New Technology in the IEEE P1394 Serial Bus – Making it Fast, Cheap and Easy to Use

Michael Teener, Plumbing Architect
Apple Computer, Inc. MS:60-AR, 20705 Valley Green Dr., Cupertino, CA 95014
(with numerous contributions by the members of the IEEE P1394 Working Group)

This paper was originally presented to the Hot Interconnects Symposium '93 at Stanford University, August 6, 1993. Minor corrections were made to this draft on August 3, 1993, so it does not exactly match the version in the proceedings.

Abstract

The IEEE P1394 High Performance Serial Bus has four primary design goals:

a) very low cost (less than $15 to the user),

b) useful in both an externally cabled and internal backplane environment,

c) very high performance (better than existing and planned SCSI designs) and

d) true plug-and-play operation.

To meet all three of these goals required significant invention by the members of the P1394 working group. Among these inventions were:

1) an encoding scheme that provides a full bit cell of jitter tolerance
2) a self-configuring hierarchical arbitration system for the cable environment
3) a fair access method that provides deterministic access times even when the bus is fully saturated
4) a real time "isochronous" data transport mechanism that operates transparently on top of a standard network-type asynchronous packet transfer

These four example inventions will be discussed in the following text, along with a brief introduction to the Serial Bus itself. Note that there are many more innovations in the specification which cannot be included in this paper for lack of space. Such P1394–unique technologies as a very low power electrical interface that consumes about 2 mW in each connection in the cable environment, an extremely rugged connector and cable system, and the simple and flexible arbitration scheme used in the backplane environment are not included.

NOTE: much of this paper is derived from draft 6.3 of the P1394 specification.

1. The P1394 High Performance Serial Bus

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 include:

a) A physical layer supporting both cable media and backplane busses.

b) Automatic assignment of node addresses – no need for address switches.

c) 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.

d) The cable medium allows up to 16 cable hops between any two device, each hop up to 4.5 meters long, giving a total cable distance of 72 meters. Bus management recognizes smaller configurations to optimize performance.

e) Bus transactions that include both variable byte-length and single quadlet reads and writes, as well as an "isochronous" mode which provides a low-overhead guaranteed bandwidth service.

f) A fair bus access mechanism that guarantees all nodes equal access. The backplane environment adds a priority mechanism, but one that ensures that nodes using the fair protocol are still guaranteed at least partial access.

g) Consistent with the IEEE 1212 Control and Status Register Architecture Specification.

1.1 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” and there are several different versions associated with different backplane bus standards. The other part is the “cable environment” and is completely specified in the P1394 document. Nodes on a single Serial Bus may reside in either environment, in any combination, without preference or restriction.

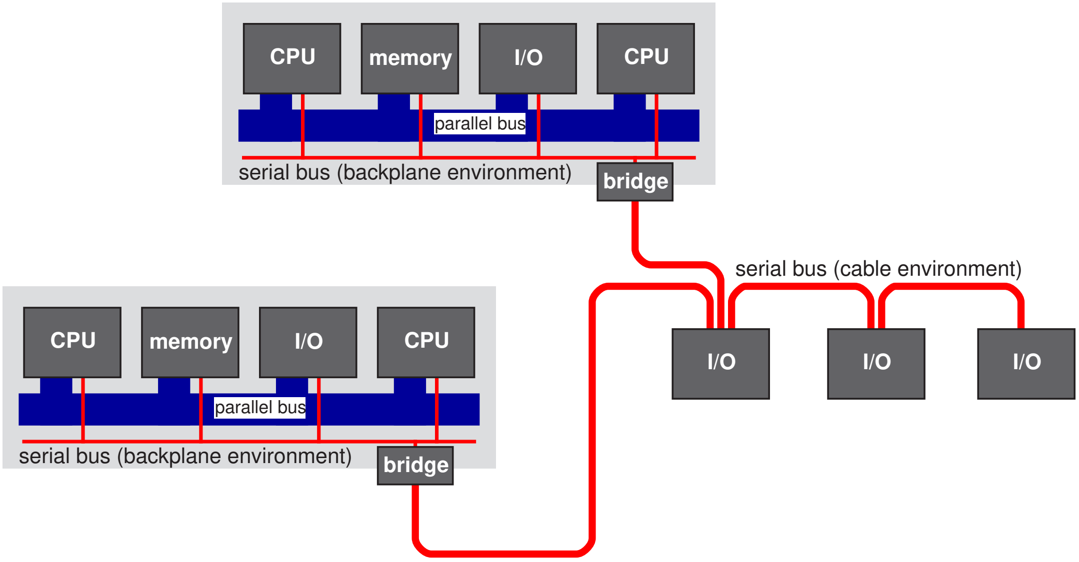

Figure 1 – Serial Bus Physical Topology

1.1.1 Cable Environment

The physical topology for the cable environment is a non-cyclic network (no loops) with the only limitation being the number of cable hops between any two devices (called “nodes”) and the length of a cable hop. The cables consists of 2 shielded twisted pairs for signals and one pair for power and ground. These cables connect to “ports” on the nodes. 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 can carry 8 to 40 VDC at up to 1.5 A. The actual current available is system-dependent.

1.1.2 Backplane Environment

The Serial Bus can be extended into each physical device as a two-signal electrical bus, which provides a simple and direct way for internal processing resources to communicate with peripherals.

The Serial Bus backplane interface uses the two pins reserved for a Serial Bus by the various ANSI/-IEEE bus standards. Drivers and receivers for Serial Bus signals follow the conventions established by the appropriate parallel bus standard: e.g., FutureBus using BTL and Fastbus and SCI using ECL.

1.2 Protocol Architecture

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

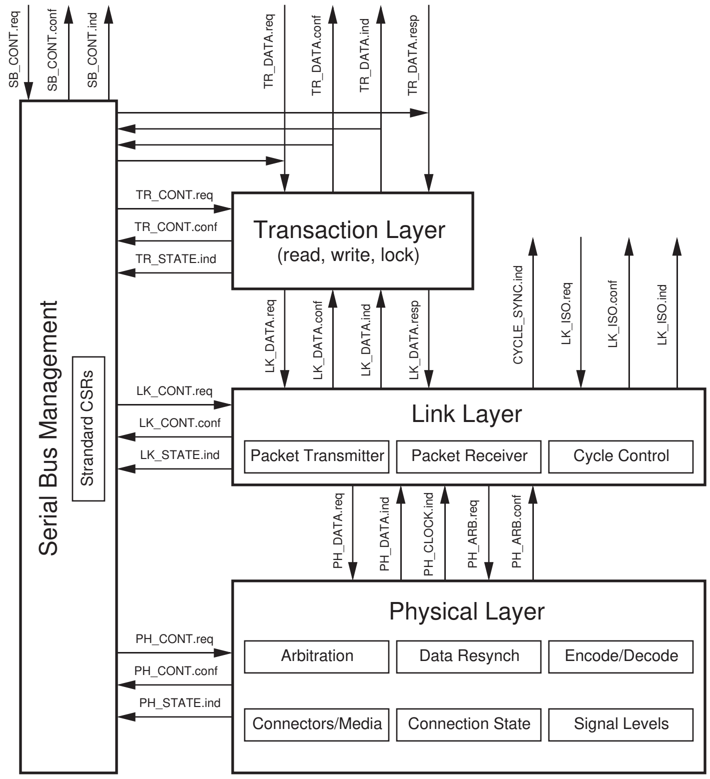

Figure 2 – Serial bus protocol stack

a) 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.

b) The Link Layer provides an acknowledged datagram[1] 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.

c) 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 by providing arbitration services. The Physical Layer also defines the mechanical interfaces for the Serial Bus. There is a different Physical Layer for each environment: cable and backplane.

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.

1.3 Transaction Layer

Data is transferred between nodes on the serial bus by three different transaction types:

a) Read — data is transferred from a responder back to a requester.

b) Write — data is transferred from a requester to one or more responders.

c) Lock — data is transferred from a requester to a responder, operated on by the responder, and then transferred back to the requester.

---

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

The Serial Bus follows the IEEE 1212 standard for 64-bit fixed addressing, where the upper 16 bits of each address represent the node_ID. This allows address space for up to 64k nodes. The Serial Bus further divides the node_ID into two smaller fields: the higher order 10 bits specify a bus_ID and lower order 6 bits specify an offset_ID. Each of the fields reserves the value of all "1"s for special purposes, so this addressing scheme provides for 1023 buses each with 63 independently addressable nodes.

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. Example lock transaction are "compare-and-swap" and "fetch-and-add".

1.4 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:

a) 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.

b) 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) Data 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. Isochronous packets and asynchronous broadcast packets do not have acknowledgments.

All asynchronous subactions are normally separated by periods of idle bus called "subaction gaps" as shown in figure 3. Note that a gap opens up on the bus between the packet transmission and ack reception. This "ack gap" is of varying lengths depending on where the receiver is located on the bus with respect to the senders of the link request and ack.

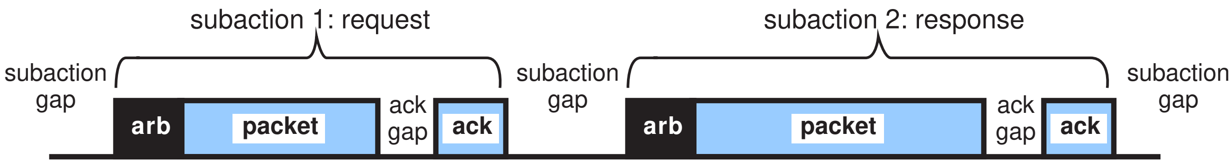

Figure 3 – Example asynchronous subactions

Similarly, isochronous subactions are separated by periods of idle bus called "isoch gaps" as shown in figure 4.

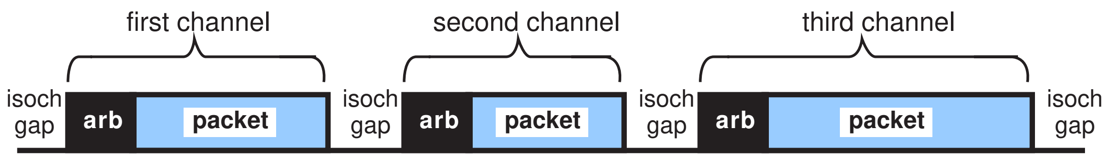

Figure 4 – Example isochronous subactions

1.5 Physical Layer

The Physical Layer (the PHY) 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.

1.5.1 Cable Physical Layer (C-PHY)

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 previously in figure 1. The primary restriction is that nodes must be connected together as an acyclic graph (no loops).

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

a) The electrical interface to the cable, which consists of:
   1) Two low voltage, low current bidirectional differential signals to carry clocked packet data or arbitration signals.
   2) A 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.

b) The cable connectors, which are small and rugged and provide 6 electrical contacts plus a shield.

c) 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 the power.

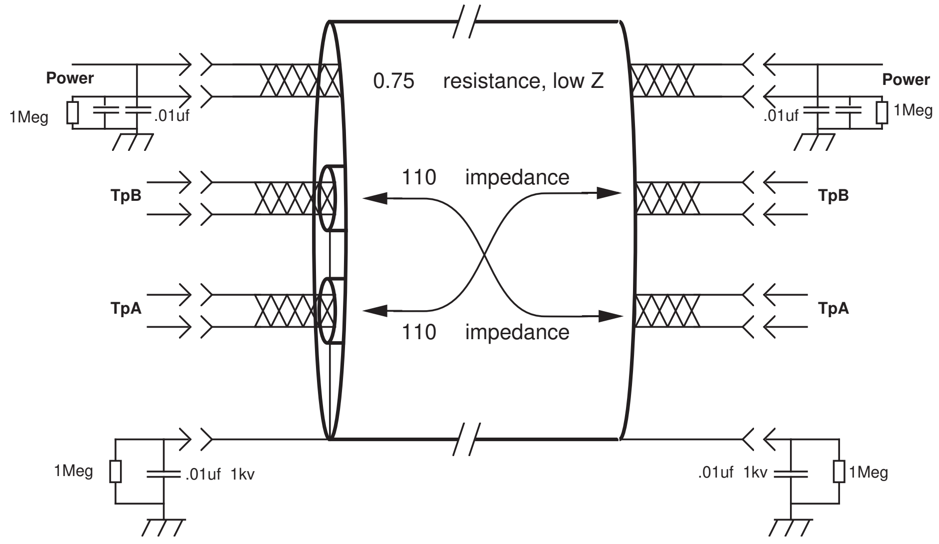

Figure 5 – Cable physical connection

The arbitration method used by the cable environment uses an automatically configuring hierarchical request-response algorithm that is described in clause 3. later in this paper.

1.5.2 Backplane Physical Layer

Unlike the cable environment, the backplane environment is a multidrop, very tightly controlled transmission line. There are two signals which are shared among all nodes on a broadcast bus, so the physical extent and speed are much more restricted. The actual transceivers used in this environment are chosen to match those used in the host backplane standard: ECL for IEEE 1596 SCI, BTL for IEEE 896 FutureBus and TTL for VME and NuBus.

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. The arbitration itself is a simple bitwise transmit-and-sample scheme.

1.6 Bus Management

The Bus Management protocol provides a multitude of management facilities:

a) Configuration management. The cable physical layer provides a great deal of configuration information during initialization. This data is used to:

   1) ensure that there is adequate power provided for all the nodes on the bus,
   2) optimization of the arbitration timing

a) Cycle master arbitration. Before any cycle transactions can start, all candidate cycle master nodes must compete to become the cycle master. The candidate with the most accurate clock should become master. In the cable environment, the cycle master must be the root, so the assignment of a cycle master may also require a bus reset to force a configuration change.

b) Isochronous channel assignment.

c) Error control. The basic philosophy for error handling is to note that there is an error, but not necessarily retry any transfer. Higher layers are responsible for guaranteeing delivery, not the interconnect. The bus does, however, provide the facilities for a non-hogging retry.

2. Invention #1: Data-Strobe Encoding

The Serial Bus operates at extremely high speeds, yet the cost and power objectives are aggressively low. Meeting these goals required tolerating a fairly high level of jitter on the received data. Although early drafts of the P1394 specification used self-clocked 4B5B and 8B10B codes, and even simple clocked NRZ, they all imposed extremely difficult timing requirements and/or added significant complexity. Fortunately, engineers at Inmos, Ltd. in the UK had come up with an innovative code that they were using for the DS-Link processor communication system (now being standardized as IEEE P1355).

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.

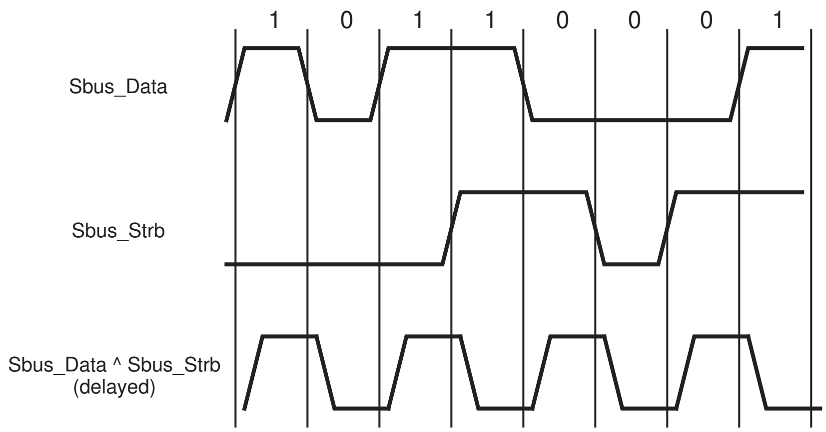

Figure 6 – Data-strobe encoding

The primary rationale for use of this transmission code is to improve the transmission characteristics of information to be transferred across the serial bus. In particular, the code ensures that transitions occurring on Sbus_Strb and Sbus_Data are approximately one bit period apart. This results in an increase in skew tolerance which could not be obtained with a clocked NRZ format. This has been demonstrated with the prototype ICs built as part of an internal Apple project: the same transceivers that were running at 100 Mbit/sec using standard clocked NRZ (using both edges of a clock) has been shown to operate correctly at 200 Mbit/sec using the data-strobe encoding.

Example circuits for encoding and decoding data is shown in figure 18.

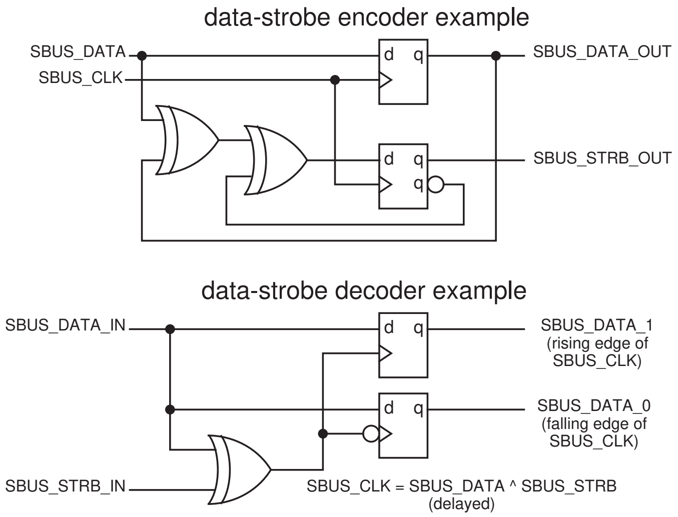

Figure 7 – Data-strobe encoder and decoder example

NOTE: The "data-strobe" encoding is based on U.K. patent number 9011700.3, claim 16 (DSLink bit-level encoding) held by INMOS Limited, 1000 Aztec West, Almondsbury, Bristol BS12 4SQ, UK.

3. Invention #2: Self-Configuring Hierarchical Arbitration

The cable version of the Serial Bus is intended to operate in a "non-supervised" customer environment (no manuals or extensive explanations needed). Fundamentally, this means that the user should not have to configure the system in any way other that to plug in the cables. At the same time, the working group wanted to support both branching and daisy-chain topologies with a large number of devices (up to 63) with minimal access time latency.

The arbitration and configuration method selected by the working group was invented by Florin Oprescu of Apple Computer, and it provides all the features outlined above, plus it allows the system to be optimized to the actual number of nodes and how they are connected together. In addition, it provides all the information necessary to build a topology map of the interconnect (useful for diagnostics) and to minimize power use.

This cable arbitration technique 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.

3.1 Cable Configuration

Cable configuration is done in three phases: bus initialization, tree identification, and self identification. During this process a treelike topology is built, each node is assigned a physical node number and also sends node-specific information that is used by the management layer.[2]

[2] This process is illustrated in more detail in annex C of the P1394 specification.

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 below:

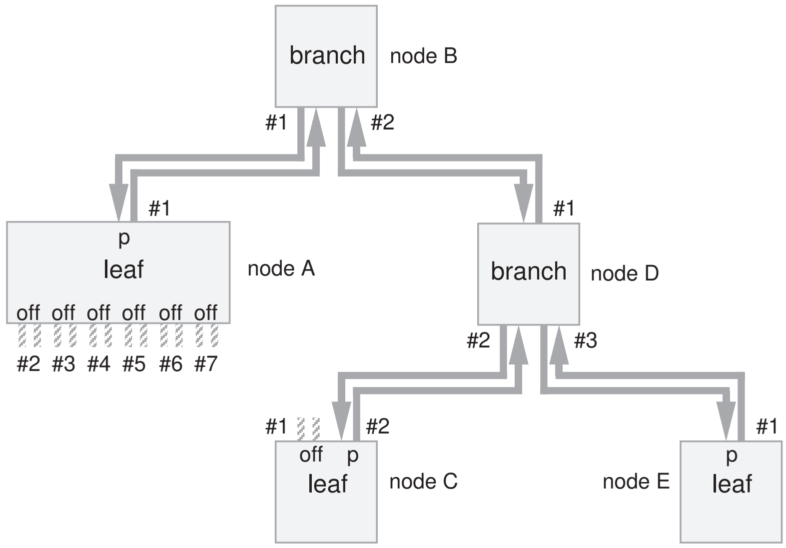

Figure 8 – Example cable state after bus initialize

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

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 configuration timeout in the tree-ID process. When the tree-identify process is complete, the example configuration will have each port labeled child (pointing to a child node) or parent (pointing to its parent node) as shown in figure 12.

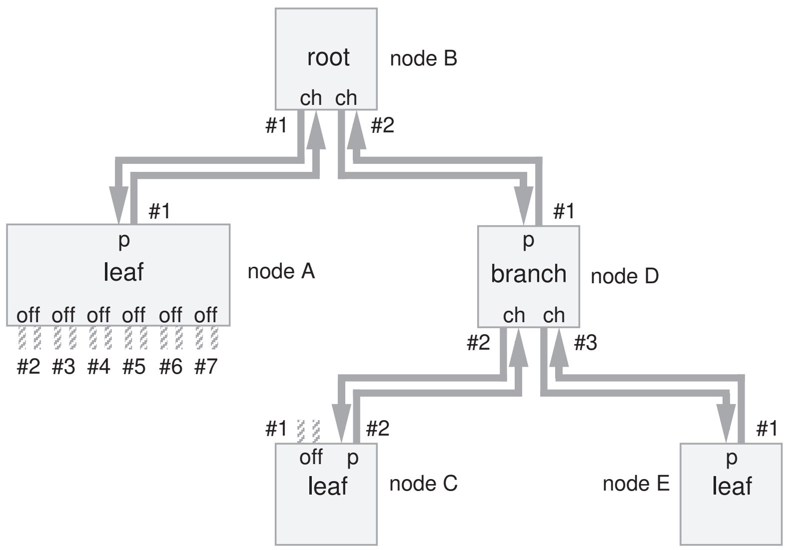

Figure 9 – Tree identify complete

Note that the selection of the root node is not topology dependent. It is completely acceptable that the root node also be a leaf. The only requirement is that the cycle master (the isochronous cycle master described in clause 3.6.3) must also be the root, since the root has the highest natural priority.

The node that waits an adequately long period[3] after the bus reset to start participating in the tree identify process becomes the root. If, for example, node A in the previous figures waited a long enough time to start the tree ID process, node B would end up sending the parent_notify to node A after step b) above. This would cause node A to become the root.

A particular node can be forced to wait a longer time by setting a bit in one of its management registers.

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. This is needed to allow low-level power management and the building of a system topology map. Details of power management and topology mapping are described in Clause 8, "Serial Bus Management," of the draft specification.

Sending the self-ID information is done by transmitting one to four very short packets at the base rate onto the cable that includes the physical ID and some management information. The physical ID is simply the count of the number of times a node passes through the state of receiving self-ID information before having its own opportunity to do so, i.e., the first node sending self-ID packet(s) chooses zero as its physical ID, the second chooses one, and so on. Note that a node is not required to decode the self-ID packet, it merely has to count the number of self identify sequences since the bus reset (this is the number of times the Self-ID Receive state described in clause 4.4.2.3 is entered).

The management information included in the self-ID packet includes codes for the power needed to turn on the attached Link Layer, the state of the various ports (unconnected, connected to child, connected to parent), and data rate limitations.

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 3-21 illustrates the state of the bus after the self-ID process is complete. Note that each child port is now labeled "ch-i", meaning that the node attached to that port is now identified.

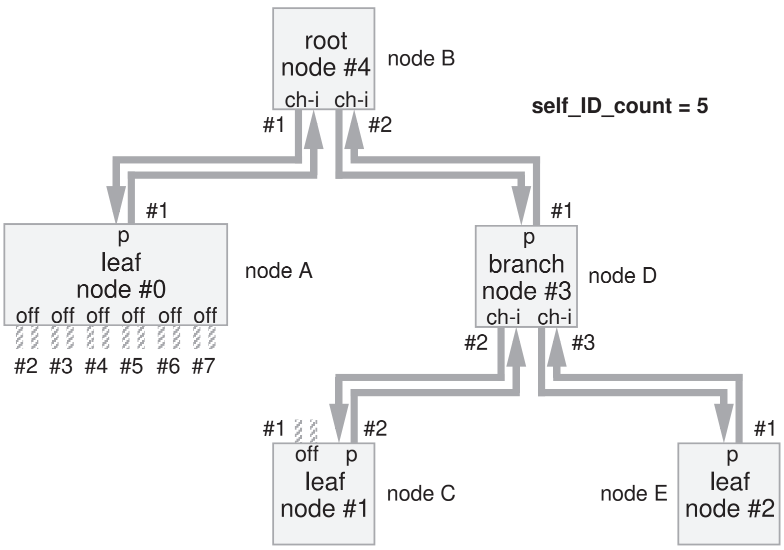

Figure 10 – Example cable state after self identify phase

[3] longer than a worst case unrestricted tree-identify process.

3.2 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:

a) After the bus is idle for a specific length of time (called the “subaction gap”), node A and node E

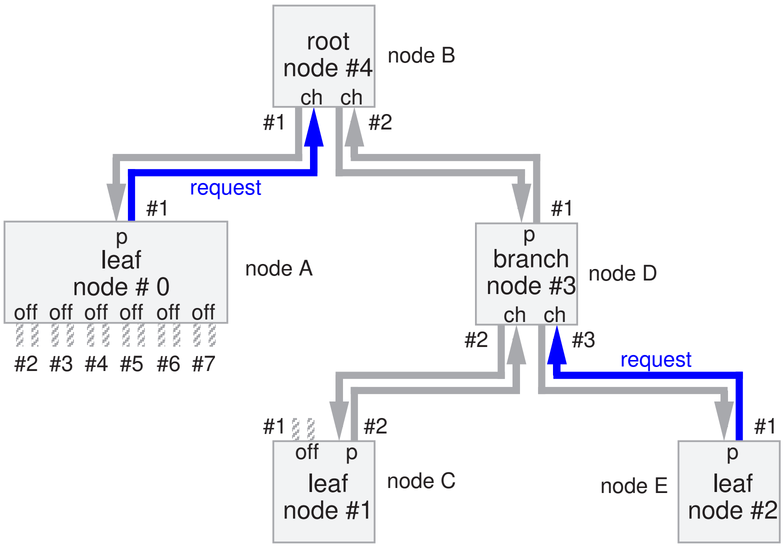

Figure 11 – Arbitration request

begin arbitrating at the same time by sending a request to their parents.

b) 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.

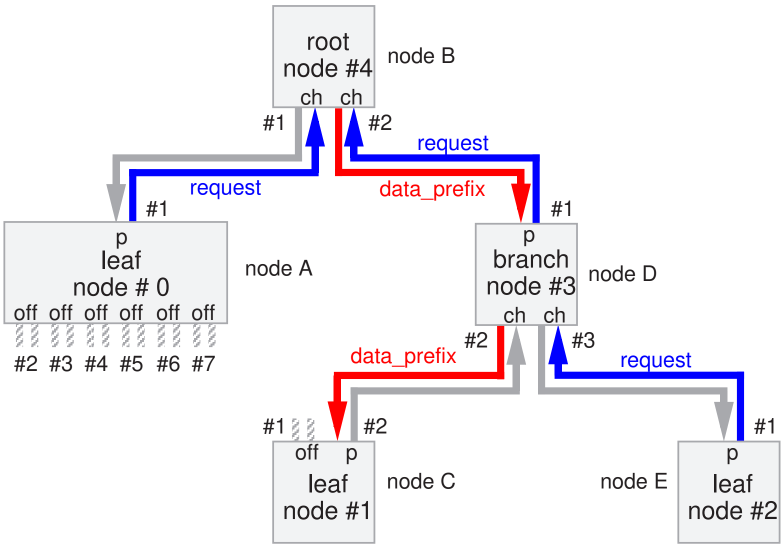

Figure 12 – Arbitration request (continued)

c) 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.

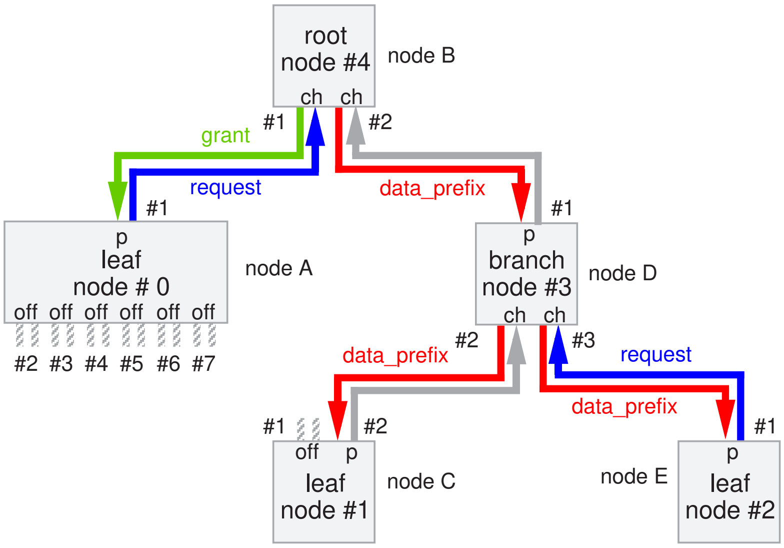

Figure 13 – Arbitration grant

d) 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.

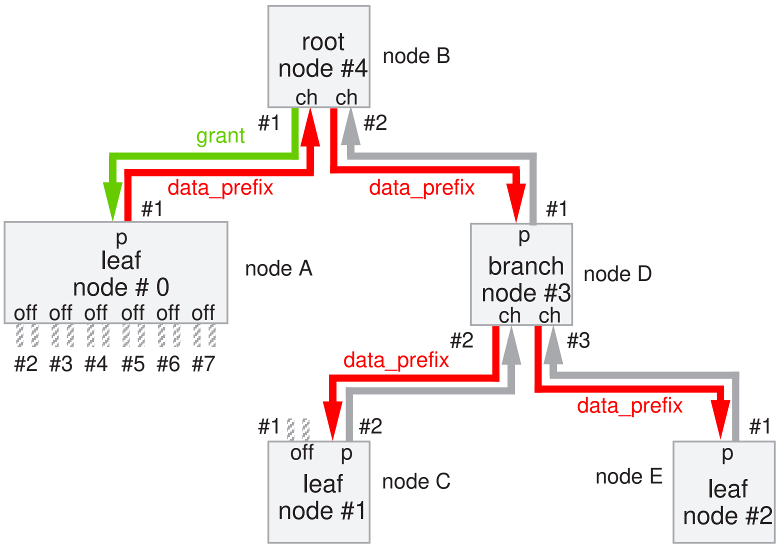

Figure 14 – Data prefix.

[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.

e) 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.

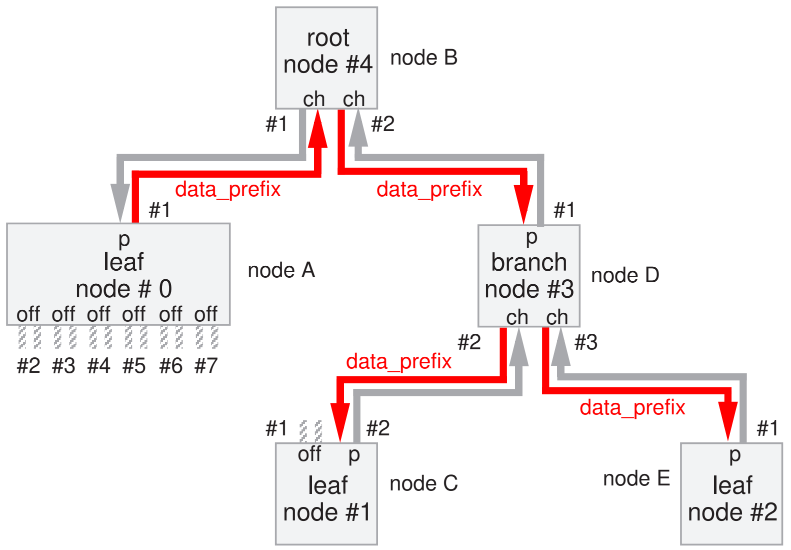

Figure 15 – Start of data transmission

The timing of this process is dependent on two values: the number of cable hops between a node and the root and the length of the subaction gap. The first value automatically scales with the complexity for the network, but the second is controlled by the management process. The length of the subaction gap is set to a value larger than the worst case round trip time for arbitration signals between any two nodes (this is the time between a node sending a link-layer packet and receiving the corresponding acknowledge). After a bus configuration process, this is set to a number corresponding to 16 hops, but bus management can readjust the time to a lower value after the bus topology has been determined from the self-ID information.

4. Invention #3: Fair and Urgent Arbitration

The normal arbitration methods used by the Serial Bus guarantee that only one node will be transmitting at the end of an arbitration period. 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 fairness scheme invented by David James in late 1985 (then of Hewlett Packard) that splits the access opportunities evenly among competing nodes, giving extra accesses to nodes that have an urgent need for the bus.

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 arbitration 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.

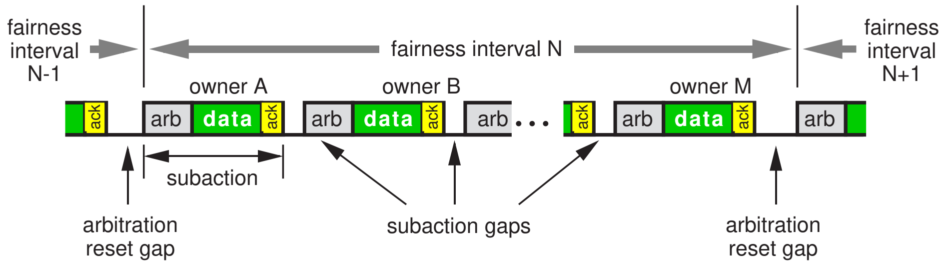

Figure 16 – Fairness interval

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.

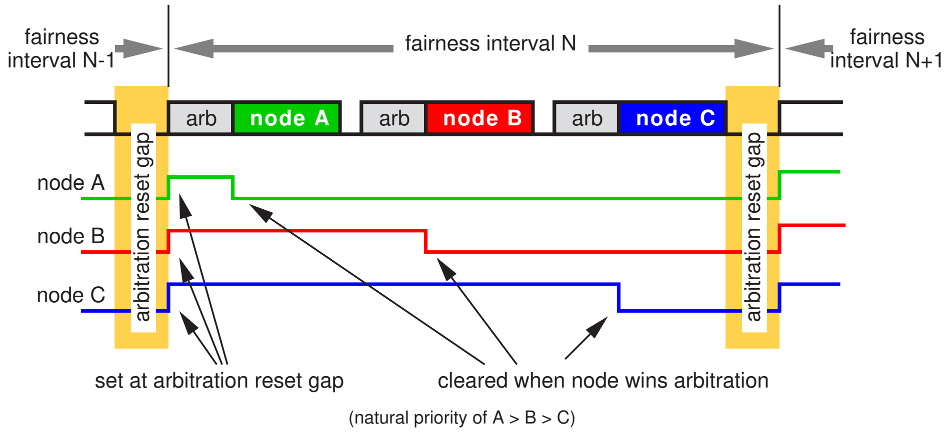

Figure 17 – Fair arbitration

The backplane environment enhances this fair algorithm to split the access opportunities between the nodes based on two priority classes, with nodes using an “urgent” priority using three fourths of the access opportunities, with the remaining equally shared among nodes using the “fair” priority. All nodes are required to implement the fair priority class, while the urgent priority class is optional. Packets are labeled as “urgent” if that priority class was used.

The fair/urgent allocation uses the same fairness interval used above, but adds an “urgent_count” to the arbitration_enable flag. The fair/urgent method works as follows:

a) As above, if the bus is idle for longer than the arbitration reset gap, a new fairness interval starts and all nodes set their “arbitration_enable” flags, while nodes implementing the urgent priority also load the value 3 into their “urgent_count” counters.

b) A node that wishes to send a packet using the fair priority class may arbitrate for any access opportunity as long as its arbitration_enable flag is set.

c) Whenever the node wins an arbitration contest using the fair priority class, it sends a packet without the “urgent” label and the arbitration_enable flag is cleared. This means that if a node is only using the fair priority it can only send a single packet each fairness interval; i.e., it must wait until all other nodes send their packets, at which point the bus will be idle long enough for an arbitration reset gap which begins a new fairness interval.

d) A node that wishes to send a packet using the urgent priority class may arbitrate for any access opportunity when its urgent_count is nonzero.

e) Whenever the node wins an arbitration contest using the urgent priority class, it sends a packet with the “urgent” label and it decrements its urgent_count.

f) A node using the urgent priority adds 3 to its urgent_count when any unlabeled packets is received (except for the first), including all those addressed to other nodes. This allows each node using the urgent priority to win arbitration three times for each single fair packet sent in a fairness interval. (The counter is not incremented on the first unlabeled packet since it has already been initialized to 3 by the start of the fairness interval.)

g) A node with a nonzero urgent_count also decrements its urgent_count whenever a packet is detected with the urgent label. This restricts the allocation of access opportunities to no more than 3/4 of the total.

NOTE: If two nodes use urgent arbitration, the node with the highest natural priority will always win.

For example, figure 3-28 illustrates a situation where there are three nodes arbitrating for the bus with node_offsets such that A has the highest, B is in the middle and C has the lowest, with nodes A and C using fair priority and B using urgent:

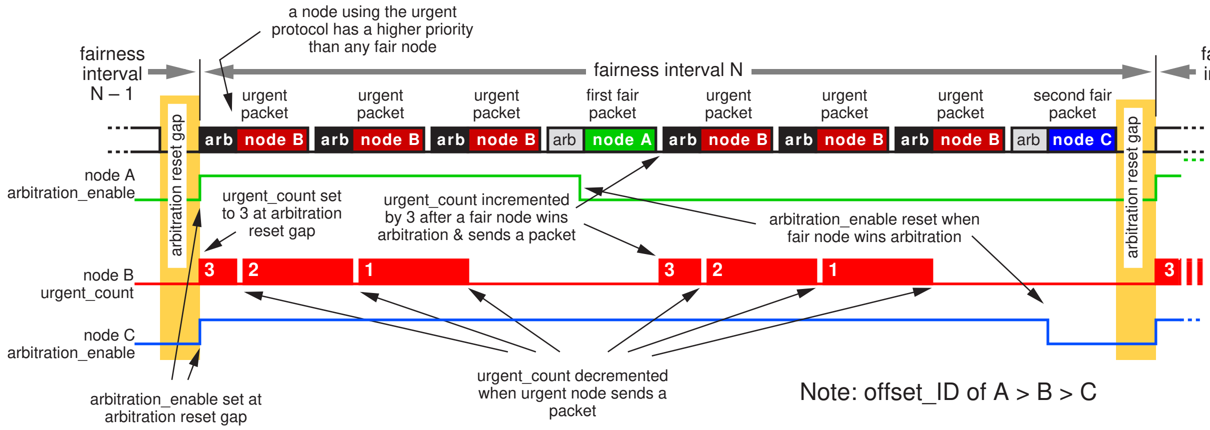

Figure 18 – Fair/urgent arbitration example

In the backplane environment, the natural priority is the concatenation of the 4-bit urgent priority level with the node_offset. This results in the following:

a) A node using the urgent protocol will always win an arbitration contest over all nodes using the fair protocol.

b) The node using the highest priority level will win the arbitration contest.

c) If more than one node uses the highest priority level, then the one with the highest node_offset will win.

5. Invention #4: Isochronous Access

The normal asynchronous access provided by the Serial Bus is adequate for nodes that do not require a low guaranteed latency or a precise timing reference (less than a microsecond, for instance). However, data such as that related to digital sound, video or instrumentation are more conveniently handled with such a service. For these applications, the Serial Bus provides “isochronous” access; meaning that the data can be delivered within a specified latency with a guaranteed bandwidth. This combination allows a data source or sink to be built with fairly small FIFOs with a guarantee that they will not over- or underrun.

The particular isochronous access method used in the Serial Bus was invented by David James and the author, both of Apple Computer, in early 1988. Ed Gardner of Digital Equipment Corporation proposed further enhancements of the protocol in 1992 which were accepted by the working group. This algorithm provides isochronous services without upsetting the basic access protocol by giving the highest priority access to a “cycle” master that maintains a common clock source. The cycle master tries to transmit a special timing request called a “cycle start” at specific intervals set by a “cycle synch” source (nominally 8 kHz ± 100 ppm, or 125 µsec ± 12.5 nsec). If a transfer is in progress when the cycle synch occurs, then the cycle start will be delayed, causing significant jitter in the start time of the transfer. Since this jitter is frequently unacceptable, the amount of time that the cycle start request was delayed is encoded within the request as a Transaction Layer quadlet write request broadcast to each node’s “cycle timer register”.

Each node with isochronous service has a 32-bit cycle timer register. The low-order 12 bits of the register are a modulo 3072 count, which increments once each 24.576 MHz clock period. The next 13 higher order bits are a count of 8 kHz cycles, and the highest order 7 bits count seconds. The cycle master copies its cycle timer register to all isochronous nodes with the cycle start request, synchronizing all nodes within a constant phase difference.

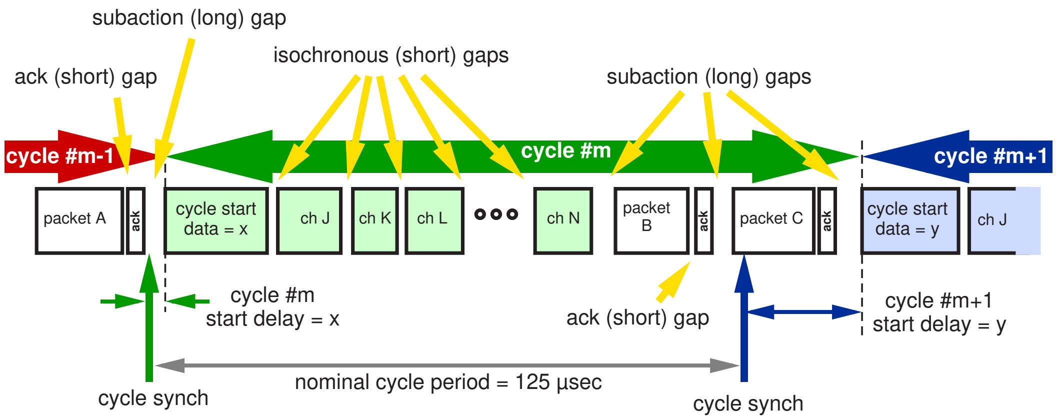

Figure 19 – Cycle Structure

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 10 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.

6. Conclusion

The IEEE P1394 High Performance Serial Bus has a number of innovations, not necessarily of a radical nature, but rather ones that make it easier to use or reduce the cost of implementing a reasonably high speed interface. The four inventions outlined in this paper are good examples of the philosophy of to working group: "high performance, but at a modest cost."

The author would like to thank the working group for its help in preparing the draft specification which provided much of the source material for this paper. In particular Forrest Crowell, Greg Floryance, David James, Florin Oprescu, and Thom Potyraj helped with their extensive comments on this text as it evolved over the years.