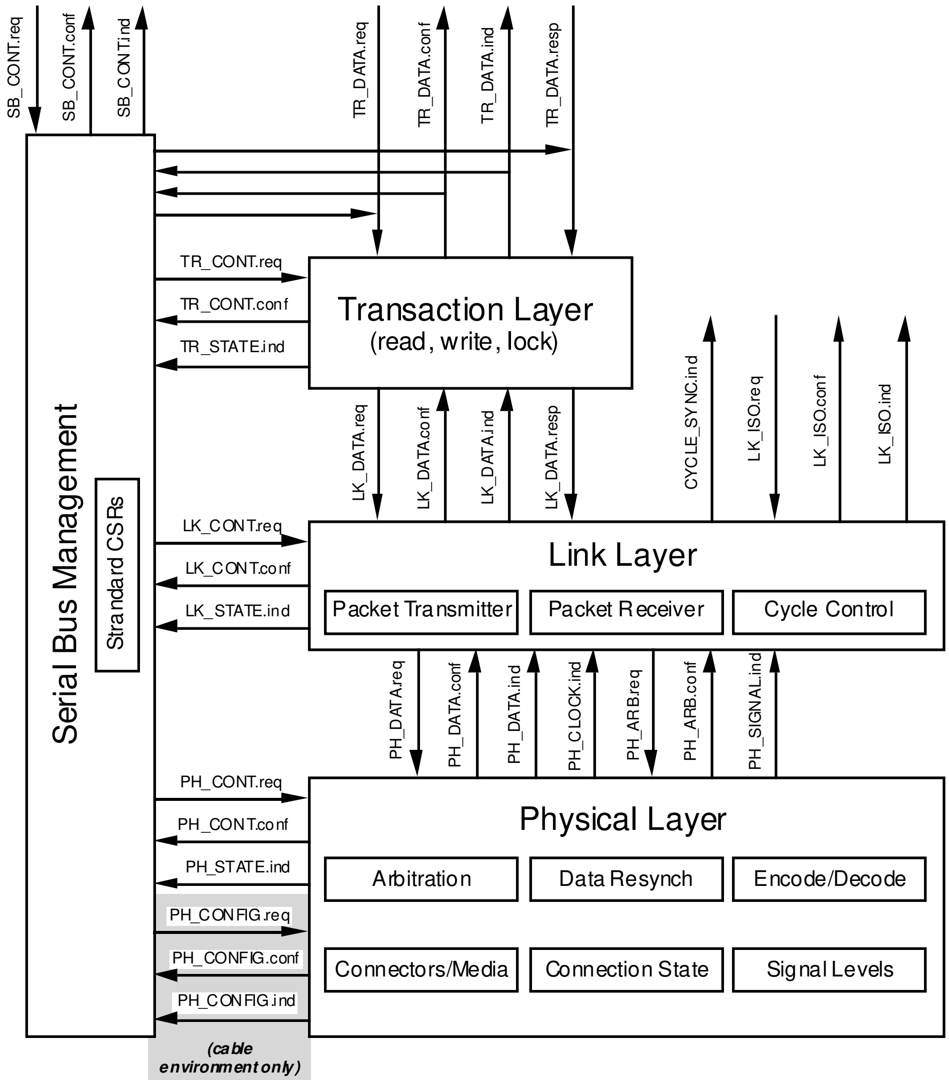

Figure 2. Serial bus protocol stack

The fourth entity in the Serial Bus protocol is Serial Bus Management which provides the basic control functions and standard CSRs needed by nodes on the bus.

3. TRANSACTION LAYER

Data is transferred between nodes (a requester and one or more responders) on the serial bus by three different types of transactions, where each transaction consists of one or two subactions: a request and possibly a response. The three transactions are:

1) Read — data is transferred from a responder back to a requester.
2) Write — data is transferred from a requester to one or more responders.
3) Lock — data is transferred from a requester to a responder, operated on by the responder, and then transferred back to the requestor.

Transactions consist of four actions:

1) Request — the action taken by a requestor to start the transaction.
2) Indication — the reception of a request by a responder.
3) Response — the action taken by the responder to finish the transaction.
4) Confirmation — the reception of the response by the requestor.

The Serial Bus architecture limits the maximum number of data bytes in a transaction to the largest power-of-two such that the whole Link Layer subaction takes less than 62 µsec (this is half the isochronous cycle time, and restricting asynchronous subaction to this value helps minimize buffer requirements). This means that packets are limited to 512 bytes at the cable base rate of 98.304 Mbit/sec. Implementations, however, can impose further restrictions. The only required transactions are quadlet write/read on quadlet aligned addresses (these are the transactions necessary to access the standard CSR resources).

3.1. Lock Subcommands

Since the Serial Bus supports split transactions, it cannot be easily locked while transaction sequences implement indivisible test-and-set operations. Therefore, special lock transactions are defined which communicate the intent from the requester to the responder, thus allowing the indivisible updates to be performed at the responder.

4. LINK LAYER

The Link Layer provides a half-duplex data packet delivery service. The process of delivering a single packet is called a “subaction”, and there are two types used in the Serial Bus Link Layer:

1) Asynchronous Subaction — a variable amount of data and several bytes of Transaction Layer information are transferred to an explicit address and an acknowledge is returned. N is
2) Isochronous Subaction or “Channel” — a variable amount of data is transferred on regular intervals with simplified addressing and no acknowledge.

The subaction has three possible parts:

1) Arbitration Sequence — a node that wishes to become a source requests the Physical Layer to gain control of the bus. This may be immediate if the node already controls the bus (as it would be if the subaction is the transaction response corresponding to the immediately preceding acknowledge).
2) Packet Transmission —the source node sends a speed code, format and transaction codes, addresses of the source and destination nodes, and data. Isochronous packets include a short channel identifier rather than source or destination addresses.
3) Acknowledgment — a uniquely addressed destination returns a code indicating the action taken by the packet receiver. If the acknowledgment is “pending”, then the transmission packet of the corresponding response may be concatenated to the end of the acknowledge code. Isochronous packets and asynchronous broadcast packets do not have acknowledgments.

All asynchronous subactions are normally separated by periods of idle bus called “subaction gaps,” but if a responding transaction layer is fast enough, then the request subaction and corresponding response subaction may be concatenated as shown in figure 3.

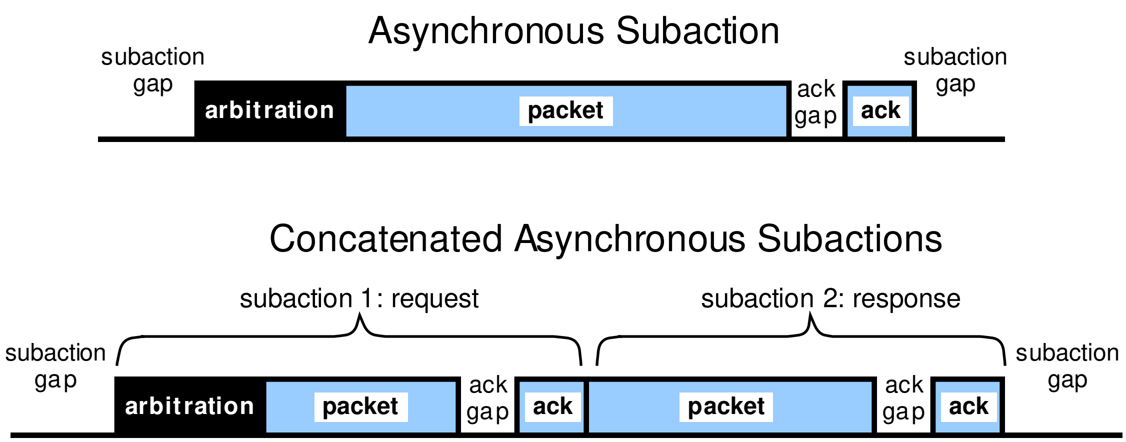

Figure 3. Example asynchronous subactions

The link layer operations also have the request, indication, response, and confirmation actions:

1) Request — the action taken by a link requestor to transmit a packet to a link responder.
2) Indication — the reception of a packet by a link responder.
3) Response — the transmission of an acknowledgement & optional packet by a link responder.

4) Confirmation — the reception of the acknowledgement & optional packet by the link requester.

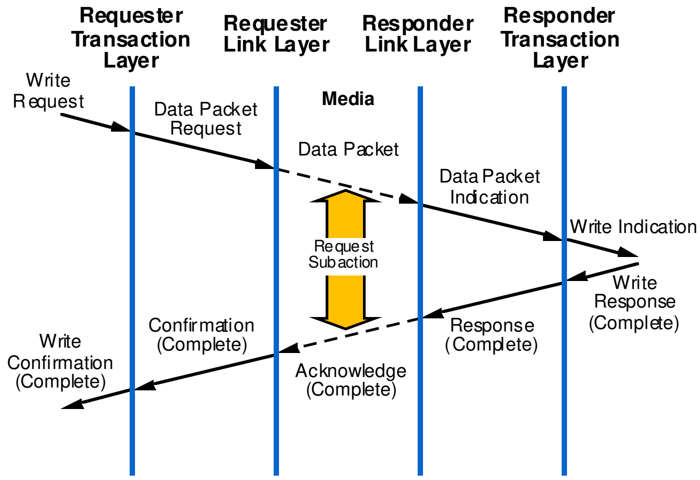

Figure 4. Unified transaction.

The Transaction Layer and Link Layer interact in a way that optimizes the use of the bus. In particular, transactions can be implemented in two different ways: unified or split. If the transaction and link layers of a responding node are fast enough, the entire transaction can be implemented with a single Link Layer subaction: the transmission of a packet and the corresponding acknowledgement.

Where the response takes longer to generate, then a split transaction is required with separate Link Layer subactions for the request and the response. Note that other Link Layer subactions occur on the bus between the request and response subactions of a single transaction.

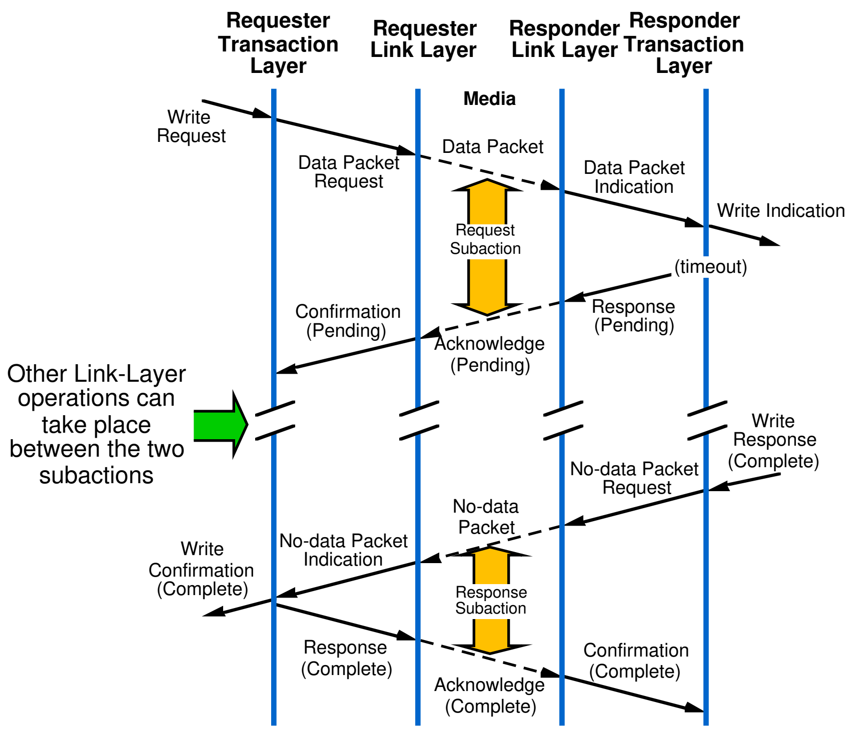

Figure 5. Split Transaction Example

When the Transaction Layer is busy and cannot accept a packet, the Link Layer sends a “busy” acknowledge back to the requesting node (this indicates that the destination Transaction Layer will not be returning a response). As an option, the Link Layer at the requesting node can retry the packet for a limited period. If bus retry traffic is high, then slower-acting nodes may be starved. To prevent this from happening there is a two-phase busy/retry protocol which allows all retries to be processed in a timely fashion.

4.1. Arbitration

To enhance the usefulness of Serial Bus, each node on the bus can use one of three different access methods. These access protocols are:

1) Fair Arbitration. Fair arbitration is used by nodes which can equally share the bus bandwidth. This is the default method.
2) Urgent Arbitration. Urgent arbitration can be used by nodes which desire a majority of the bus bandwidth or have severe latency requirements. For example, a high-bandwidth real-time data collection node may use priority arbitration when critical data buffers are more than half full. This method is only available in the backplane environment.
3) Isochronous Access. Data that is regularly generated and consumed, such as digital sound, can be transferred using an isochronous access method that guarantees bandwidth on the bus with very low overhead.

The first two arbitration methods use the request-response arbitration service provided by the Physical Layer while isochronous access requires a higher level protocol.

4.2. Isochronous Access

The normal bus request used by asynchronous subactions is quite adequate for nodes that do not require a guaranteed amount of bandwidth or a relatively precise timing reference (less than a microsecond, for instance). Some data, such as that related to digital sound or instrumentation, are more conveniently handled using an isochronous access method.

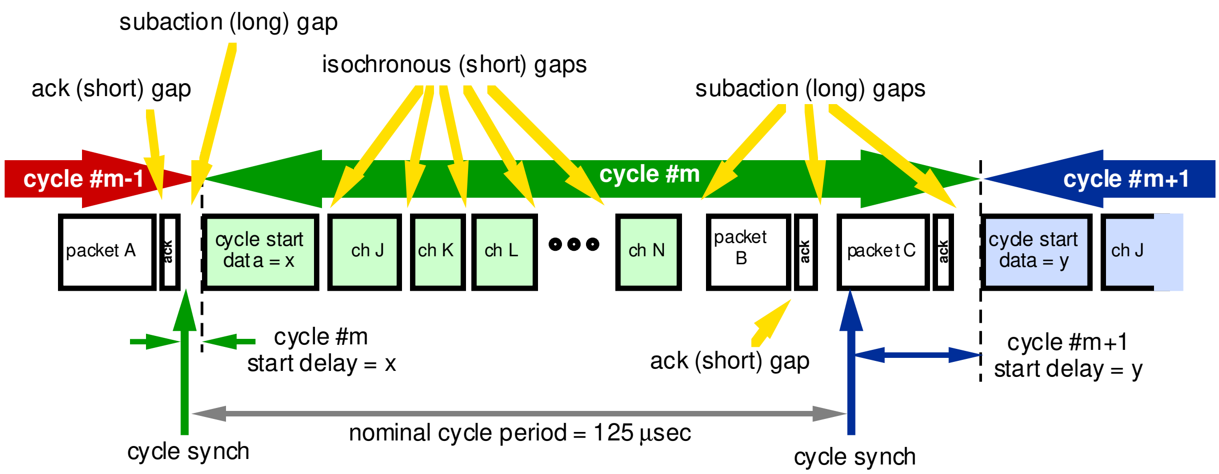

Figure 6. Cycle Structure

These isochronous services can be provided without upsetting the basic access protocol by establishing the convention that the highest priority access is reserved for a "cycle" master that is responsible for maintaining a common clock source. The cycle master will try to transmit a special timing request called a "cycle start" at very specific intervals set by a "cycle synch" source (nominally 8 KHz ± 100 ppm, or 125 µsec ± 12.5 nsec). If another transfer is still taking place when the cycle synch occurs, then the cycle-start will be delayed resulting in significant jitter in the start time of the transfer. Since this jitter is frequently unacceptable for a node, the amount of time that the cycle start request was delayed is encoded within the request.

Nodes sending isochronous data respond to cycle start requests by immediately arbitrating for the bus without waiting for a subaction gap and sending an isochronous packet as soon as arbitration succeeds. This leads to a smaller minimum gap between isochronous packets than is needed for asynchronous arbitration to start. (The timing for gaps between isochronous packets is the same as for asynchronous acknowledges.) Only when all the nodes sending isochronous data have won arbitration and finished sending their data will the bus stay idle long enough for a subaction gap to appear. It is the subaction gap which will allow normal asynchronous arbitration to resume. Figure 6 illustrates the basic isochronous access system.

The isochronous packets are labeled with 8-bit "channel" numbers which have been previously assigned by standard write transactions sent by a channel manager application. Since nodes sending isochronous packets still arbitrate for the bus and the natural priority of a node is not related to channel number, the order of packet transmission is also not related with channel number.

When a node has been assigned a channel, it can send a variable amount of data up to a maximum specified by the channel manager. It is up to a channel manager application to guarantee that the time consumed by all nodes sending isochronous data not exceed the 125 µsec in a cycle.

5. PHYSICAL LAYER

The Physical Layer has three primary functions: data transmission and reception, arbitration, and the actual electrical and mechanical interface. The cable and backplane environments each have a separate physical layer but share two fundamental concepts: “data-strobe” encoding for packet data, and a simple method for ensuring fair access to the bus.

5.1. Packet Data Transmission and Reception

During packet transmission, there is only a single node transmitting on the bus, so the entire media can operate in a half duplex mode using two signals: Sbus_Data and Sbus_Strb. NRZ data is transmitted on Sbus_Data and is accompanied by the Sbus_Strb signal which changes state whenever two consecutive NRZ data bits are the same, ensuring that a transition occurs on either Sbus_Data or Sbus_Strb for each data bit. A clock which transitions each bit period can be derived from the exclusive-or of Sbus_Data with Sbus_Strb as shown below.

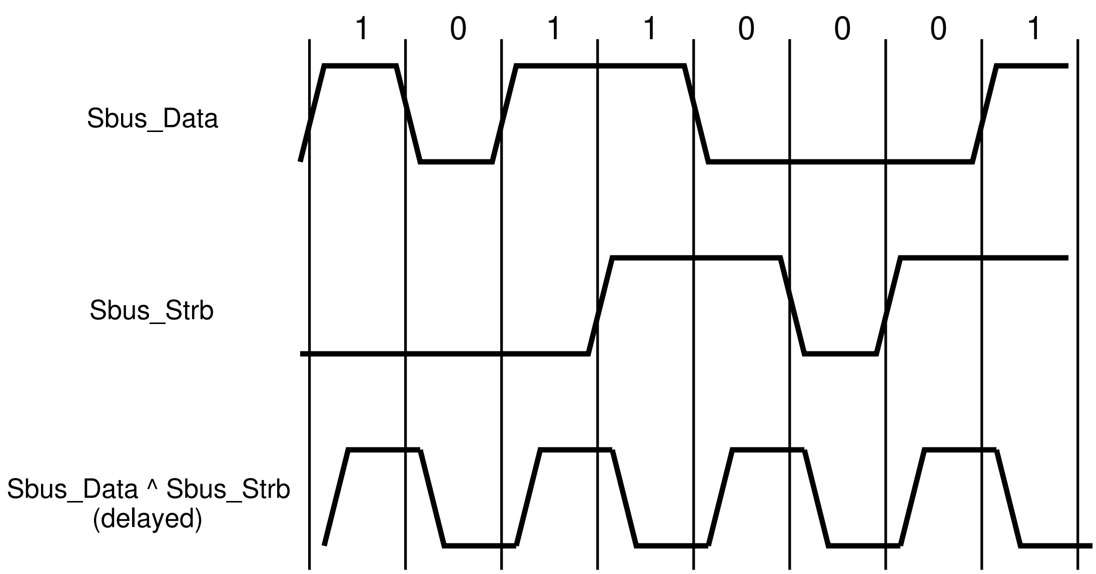

Figure 7. Data-strobe encoding

5.2. Fair Arbitration

The normal cable and backplane arbitration methods guarantee that only one node will be transmitting at the end of the arbitration period. As described above, these methods only provide a strict priority access; the node with the highest natural priority (highest arbitration number on a backplane, closest to the root on a cable) will always win. The normal asynchronous access method for the Serial Bus adds a simple scheme that splits the access opportunities evenly among competing nodes.

The fairness protocol is based on the concept of a fairness interval. A fairness interval consists of one or more periods of bus activity separated by short idle periods called subaction gaps and is followed by a longer idle period known as an arb reset gap. At the end of each subaction gap, bus arbitration is used to determine the next bus owner. This concept is shown in figure 8.

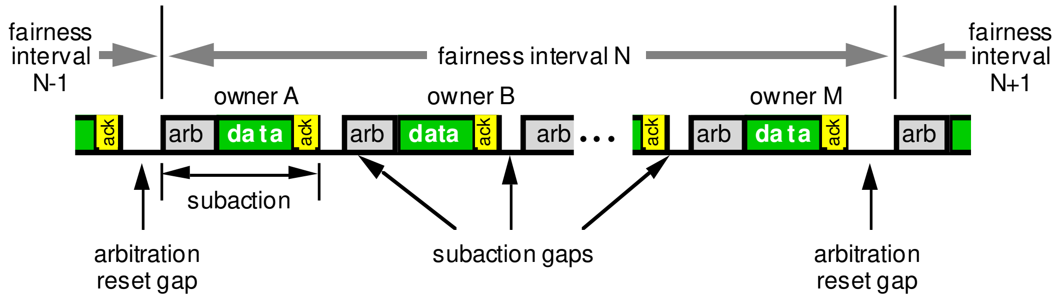

Figure 8. Fairness interval.

The implementation of the fair arbitration protocol is defined in terms of these fairness intervals, as is discussed in the following clauses.

When using fair arbitration, an active node becomes a bus owner exactly once in each fairness interval. The arb_enable signal is set to one by an arb_reset_gap and is cleared when the node becomes a bus owner. This disables further arbitration requests for the remainder of the fairness interval. A fairness interval ends when arbitration by the final fair node is successful; this generates a arb_reset_gap since all nodes now have their arb_enable signals reset and cannot drive the bus. The arb_reset_gap re-enables arbitration on all cards and starts the next fairness interval. This process is illustrated in figure 9.

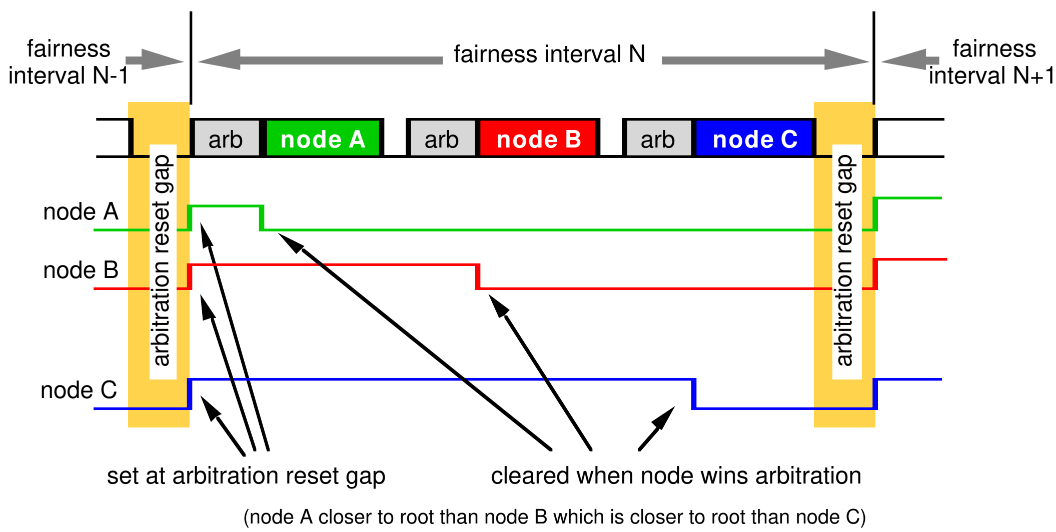

Figure 9. Fair arbitration.

5.3. Cable Physical Layer

The cable environment is a network of nodes connected by point-to-point links called *physical connections*. The physical connection consists of a port on each of the nodes and the cable between them. A node can have multiple ports, which allows a branching multihop interconnect as shown in figure 1. The primary restriction is that nodes must be connected together as an acyclic graph (no loops).

5.3.1. Cable Arbitration

The cable access method takes advantage of the point-to-point (non-bussed) nature of the cable environment by having each node handshake with its immediate neighbors to determine ownership of the media. There are four phases to this scheme, three to initialize the cable configuration, and one for normal arbitration.

5.3.1.1. Bus initialize

Whenever a node joins the bus, a signal forces all nodes into a special state that clears all topology information and starts the next phase. After bus initialize, the only information known to a node is whether it is a branch (more than one directly connected neighbor), a leaf (only a single neighbor) or isolated (unconnected). This is illustrated in figure 10.

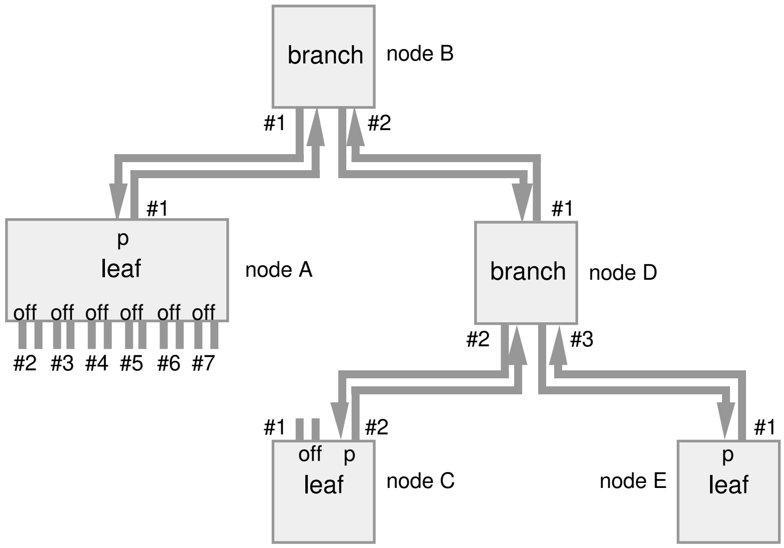

Figure 10. Example state after bus initialization.

Note that each port of the node is individually numbered. There is no particular order to the numbering, it is just a way to give each port a unique label.

5.3.1.2. Tree identify

After a bus initialize, the tree ID process translates the general network topology into a tree, where one node is designated a root and all of the physical connections have a direction associated with them pointing towards the root node. The direction is set by labeling each connected port as a “parent” (connected to a node closer to the root) or “child” port (connected to a node further from the root). Any unconnected ports are labeled “off” and do not participate in further arbitration processes. Any loop in the topology is detected by a timeout in the tree-ID process.

5.3.1.3. Self identify

The next step is to give each node an opportunity to select a unique physical_ID and identify itself to any management entity attached to the bus.

The self-ID process uses a deterministic selection process where the root node passes control of the media to the node attached to its lowest numbered connected port and waits for that node to signal that it and all of its children have identified themselves. The root then passes control to its next highest port and waits for that node to finish. When the nodes attached to all the root’s ports are finished, the root itself does a self identify. The child nodes use the same process in a recursive manner.

Figure 11 illustrates the state of the bus after the self-ID process is complete.

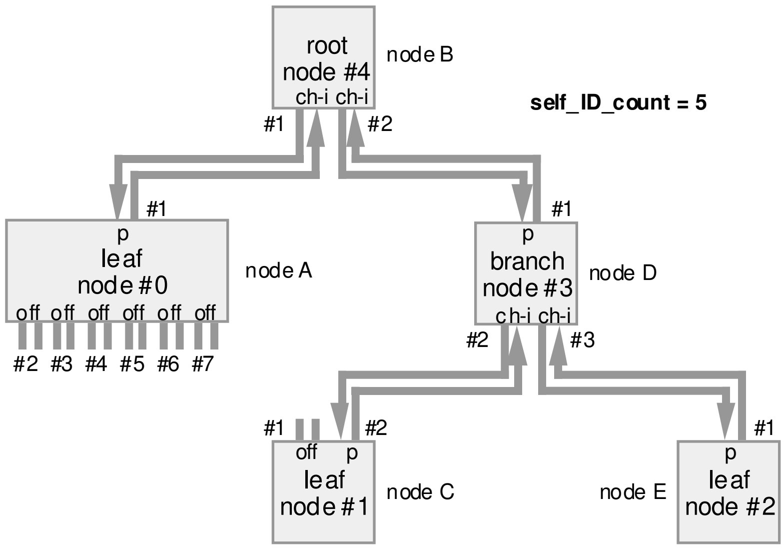

Figure 11. State after self-identify.

5.3.1.4. Normal arbitration

Once the self identify process is complete, nodes can use the normal arbitration method to send packets. This process is illustrated by the following example:

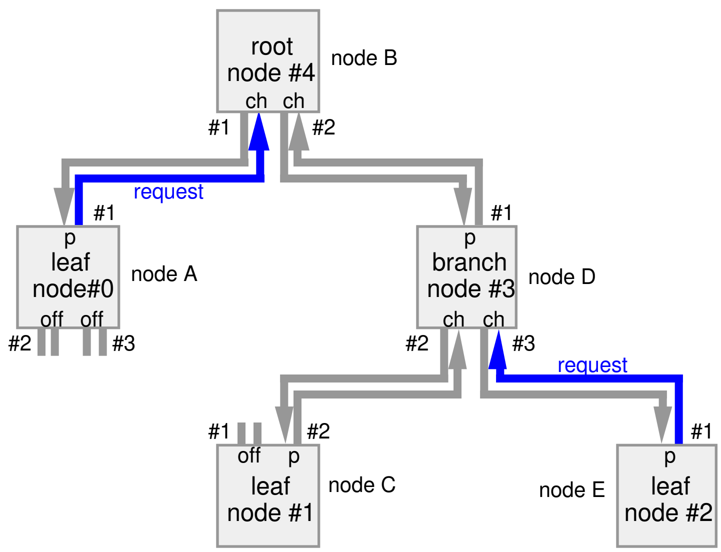

Figure 12. Arbitration request

1) Node A and node E begin arbitrating at the same time by sending a request to their parents.

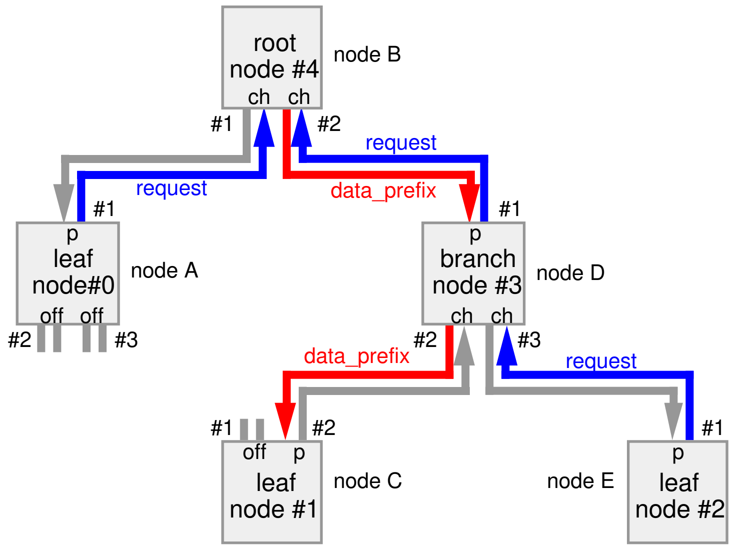

Figure 13. Arbitration request (continued).

2) The parent of node E (node D) forwards the request to its parent (node B) and denies access to its other children (node C) by sending a data_prefix, while simultaneously the parent of node A (node B) denies access to its other children (node D). Since node B is root, it doesn't have to forward the request any further.

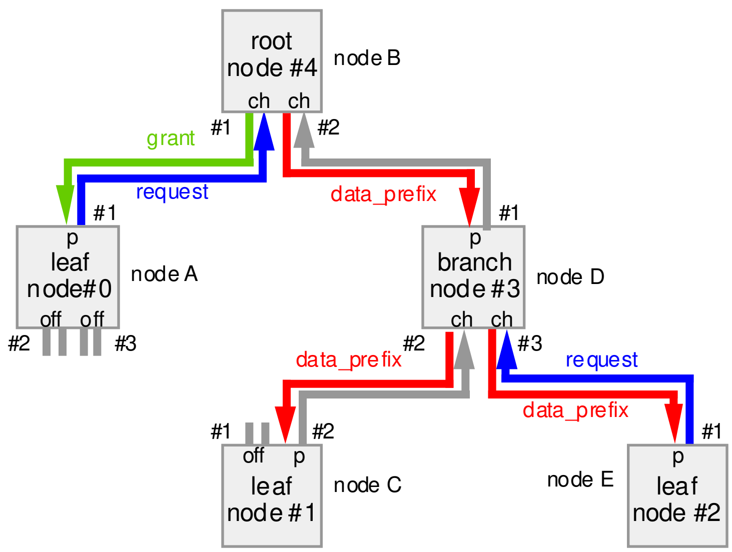

Figure 14. Arbitration grant.

3) Instead the root grants access to the first request (node A), while the other parent (node D) acknowledges the deny by withdrawing its request and passing on the deny.

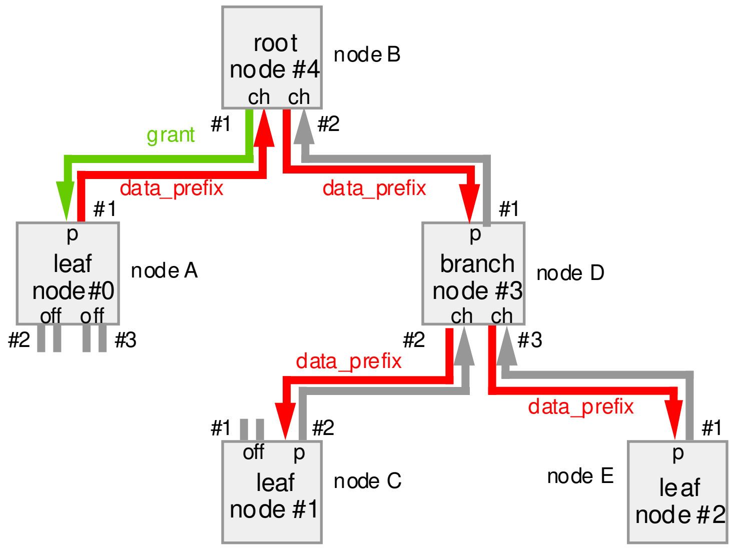

Figure 15. Data prefix.

4) This causes node E to withdraw its request. Simultaneously, node A receives the grant and, since it was the original requesting node[4], sends a data_prefix signal to warn all nodes that data is about to be sent.

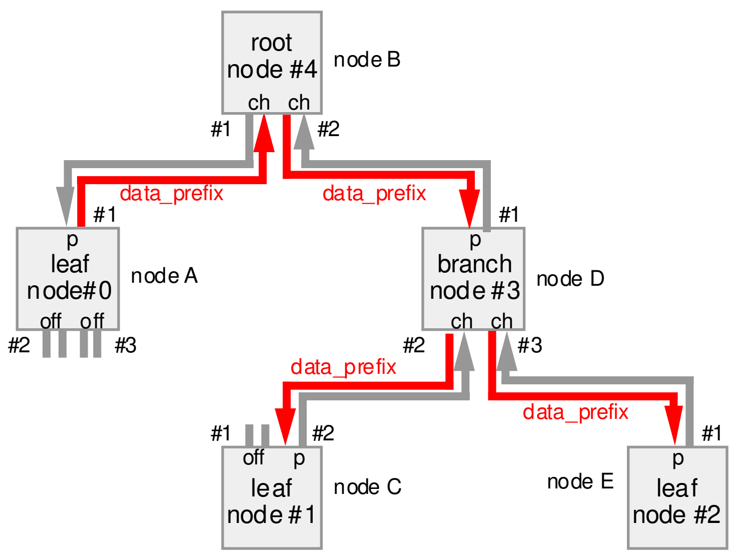

Figure 16. Start of data transmission.

5) The parent of node A (the root in this case) sees the data prefix and withdraws the grant. At this point, the physical connections between all the nodes are now in the same state and pointed away from the node that won the arbitration. This allows the unused second signal to be turned around and used as a strobe to time the transmission of data.

5.3.2. Cable Media Interface

The cable media interface is the implementation of a physical connection. There are three major parts of the cable media interface:

1) The electrical interface to the cable, which consists of:

---

[4]If node A had children, they would have received a data_prefix when A started arbitration. If, however, one of node A's children had requested the bus first, node A would have done an internal deny and passed on the request to the root, and later on the grant when it received it.

a) Two low voltage, low current bidirectional differential signals to carry clocked packet data or arbitration signals.
b) power pair that provides the current needed by the physical layer to repeat signals. Nodes can either source or sink current, and there can be multiple power sources on a bus.

2) The cable connectors, which are small and rugged and provide 6 electrical contacts plus a shield as shown in figure 17. The hole for the socket must be roughly 6 mm by 11 mm.
3) The cable media itself, which has two well shielded signal pairs with relatively high-impedance (meaning that little power is needed to drive an adequate signal), and one relatively low impedance pair for power.

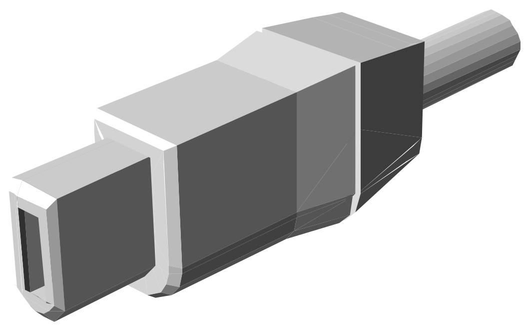

Figure 17. Cable plug.

5.4. Backplane Physical Layer and Arbitration

The backplane environment is a multidrop, very tightly controlled transmission line. There are two signals: Sbus_Data and Sbus_Strb which are shared among all nodes on a broadcast bus.

The backplane environment does not have the initialization requirements of the cable environment since the topology is fixed as a broadcast bus (no repeaters) and physical addresses may be set by the host backplane using its built-in slot identifier mechanism. If the backplane does not provide such a slot identifier mechanism, then the serial bus shall support some method for determining a unique arbitration number (i.e., hardwiring the address).

The backplane access method is bit-wise arbitration on a dominant-mode media. It depends on two assumptions: (1) if any node asserts the bus, then all nodes perceive the bus as asserted after a certain length of time ("wired-or" and "open-collector" are other names for this); and (2) each node has a guaranteed unique arbitration number. The procedure followed by competing nodes is:

```
START
wait for SBUS_DATA and SBUS_STRB to remain unasserted for a fixed period (a gap);
assert SBUS_STRB;
for each bit in the arbitration sequence {
    if the arbitration bit is a one {
        assert SBUS_DATA;
        wait eight base rate bit times (an arbitration symbol period); }
    else {
        release SBUS_DATA;
        wait five base rate bit times (the sample time) and sample SBUS_DATA;
        if SBUS_DATA is asserted {
            lost arbitration,
            wait three base rate bit times (the hold time);
            release SBUS_STRB and go back to START }
    }
won arbitration, release SBUS_DATA and continue to assert SBUS_STRB;
begin packet transmission
```

ACKNOWLEDGEMENTS

The P1394 working group has been blessed with unusually good working relationships. The author is particularly grateful to David James and David Gustavson for sticking with the development since its beginnings. Mr. James and Dan O'Connor have been the primary contributors to the link and transaction protocols. The physical layer is largely the work of Roger Van Brunt with the able assistance of Florin Oprescu. The cable environment PMD development has been led by Andy Carter.

REFERENCES

[1] *High Performance Serial Bus,* IEEE P1394 Draft 6.0, April 1993.

(All P1394 standards drafts are available from the IEEE Computer Society at 201-371-0101 or via anonymous FTP from "ftp.apple.com.")