IRIS® HIPPI Administrator's Guide

Document Number 007-2229-002

CONTRIBUTORS

Written by Carlin Otto
Illustrated by Carlin Otto and Dan Young
Edited by Christina Cary
Production by Derrald Vogt
Engineering contributions by Thomas Skibo and Ken Powell

IRIS® HIPPI Administrator's Guide
Document Number 007-2229-002

Contents

Figures

Tables

Introduction

The IRIS® HIPPI product is a network interface controller board (hardware) and driver (software) providing data communication through the High-Performance Parallel Interface (HIPPI). The product provides HIPPI connectivity for CHALLENGE™ L and XL, and Onyx™ platforms.

The IRIS HIPPI hardware must be installed by a Silicon Graphics system support engineer (SSE) or other person trained by Silicon Graphics. The *IRIS HIPPI Installation Instructions* (shipped, in a sealed envelope, with each IRIS HIPPI board) contains complete details for hardware installation. The seal on the envelope must not be broken by anyone except the SSE.

Installation and configuration of the software can be done by customers and / or SSEs. This document, *IRIS HIPPI Administrator's Guide* (shipped with each IRIS HIPPI board), provides software configuration details. The online *IRIS HIPPI Release Notes* provide software installation instructions.

Support for Upper Layer Applications

IRIS HIPPI supports the following upper layer applications:

- standard UNIX applications:
  For Internet (IP) networking, IRIS HIPPI supports IP over HIPPI-LE in conformance with RFC 1374 guidelines. All IP applications can use the IP over HIPPI interface, just as they would IP over Ethernet or FDDI.
- IRIS HIPPI utilities:
  IRIS HIPPI includes utilities for monitoring, maintaining, and testing the IRIS HIPPI subsystem.
- customer-developed applications:
  IRIS HIPPI provides an application programming interface (API) that customers can use to develop their own upper-layer applications

(ULPs). See the *IRIS HIPPI API Programmer's Guide* (shipped with each IRIS HIPPI board) for details.

Style Conventions

This guide uses the following stylistic conventions:

`screen display`
Indicates system output, such as responses to commands that you see on the screen. Code samples, screen displays, and file contents also appear in this font.

`user input`
Indicates exact text that you must enter at a command line, such as commands, options, and arguments to commands.

variable
Indicates generic, place-holding variable names. Can indicate a user input variable, where you must replace the variable with text that you select.

`<xx>`
Indicates keys on the keyboard that you press; for example, press **`<Enter>`** means press only the key labeled **Enter**.

physical label
Indicates a label for a piece of hardware (for example, a pin, a wire, a port). Can also indicate the signal on a wire or pin.

command
Designates command and utility names.

file name
Indicates file names and file name suffixes.

[ ]
Encloses optional command arguments.

...
Denotes omitted material or indicates that the preceding optional items may appear more than once in succession.

Product Support

Silicon Graphics, Inc., provides a comprehensive product support and maintenance program for its products. If you are in North America and would like support for your Silicon Graphics-supported products, contact the Technical Assistance Center at 1-800-800-4SGI. If you are outside North America, contact the Silicon Graphics subsidiary or authorized distributor in your country.

Chapter 1

What is HIPPI?

This chapter is an introduction to the High-Performance Parallel Interface (HIPPI) protocol. The chapter provides an brief introduction to HIPPI, a description of the HIPPI protocol, some common configurations of HIPPI equipment, and how to obtain official HIPPI documentation.

Introduction to the HIPPI Protocol

This section provides a brief introduction to HIPPI.

HIPPI Terminology

HIPPI uses *source* to refer to the transmitting endpoint, host, network interface, or program.

It uses *destination* to refer to the receiving endpoint, host, network interface, or program.

A *word* in the HIPPI environment can be either 4 bytes (32 bits) or 8 bytes (64 bits), depending on the HIPPI implementation. When not clarified, both definitions apply. For example, “A burst consists of 256 words” means that a burst can be either 1024 bytes (256 words times 4 bytes) or 2048 bytes (256 words times 8 bytes).

How HIPPI Works

HIPPI is an extremely fast, simplex point-to-point protocol. HIPPI provides for transmission at 800 or 1600 megabits per second.[1] Before data can be sent from one HIPPI network interface (endpoint) to another, there must be both a physical link and an open connection between them. The physical link is

made up of one or more 25-meter sections of copper cable. Each section connects two HIPPI nodes. The nodes can be endpoints or intermediate HIPPI switches. The open connection is an agreement for data transfer from one endpoint to another, and is arranged with an exchange of signals. Once a connection is open, the entire physical link is dedicated to one-way communication from the source to the destination.

Figure 1-1 illustrates a configuration of HIPPI equipment with nine endpoint-to-endpoint links (listed below) of which three can be simultaneously active (engaged in open connections):

- A-source transmitting to A-destination (itself), B-destination, or C-destination
- B-source transmitting to A-destination, B-destination (itself), or C-destination
- C-source transmitting to A-destination, B-destination, or C-destination (itself)

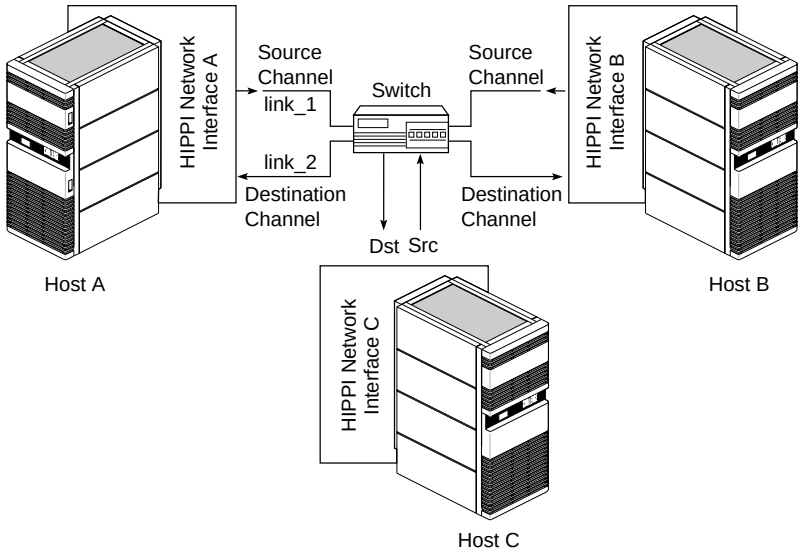


Figure 1-1 HIPPI Links and Connections

[1] IRIS HIPPI supports only 800 megabits per second.

An open connection consists of an exchange of signals between a source and a destination. During this exchange, the destination agrees to accept data exclusively from the source. Each endpoint-to-endpoint link supports one connection (that is, HIPPI is point to point). It is common for an interface's source and destination channels to have open connections with different hosts (for example, A-source connected to B-destination while A-destination is connected to C-source). To move data in both directions between two hosts, two endpoint-to-endpoint links and two connections are needed between the two hosts.

Unlike Ethernet, 802.5 Token Ring, or FDDI, HIPPI does not use a shared medium. Once a connection is established, the cable between the two HIPPI interfaces contains only packets transmitted by the source (that is, HIPPI connections are simplex). HIPPI packets may be seen by intermediate switches but not by other host interfaces. When one packet has been sent, the connection may be closed or kept open so that additional packets can be sent; however, each endpoint may not participate in another connection until the current one has been closed.

HIPPI communication is controlled by three basic functions: connection control, packet and flow control, and routing control (pertinent only when one or more switches are involved). Each of these is discussed separately in the subsections that follow.

Connection Control

One of the first things any HIPPI endpoint does upon startup is to assert its two outgoing **INTERCONNECT** signals and to look for assertion of its two incoming **INTERCONNECT** signals. Each channel (the source and the destination) has both incoming and outgoing **INTERCONNECT** signals. When both signals on a channel are asserted, the physical link between the local system and the system at the other end is ready for use. When the other system is a switch, the exchange of **INTERCONNECT** signals occurs between the endpoint and the switch, not between endpoints.

Before a source (transmitting) HIPPI network interface can send a packet, it must open a connection to one destination HIPPI endpoint. The source interface is always the initiator for opening the connection. To open a connection, the sender issues a connection request by asserting the **REQUEST** signal on the link. Each connection request includes an I-field

(described in the section “The I-field”). The I-field mainly contains routing information, used by any switches encountered along the path to the destination.

The destination endpoint accepts a connection by asserting its **CONNECT** signal in response to the request. If the destination endpoint is unwilling to accept the connection or if there is a problem with the connection request (for example, bad parity on the I-field or incompatible word size), the connection request is denied (that is, acknowledged, then rejected). The transmitter must wait and try again later or forgo the communication. If the destination is unreachable (for example, a broken physical link, a powered-down or dysfunctional network interface), there is no response and the source program times out.

When a switch exists between the source and destination, the source receives its connection rejections from the switch, not directly from the destination. The rejection can be caused by any of the following conditions, and it is not possible to distinguish among them (except as explained below):

1. The destination is malfunctioning.
2. The destination refuses to accept the requested connection.
3. The connection request has an error.
4. Asection of the physical link to the destination is busy (currently engaged in another connection).

A feature is available that allows the source to be informed of rejections that are due to error conditions (items 1-3 above) but not to be bothered when the rejection is due to a busy link (item 4). This feature is called *camp-on*. By setting the camp-on bit in the I-field, the source can program the switches to hold onto the connection request until the busy link to the destination becomes available.

When the camp-on bit is set, the first switch enqueues the connection request if it finds any link along the path to the destination busy. The switch periodically checks to see if the link has become available. When the link becomes available, it sends the connection request. A switch continues to wait until it sends the **REQUEST** to the ultimate destination endpoint or until the source aborts the connection request. If a number of sources are all trying to send data through the same link, the camp-on feature ensures fair (first come, first served) access to the link.

Once opened, a HIPPI connection may be kept open for as long as the two endpoints maintain it. Either endpoint may terminate the connection at any time; however, the source network interface is usually the initiator.

Packet and Flow Control

Once a connection is open, one or multiple packets may be sent. The destination indicates it is ready to receive data by sending a **READY** signal to the source endpoint. Each **READY** allows the source to transmit one HIPPI *burst* (as explained below). All HIPPI source endpoints are required to be capable of enqueuing a minimum of 63 **READY**s. There is no minimum requirement for a destination's ability to generate **READY**s.[2] By sending ahead and enqueuing **READY**s, the two endpoints can optimize the throughput on their connection.

The source delineates its packets with the **PACKET** signal: at the beginning it asserts the **PACKET** signal, and at the end it deasserts the signal. A HIPPI packet consists of one or more bursts, as illustrated in Figure 1-2. Each burst contains 256 words, except in the case where the burst is *short* (as described below). The size of each word depends on the source's data bus (32 or 64 bits, as indicated by a bit in the I-field).[3] At the end of each burst, the source generates a checksum (LLRC) so that the destination can detect any errors in the received data; in addition, each word has four bits of parity for error checking.

[2] The source channel on the IRIS HIPPI board can enqueue up to 65,535 **READY**s; the destination channel can generate up to 255 outstanding **READY**s.

[3] The IRIS HIPPI board supports 32-bit words only.

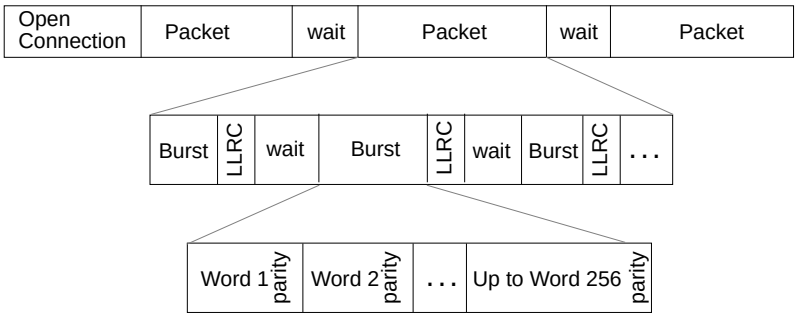


Figure 1-2 HIPPI Packets and Bursts

The HIPPI protocol requires very small waiting periods between packets and between bursts. These required periods are counted in nanoseconds and are imperceptible to the user; however, in normal operation there may be noticeable pauses between bursts (for example, when the source is waiting to receive a **READY**).

As long as the source has **READY**s, it can transmit data as fast as it is capable of transmitting (but no faster than the protocol allows: 25 million words per second). When the sender has sent all the data for one packet, it indicates the end of the packet, using the **PACKET** signal. Indicating the end of the packet is necessary because HIPPI allows packet size to be undefined (indeterminate) at the start of the packet. A sender could essentially send an infinite-sized packet by keeping the **PACKET** signal asserted at all times.

A packet's first burst often contains some kind of header (for example, a HIPPI-FP header as described in "The FP Header"). The first burst can contain header only, or header and user data. In other words, the first words of user data can be in the first burst or the second. If the source program is generating HIPPI-FP packets, it can indicate the location of the packet's first word of user data by setting the B bit in the HIPPI-FP header.

Either the first or the last burst of a packet (but not both) can be less than 256 words. This burst is referred to as a *short* burst. Usually, the last burst is the short one. When the first burst is the short one, it contains only the header and, optionally, control information. The first word of the packet's user data is, in this case, located in the second burst, and the final burst may be padded to meet the 256-word length requirement. When the last burst is the short

one, the packet's final burst never needs to be padded and the first word of user data may be included in the first burst.

Once the end of the packet has been indicated, the source has the option of keeping the connection open to transmit additional packets or of closing the connection.

Routing

The I-field contains HIPPI routing information in its 24-bit Routing Control field. This information is interpreted only by intermediate systems (switches); the Routing Control information does not need to be interpreted when the connection is directly between two endpoints.

The addresses in a Routing Control field can be in "logical addressing" or "source addressing" format. The format is indicated by the Path Selection bits of the I-field. The two formats cannot be used simultaneously in one I-field; however, both formats can be used simultaneously in one HIPPI local area network (LAN).

Note: The word *source* in "source addressing format" does not mean that the address is the source's address; it refers to the fact that the address, supplied by the source endpoint, defines the complete path (route). ♦

Logical Addressing

With logical addressing, the Routing Control field contains two 12-bit addresses: a destination (receiver's) address and the source (sender's) address, as illustrated in Figure 1-3. The order in which the addresses are placed within the field is defined by the I-field's Direction bit, as illustrated in Figure 1-3.

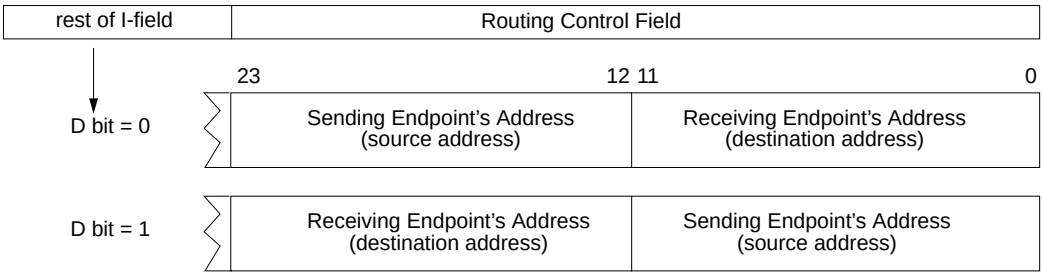


Figure 1-3 Routing Control Field With Logical Addressing

When a HIPPI LAN uses logical addressing, each HIPPI network interface (endpoint) within the LAN is assigned an address that is unique within that LAN. One address can be used for both the source and destination channels of a network interface, if desired. Assignment of these addresses is a local matter; the addresses do not need to be unique outside the particular HIPPI network. Logical addresses have the formats described in Table 1-1. Some of the addresses are reserved for special purposes.

Table 1-1 Logical Addressing Formats

Logical Address (binary)	Number of Addresses	Usage
xxxx x0xx xxxx	4032	Endpoint addresses
1111 110x xxxx	32	Local assignment to network services
1111 111x xxxx	32	Reserved for global assignment
1111 1111 1111	1	Address is unknown

Each switch maintains a "map" of its LAN and uses a routing table to select the path along which to open a connection for each request. For example, Figure 1-4 illustrates a scenario where two paths are available between endpoints A and B. When endpoint A requests a connection to endpoint B, switch 1 can select either of these paths.

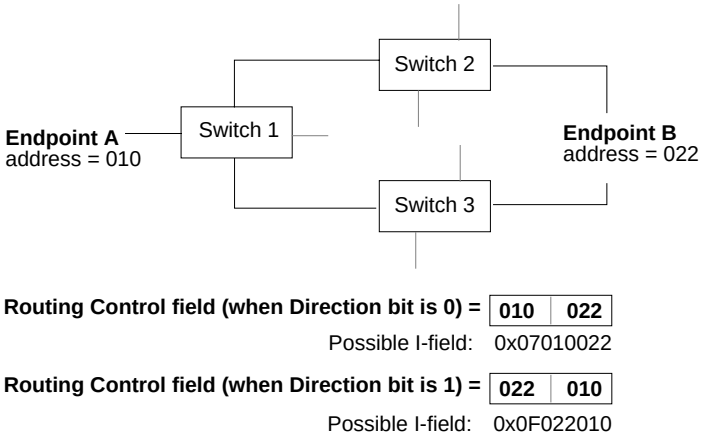


Figure 1-4 Routing With Logical Addressing

The 12 bits make it possible to create 4096 unique addresses. The HIPPI-SC standard reserves 64 of these addresses, leaving 4032 addresses available for local assignment to HIPPI end points. 4032 is the maximum number of destination endpoints that can exist on one HIPPI LAN using logical addressing.

Source Addressing

The addresses used for source addressing are of variable lengths, from 1 to 24 bits. When the Path Selection bits in the I-field indicate that source addressing is being used, the Routing Control field contains a list of port identifiers, as illustrated in Figure 1-5. The I-field's Direction bit determines the order in which the port identifiers are placed within the field and the alignment of (placement for) the addresses, as illustrated in Figure 1-5.

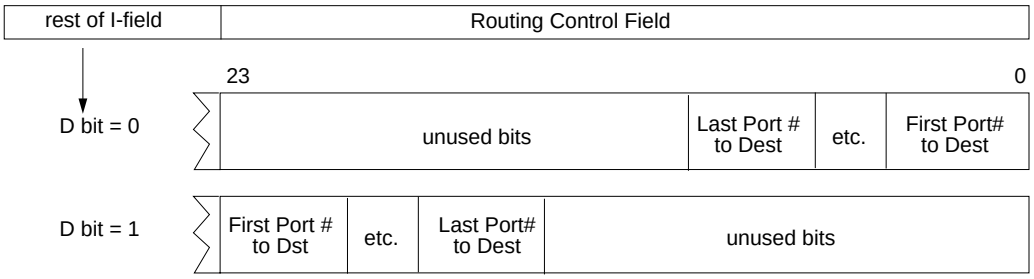


Figure 1-5 Routing Control Field (As Created by Sender) With Source Addressing

Each port identifier uniquely identifies one port within a switch. A port is a pair of physical links: both a source and a destination. For example, a 4x4 switch has 8 physical links to 4 systems, and for this it uses 4 port identifiers, as illustrated in Figure 1-6. Port identifiers are unique among all the ports on the same switch, but not among all the ports within the LAN. For example, a LAN with 5 switches might easily have 5 port identifiers of "1." Figure 1-6 is an example of the port identifiers used in a LAN with 2 switches.

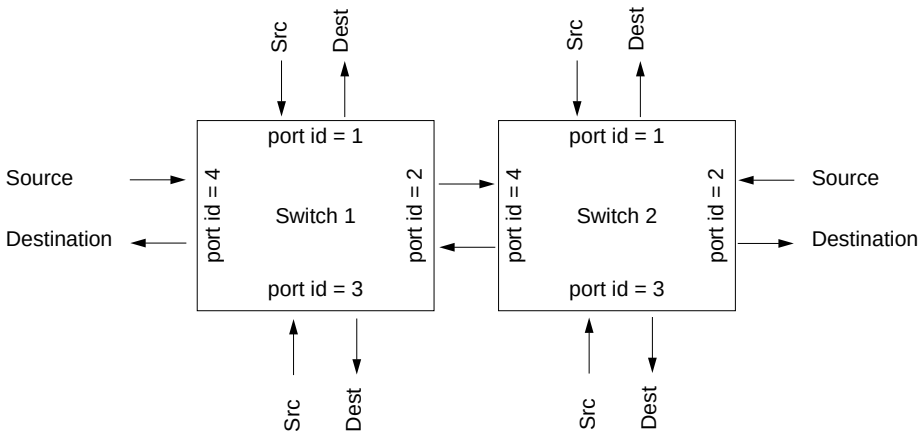


Figure 1-6 Switches and Port Identifiers

A Routing Control field in source address format is interpreted as a series of "stepping stones" leading to the destination in the following manner:

1. The first switch (the one attached to the source endpoint) reads the first port identifier, opens a connection at that outgoing port, and sends the I-field (that is, the connection request).
2. If the system at the end of that physical link is another switch, it reads the second port identifier, opens a connection at that outgoing port, and sends the I-field.
3. And so on, until the receiving system is the destination endpoint.

When the port identifiers are followed sequentially, they create the path between the two endpoints. Each path (address) consists of a list of all the outgoing ports through which the connection request must pass in order to reach the destination. For example, in the simplest configuration, where one switch exists between two network interfaces, the address consists of one port identifier: the one to which the receiving interface is connected, as illustrated by Example 1 in Figure 1-7. When two switches exist between the interfaces, the address consists of two port identifiers, as illustrated by Example 2 of Figure 1-7.

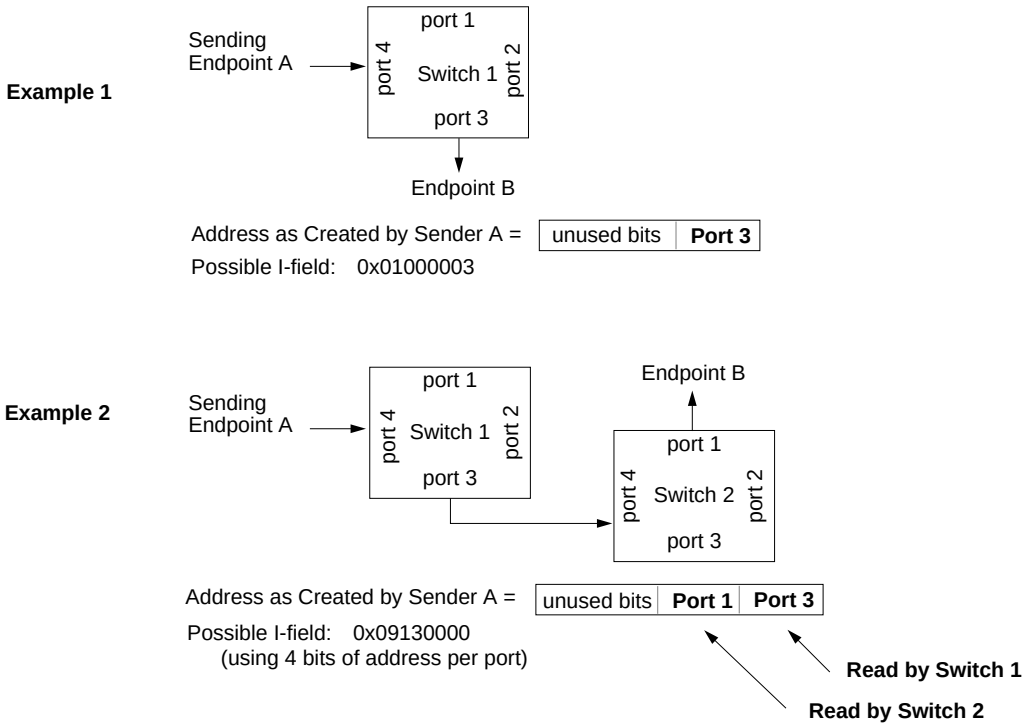


Figure 1-7 Port Identifiers for Source Addressing

The Direction bit in the I-field defines whether each port identifier should be read from the most significant or least significant end of the Routing Control field. For example, Figure 1-8 illustrates two addresses that endpoint A might use to open a connection with B.

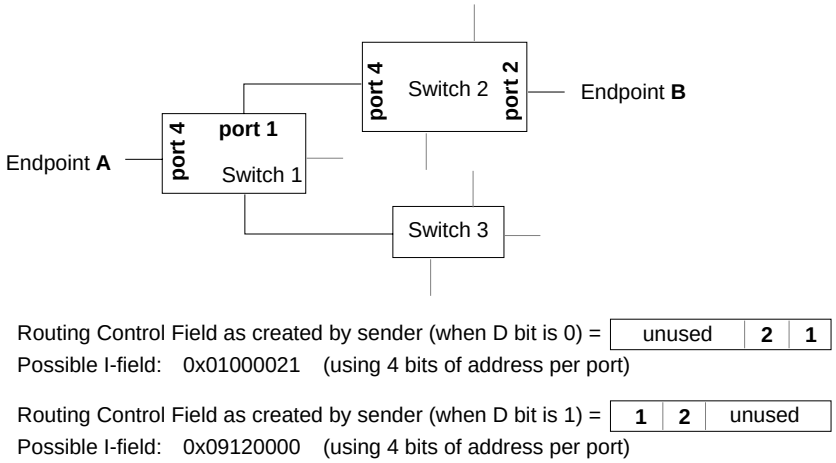


Figure 1-8 Routing With Source Addressing

Each HIPPI host within the LAN maintains a table of paths (addresses in source addressing format) for reaching each of the other endpoints. With each of its connection requests, a source attaches one of these paths, thus indicating how to reach the destination. The path is completely defined by the sending endpoint.

Unlike logical addresses (which are not altered enroute to the destination), addresses in source addressing format are changed by each switch that handles the I-field. The source program creates a list of outgoing port numbers that define a path from the sender to the receiver. By the time the packet arrives at its destination, the address has been altered so that it defines the return path (that is, the path from the receiver back to the sender). This change is brought about by each switch removing the outgoing port identifier that it reads, shifting the remaining bits into alignment, and adding an incoming port identifier (that is, the port through which the I-field just arrived), as illustrated in Figure 1-9.

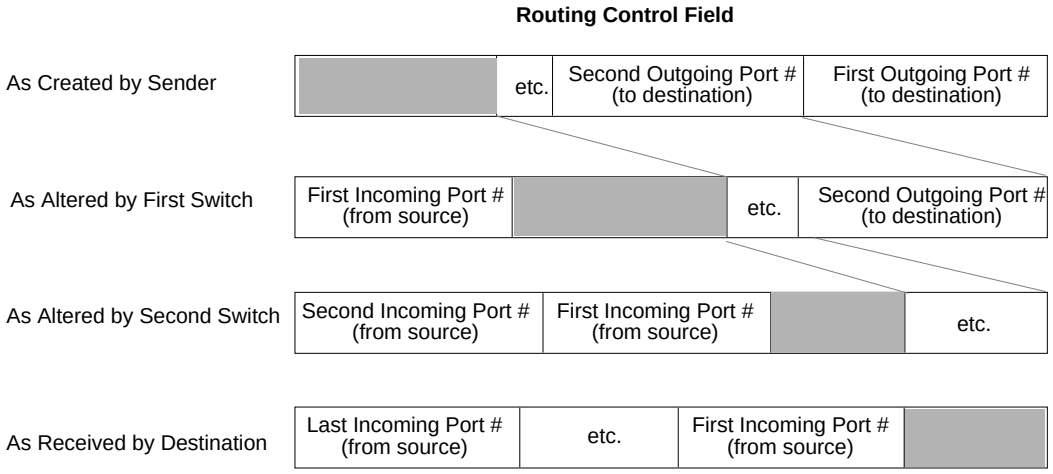


Figure 1-9 How Switches Alter Source Addresses

A destination program can copy a received Routing Control field into its own I-field and simply change the setting of the Direction bit to open a return connection, thus bypassing the table lookup procedure. Normally, the source that first creates the Routing Control field sets the D bit to zero and places the address bits in the least significant positions of the Routing Control field. The receiver changes the setting for the D bit and uses the received Routing Control field exactly as it is received. In this manner, the port identifier labeled *Last Incoming Port #* in Figure 1-9 becomes the *First Outgoing Port #* for the return connection.

Port identifiers can be one to six bits. The number of bits varies from switch to switch. The size of the port identifier is the number of bits needed to uniquely identify all the possible ports on a switch. For example, a 4x4 switch has four ports and requires at least two-bit port identifiers (binary port identifiers 00, 01, 10, and 11). If a switch is capable of being enlarged, it may use large-sized port identifiers (for example, five or six bits) to avoid a reconfiguration of all the LAN's routing tables when the switch is upgraded.

The I-field's 24-bit Routing Control field limits the number of port identifiers that can be contained in an address, as summarized in Table 1-2.

Table 1-2 Maximum Number of Port Identifiers in Routing Control Field

Number of Bits Used in Port Identifier	Maximum Number of Port Identifiers Possible in Routing Control Field
1	24
2	12
3	8
4	6
5	4
6	4

The Protocol

This section describes the format for the HIPPI I-field and FP header.

The I-field

The I-field is defined by the HIPPI-SC standard. The format for the 32-bit HIPPI I-field (also called CCI) is shown in Figure 1-10, and its fields are explained in Table 1-3.

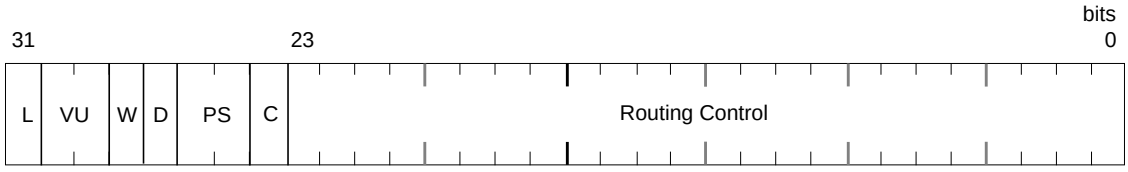


Figure 1-10 I-field Format

Table 1-3 Fields of the HIPPI I-field

Field	Bits	Description
L	31	Local or Standard Format: 0=Bits 30:0 of I-field conform to the usage described in this table. 1=Bits 30:0 are implemented in conformance to a private (locally-defined) protocol.
VU	30:29	Vendor Unique Bits: Vendors of end-system HIPPI equipment may use these bits for any purpose. Switches do not alter or interpret these bits.
W	28	Width: 0=The data bus of the transmitting (source) HIPPI is 32 bits wide for 800 megabits/second transmission. 1=Source's data bus is 64 bits wide for 1600 megabits/second transmission.
D	27	Direction: 0=Least significant bits of Routing Control field contain the destination address for the current switch to use. 1=Most significant bits of Routing Control field contain the destination address for the current switch to use.
PS	26:25	Path Selection: 00=Source routing. 01=Logical routing. Switch must select first route from a list of routes. 10=Reserved. 11=Logical routing. Switch selects any (or best) route from its list.

Table 1-3 (continued) Fields of the HIPPI I-field

Field	Bits	Description
C	24	Camp-on: 0=Switch rejects connection request immediately if port to destination is busy. 1=Switch holds connection request if port to destination is busy and establishes connection when the port becomes free or when source aborts the request.
Routing Control	23:0	Address: This field contains addressing / routing information. The contents are in source routing or logical routing format, as indicated by the PS field. For source routing, the field contains a list of switch port identifiers that, when followed, lead to the destination. For logical addressing, the field contains two 12-bit addresses (receiver's and sender's) that are used by the intermediate switches to select a route from a table.

The FP Header

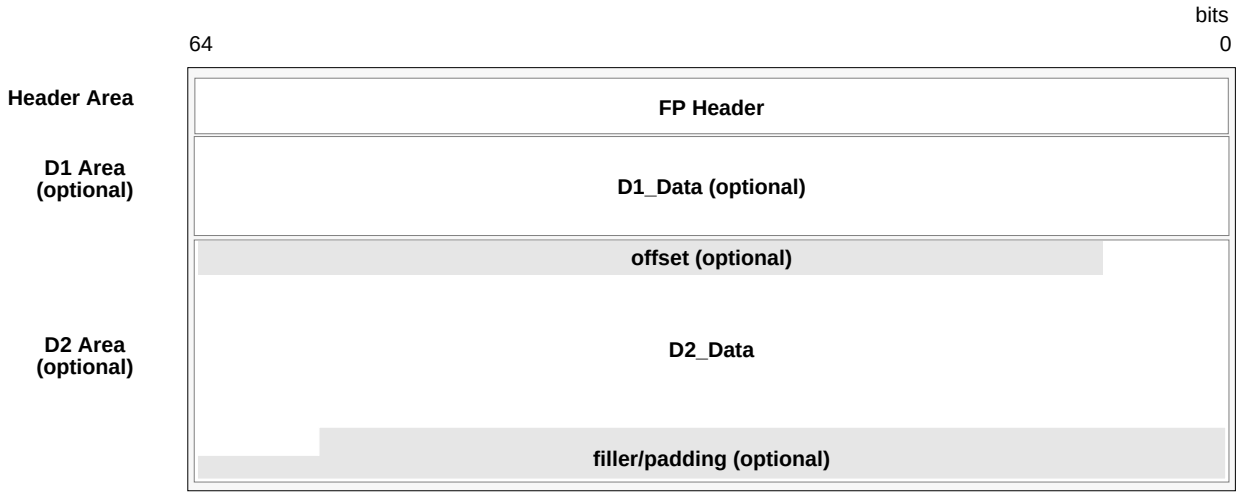
The FP header is defined by the HIPPI-FP standard. When a HIPPI endpoint is HIPPI-FP conformant, all the packets it transmits and / or receives (without error) are HIPPI-FP packets. The first burst of each of its transmitted packets contains an FP header, and it looks for an FP header in the first burst of each received packet. A HIPPI-FP packet consists of three segments, listed below and illustrated in Figure 1-11. Each segment is eight-byte aligned (that is, contains an integral number of 64-bit words).

- Framing Protocol header:
  This area contains the 64-bit HIPPI-FP header, described in more detail in Table 1-4 on page 19.
- D1_Area:
  This optional area, if present, must be completely contained in the first burst. It may contain control information (the D1 data set), it may be defined for padding purposes only, or it may serve both of these functions. The D1 area can be 0 to 255 words in size; however, regardless of the word size used by a HIPPI implementation, the D1 area must contain an integral number of 64-bit words. So, for a 64-bit

implementation, the D1 area can have up to 255 words. For a 32-bit implementation, the D1 area can have up to 254 words.

The D1 data set (located within the D1 area) is optional. The maximum size of any D1 data set is 1016 bytes (that is, 254 32-bit words), thus allowing the FP header (8 bytes) and D1 data to fit in the first burst of any HIPPI implementation (for example, a burst made of 32-bit words). The content and format of the D1 data is locally defined and the data must be self-defining. For example, each upper layer application (with its own ULP-id) could use a different format and length for its D1 data.

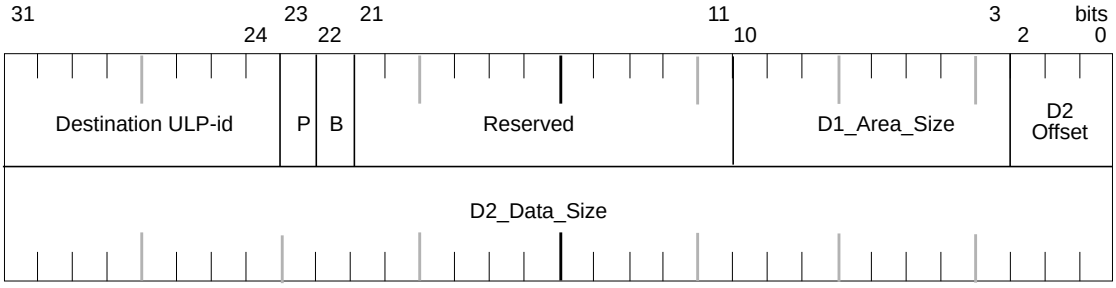
- D2_Area:
  The optional D2 area contains the user / application data. This area can be 0 to 4-gigabytes minus 1-byte in size, or it can be defined as indeterminate. The size of this area must be an integral number of 64-bit words. The area may contain padding (an offset and possibly filler).



NOTE: The size of each included area must be an integral number of 64-bit words. For IRIS HIPPI, the first word of each area must be 8-byte aligned.

Figure 1-11 HIPPI-FP Packet Format

The 64-bit FP header describes the HIPPI packet using six fields, illustrated in Figure 1-12 and described in Table 1-4.



P = D1 data are included/not included in this packet
B = First word of D2 data is in first/second burst

Figure 1-12 FP Header Format

Table 1-4 Fields of the HIPPI-FP Header

Field	Bits	Description
ULP-id	63:56	The 8-bit upper layer protocol identification field identifies a system's upper layer protocols. A transmitting application uses this number to specify the upper layer protocol of the intended recipient of the packet. A receiving HIPPI subsystem can use this number to demultiplex incoming packets among a number of upper layer protocols (or applications) and to determine whether an intended recipient is known or not.
P bit	55	The 1-bit present bit indicates whether or not the packet contains D1 data. Note that the D1_Area may be present, even when there is no D1 data.
B bit	54	The 1-bit burst boundary bit indicates which burst contains the first byte of D2 data. D2 data can be included in the first burst or it can start with the first word of the second burst.

Table 1-4 (continued) Fields of the HIPPI-FP Header

Field	Bits	Description
D1 Area Size	42:35	The 8-bit D1 area size field indicates the number of 64-bit words in the D1_Area of this packet. The area does not necessarily contain valid data; that is, the area may be defined for padding purposes only. Note that the size is always stated in 64-bit words, regardless of the implementation's word size.
D2 Offset	34:32	The 3-bit D2 offset field indicates the number of bytes between the last byte of D1 data and the first byte of D2 data.
D2 Data Size	31:0	The 32-bit D2 data size field indicates the number of bytes of D2 data included in this packet. Bytes of offset or fill are not included in this count.

Figure 1-13 illustrates a HIPPI-FP packet with D1 data where the first burst of the packet contains only the FP header and the D1 data. Figure 1-14 illustrates some of the HIPPI-FP packets that are commonly created. Examples 2, 3, and 4 in Figure 1-14 illustrate how the D1 area can be used to position the D2 data in the second burst.

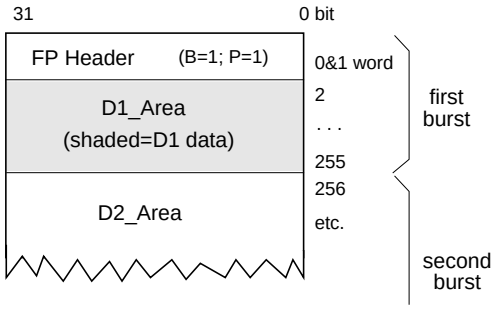


Figure 1-13 Sample HIPPI-FP Packet

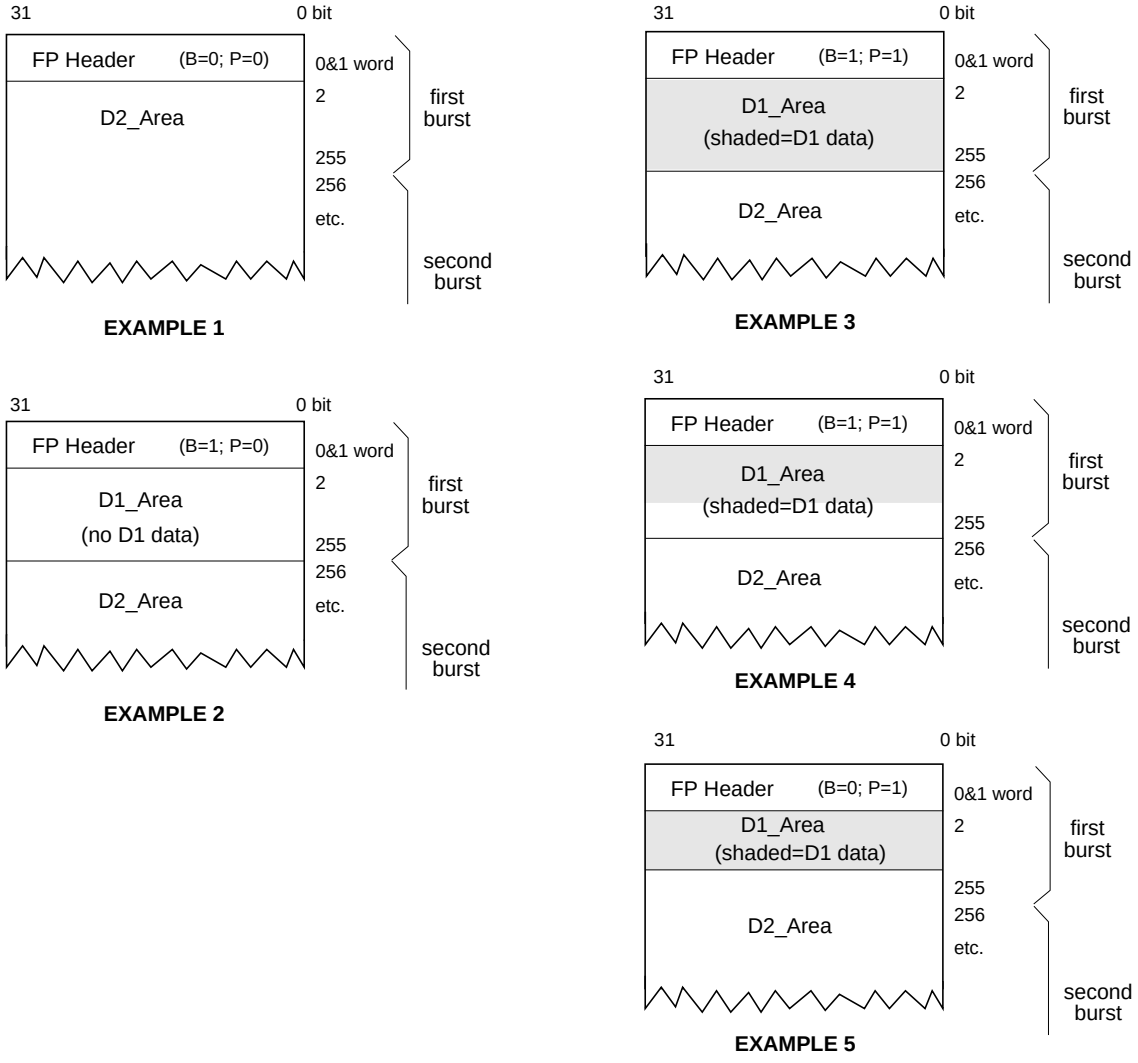


Figure 1-14 Some Common HIPPI-FP Packets

The Signals

The signals at the HIPPI-PH layer of each physical link are used for controlling the connection, packet boundaries, and data flow. These signals are described in Table 1-5 and illustrated in Figure 1-15. Within the figure, the signals are numbered to represent the order in which they are asserted when power is first applied at the two endpoints. The two **INTERCONNECT** signals are not dependent on any other signal; they are asserted as the HIPPI hardware receives power and becomes active. Each of the other signals is asserted only when all the signals before it have been observed.

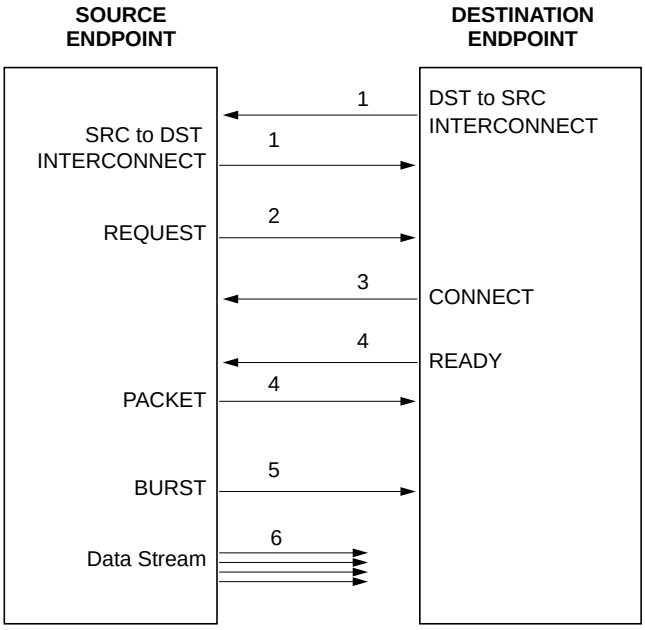


Figure 1-15 HIPPI Signals Used on Each Point-to-Point Link

Table 1-5 HIPPI Signals

SIGNAL	DESCRIPTION
Generated by the source on this physical link	
Source-to-Destination INTERCONNECT	When asserted, indicates source is attached and ready for action. This signal is sometimes referred to as **SDIC**.
REQUEST	When asserted, indicates source is requesting a connection to be opened. This signal is accompanied by an I-field. When deasserted, indicates the source is closing the connection (if one is open) or aborting the connection request (if no connection is currently open).
PACKET	When asserted, indicates a packet is in progress. When deasserted, indicates the end of a packet. This signal does not indicate that any data is being sent; it only delineates the boundaries of a packet.
BURST	When asserted, indicates data is being sent. This signal is accompanied by one burst of data. When deasserted, no data is being sent.
Generated by the destination on this physical link	
Destination-to-Source INTERCONNECT	When asserted, indicates destination is attached and ready for action. This signal is sometimes referred to as **DSIC**.

Table 1-5 (continued) HIPPI Signals

SIGNAL	DESCRIPTION
CONNECT	When asserted, indicates destination is accepting the connection (opening a connection in response to a **REQUEST** signal). When deasserted, indicates destination is closing the connection (if one is open) and is now available for a new connection.
READY	When pulsed, indicates destination can accept (has buffer space available) one burst of data. Destination can send any number of these to source; however, the source is required by the HIPPI-PH standard to queue only 63.

HIPPI Network Configurations

This section describes some of the common configurations of HIPPI equipment. Because HIPPI is a simplex point-to-point protocol, only one source can transmit data onto the transport medium (cable) between two endpoints. Two physical links (one cable for each source) are required for bidirectional communication. This aspect of HIPPI makes it quite different from protocols such as Ethernet, FDDI, or 802.5 Token Ring.

Basic HIPPI Configurations

A basic (non-networked) HIPPI configuration consists of two endpoints, one sending and the other receiving, as shown in Figure 1-16.

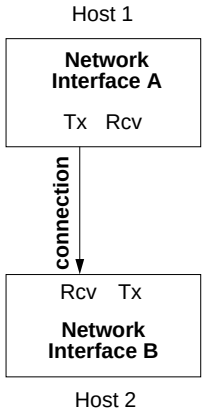


Figure 1-16 Basic HIPPI Configuration

To exchange data in both directions, two physical links and two connections are required between the two endpoints, as illustrated by the examples in Figure 1-17. Each endpoint's source channel must open a connection with the destination of the other endpoint.

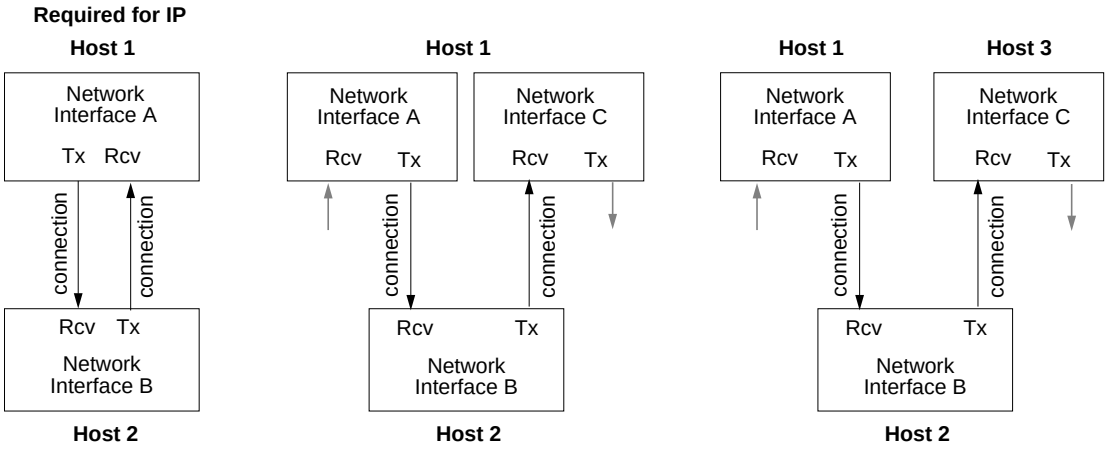


Figure 1-17 Three Variations of the Basic Configuration

The IRIS HIPPI network interface board has two channels (visible at the I/O panel). It treats each one as a separate entity, so each IRIS HIPPI network interface supports two autonomous, simultaneous connections: one sending and one receiving. The two connections can be to two different endpoints (as shown by the examples on the right in Figure 1-17) or to the same endpoint (as illustrated by the example on the left in Figure 1-17). IP communication over HIPPI requires the latter configuration.

HIPPI Local Area Network Configurations

One or more HIPPI switches may be placed along the endpoint-to-endpoint link, making it possible to configure a number of endpoints into a HIPPI local area network (LAN, or fabric). Configuring the endpoints in this way does not alter the fact that each communication is a point-to-point connection. The switches are cross switches, not “routers.”

When a switch is included in a HIPPI configuration, each endpoint has a number of hosts with which it can communicate (one at a time). Figure 1-18 illustrates a HIPPI LAN with one switch. The switch in this illustration is a 4 x 4, meaning that the switch can have four systems (8 HIPPI channels) attached to it. The switch supports four simultaneous connections. For example, in Figure 1-18, any one of the following connection scenarios could be occurring at any single point in time:

- A and D could be exchanging TCP/IP traffic. There would be two connections open between them. C and D could be doing the same. This scenario opens all four possible connections.
- A could be transmitting to B, while B transmitted to C, C to D, and D to A. This scenario also opens four connections.
- A and C could be exchanging bidirectional traffic. D could be transmitting to B. Only three connections are open in this scenario.

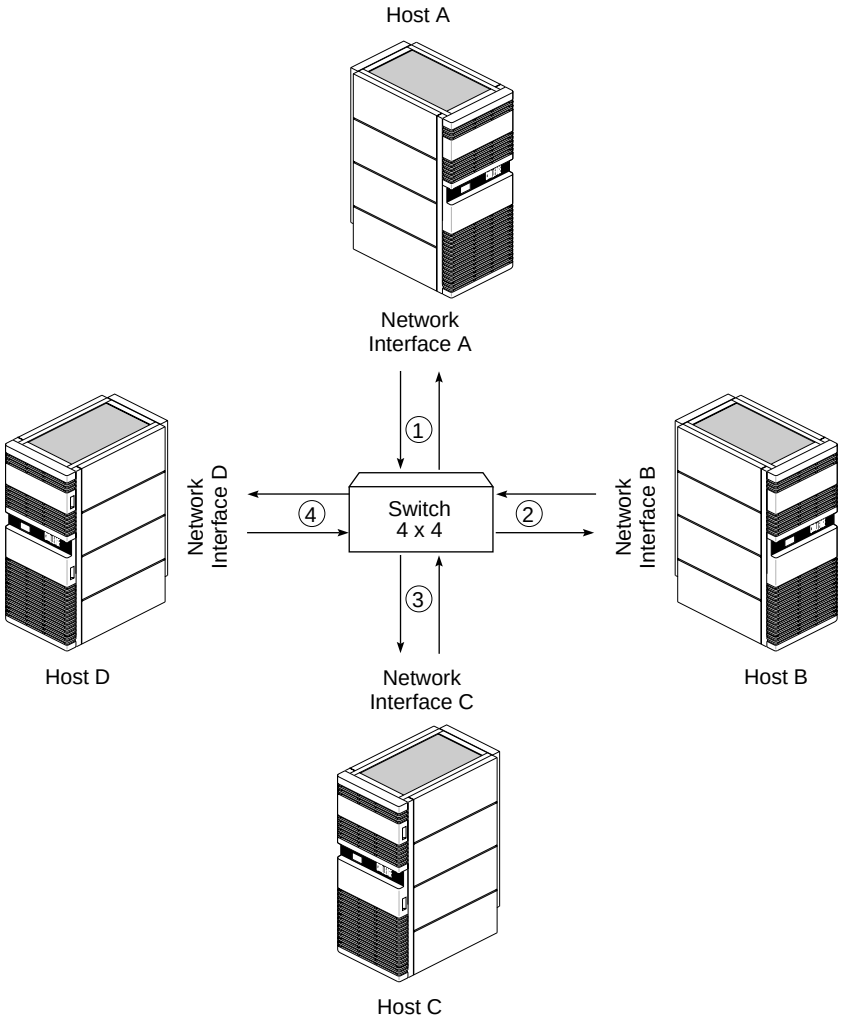


Figure 1-18 HIPPI LAN Configuration With One Switch

Figure 1-19 illustrates a LAN with multiple switches, and Figure 1-20 illustrates a complex HIPPI LAN including a long-distance fiber optic link and multiple ports between switches to improve connection setup time by reducing the probability of encountering a busy link.

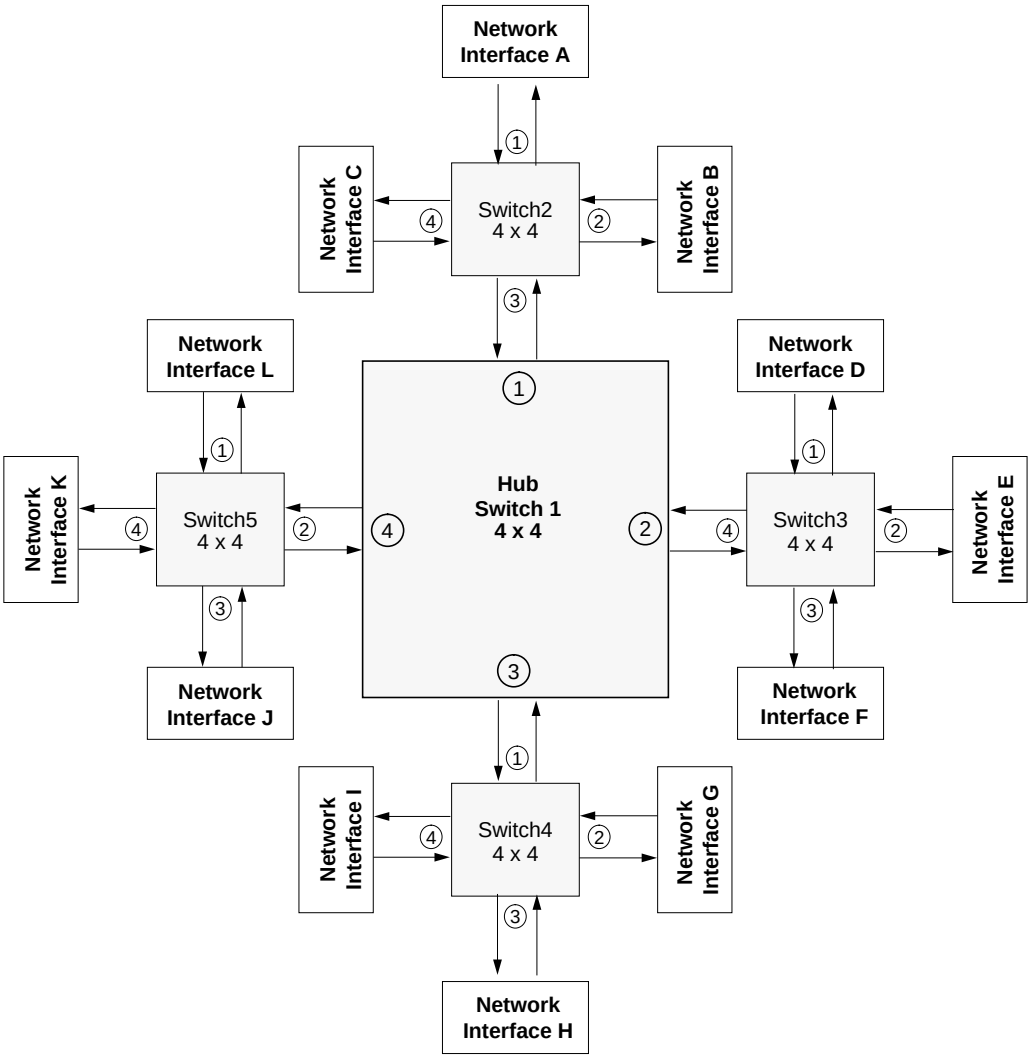


Figure 1-19 HIPPI LAN Configurations With Multiple Switches

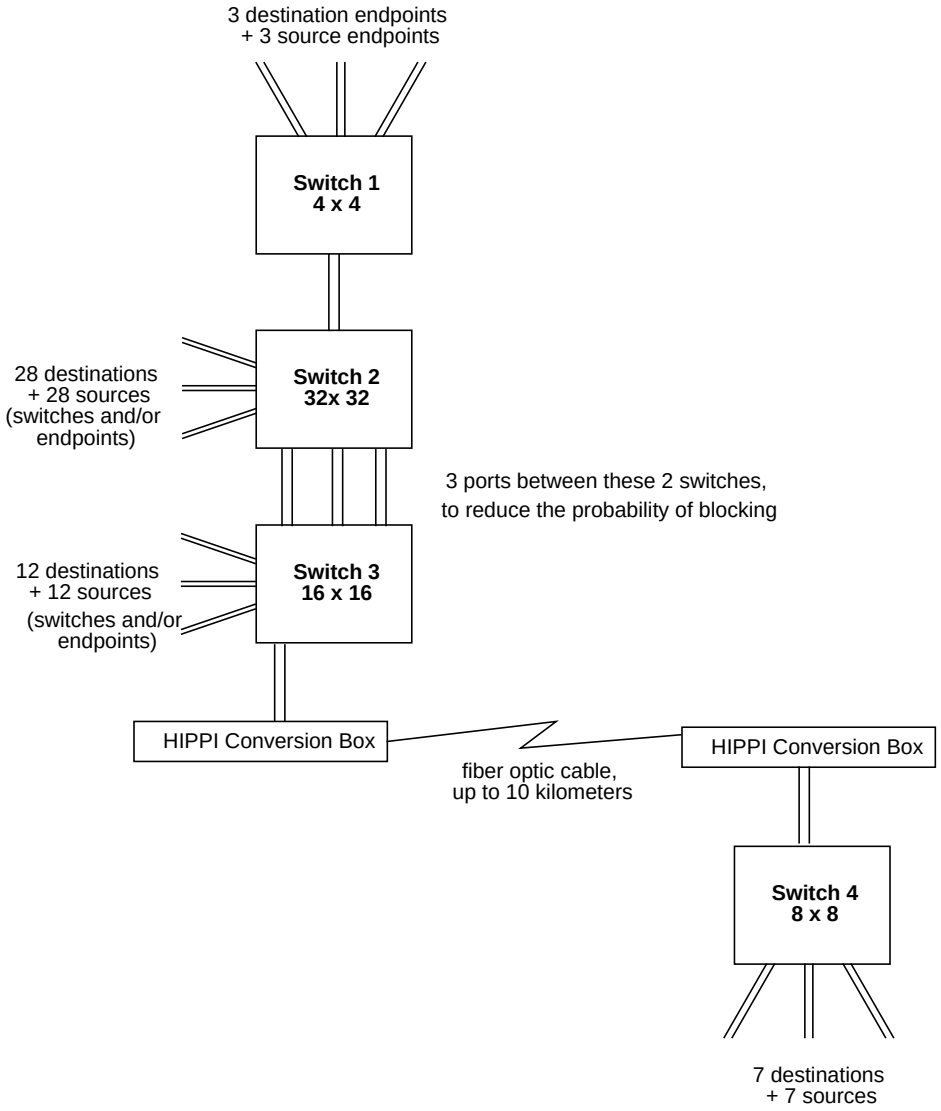


Figure 1-20 Complex HIPPI LAN Configuration

The maximum number of switches and endpoints within a LAN is limited by three factors:

- When logical routing is used, the 12-bit HIPPI address (half of the Routing Control field) limits the number of unique endpoint addresses to 4096. It is possible for a site to implement this number of networked HIPPI endpoints; however, to be compliant with the HIPPI-SC standard, 64 reserved addresses should not be assigned to local endpoints (that is, hosts). This limits the number of endpoints to 4032 per LAN. There is no limit to the number of switches when logical routing is used.
- When source routing is used, the I-field's 24-bit Routing Control field limits the number of port identifiers that can be included in the list. The exact number depends on the sizes of the port identifiers used by the switches along the specific endpoint-to-endpoint path, as explained in "Source Addressing." Each port identifier in the Routing Control field represents one switch along the path. Table 1-6 summarizes the maximum number of switches along any point-to-point path within a HIPPI LAN, assuming that all switches along that path use port identifiers of the same size. (This assumption does not reflect actual site configuration practices, but is useful here for illustration of a point.) When source routing is used, the number of switches and endpoints that are possible is not limited; however, the number of switches between any two endpoints within the LAN is limited. This limit affects the configuration of the LAN.
- If a LAN is built according to the guidelines in Appendix B of RFC 1374, the recommended maximum number of hops (switches) between any two endpoints is three. This limit has major implications for the structure and size of a HIPPI LAN. The structure is limited to a single hub switch with satellite switches attached to the hub's ports, but no switches attached to any satellite ports. Figure 1-19 shows an example of an RFC 1374-compliant LAN. Table 1-7 summarizes the maximum number of switches and endpoints possible for a LAN in which switches of only one size are used throughout the LAN.

Table 1-6 Maximum Number of Switches Along Any Single Point-to-Point Path When Using Source Addressing

Number of Bits Used for All Port IDs in Routing Control Field	Max. Number of Switches Along Any Single Point-to-Point Path
1	24
2	12
3	8
4	6
5	4
6	4

Table 1-7 Maximum Number of Switches and Endpoints on a LAN Built in Accordance With RFC 1374, Appendix B Guidelines

Size of All Switches Within LAN				
4 x 4	**8x 8**	**16 x 16**	**32 x 32**	**64 x 64**
5 switches / 12 endpoints	9 switches / 56 endpoints	17 switches / 240 endpoints	33 switches / 992 endpoints	65 switches / 4032 endpoints

The HIPPI Standards and Documentation

The documents listed below provide the official definitions of what HIPPI is and how it works.

- ANSI HIPPI-PH
  The *HIPPI Mechanical, Electrical, and Signalling* document defines the standard for the physical layer: electrical and mechanical aspects of HIPPI cables, as well as the behavior of HIPPI physical interfaces (including the HIPPI signals, like **SDIC, DSIC, REQUEST**, **CONNECT, READY**, **PACKET**, and **BURST**).
- ANSI HIPPI-SC
  The *HIPPI Physical Switch Control* document defines the standard for switch behavior, routing methods, and connection management. The HIPPI I-field is defined by this standard.
- ANSI HIPPI-FP
  The *HIPPI Framing Protocol* document defines the standard for data framing issues: how a packet is formed, how its data contents are described and interpreted. The HIPPI-FP packet (FP header, D1_Data, and D2_Data) is defined by this standard.
- ANSI HIPPI-LE
  The *HIPPI Encapsulation of ISO 8802-2 (IEEE 802.2) Logical Link Control Protocol Data Units* (HIPPI-LE) standard defines the method for encapsulating (and thus interoperating with) 802.2 compliant data link layers such as FDDI, 802.5 Token Ring, and CSMA / CD (Ethernet).
- ANSI HIPPI-IPI-3 for Disk
  The *HIPPI Intelligent Peripheral Interface—Device Generic Command Set for Magnetic and Optical Disk Drives* standard defines an upper-layer protocol for interfacing disks to the HIPPI subsystem.
- ANSI HIPPI-IPI-3 for Tape
  The *HIPPI Intelligent Peripheral Interface—Device Generic Command Set for Magnetic Tape Drives* standard defines an upper-layer protocol for interfacing tapes to the HIPPI subsystem.
- RFC 1374
  IP and ARP on HIPPI, by J. Renwick and A. Nicholson (October 1992) defines the protocol for using the IP suite of network and transport layer protocols over HIPPI.

The ANSI documentation for HIPPI standards is maintained by the American National Standard of Accredited Standards Committee (ANSI X3T9.3). Copies of the ANSI standards listed above can be obtained by writing or calling the following address:

Don Tolmie, Chairperson
Los Alamos National Laboratory
C-5, MS-B255
Los Alamos, NM 87545
Telephone: 505-667-5502
Internet email: det@lanl.gov

Implementation Details for IRIS HIPPI

This section describes some of the details of the Silicon Graphics implementation of the HIPPI protocol.

Application Programming Interface

The IRIS HIPPI driver provides access and control of the HIPPI subsystem to upper-layer applications. The upper layer-applications that are shipped with the IRIS HIPPI product are the IRIS HIPPI-LE module serving the IP network stack, and the IRIS HIPPI utilities. Upper-layer programs can also be developed by customers, using the IRIS HIPPI application programming interface.

Customer-developed applications can define their own upper-layer protocol (ULP) and can program the IRIS HIPPI driver to implement HIPPI-FP, or they can bypass the HIPPI-FP layer and access the HIPPI protocol directly at the HIPPI-PH layer. Refer to the *IRIS HIPPI API Programmer's Guide* for complete details.

The rest of this section describes the IRIS HIPPI-LE upper-layer protocol implementation.

Handling of HIPPI Protocol for HIPPI-LE

This section describes how the IRIS HIPPI driver and the HIPPI-LE module handle HIPPI I-fields, HIPPI-FP headers, and 802.2 encapsulation items. There are separate sections for transmission and reception.

On Transmission

The HIPPI-LE module obtains the I-field for each destination from a lookup table that is initialized at startup time from the */usr/etc/hippi.imap* file. It obtains this I-field value before programming the IRIS HIPPI driver to make a connection request. The *hippi.imap* file maps IP addresses (or host names) to 32-bit values that are used as I-fields. The software uses each I-field value exactly as read from the file, as summarized in Table 1-8. It is the

responsibility of the system administrator to ensure that the values in the lookup table are appropriate for the site's configuration.

Table 1-8 I-field Recommended for Use With IRIS HIPPI-LE

Field	Value Recommended for HIPPI-LE	Comments
L	0	0=HIPPI-SC compliant; 1=local format for I-field. A site may use any value.
VU	0	A site may use any value.
W	0	0=32 data bus. No other setting is supported by the IRIS HIPPI board.
D	0	0=Least significant bits contain address for next hop; 1=address is placed in most significant bits. A site may use any setting.
PS	any setting	A site may use any of the addressing formats.
C	1	1= camp-on; 0=do not camp-on. This setting makes the HIPPI network more efficient; however, a site may use any setting.
Routing Control	any setting	A site may use any settings.

Once the connection has been opened, the HIPPI-LE module creates a HIPPI packet in the format illustrated in Figure 1-21. This packet conforms with the HIPPI-FP standard and the RFC 1374 guidelines.

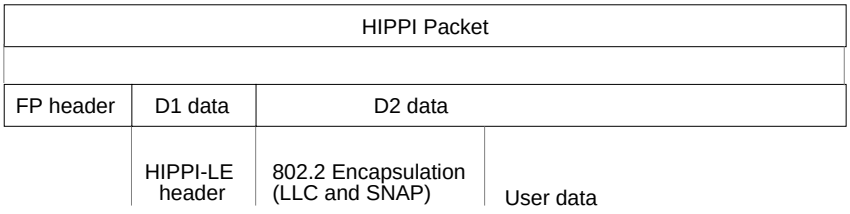


Figure 1-21 HIPPI Packet Created by IRIS HIPPI-LE

The IRIS HIPPI-LE module creates the HIPPI-FP header with the values summarized in Table 1-9.

Table 1-9 FP Header Created by IRIS HIPPI-LE ULP

Field	Value Used	Comments
ULP-id	4 = HIPPI-LE	As defined by HIPPI-FP.
P bit	1 = D1 area is included in this FP header	D1 area contains the HIPPI-LE header as defined by HIPPI-LE.
B bit	0 = D2 data is included in first burst	As specified by RFC 1374.
D1 Size	3 = three 64-bit words (that is, 24 bytes)	As defined by HIPPI-LE.
D2 Offset	0	
D2 Size	Up to 64 kilobytes.	Maximum IP packet as defined by the Internet Protocol.

The IRIS HIPPI-LE module creates the HIPPI-LE header with the values summarized in Table 1-10. The HIPPI-LE header becomes the D1 data set for the HIPPI packet.

Table 1-10 D1 Data (HIPPI-LE Header) Created by IRIS HIPPI-LE

Field	Size	Value Used	Comments
FC	3 bits	0	As defined by HIPPI-LE and restated in RFC 1374.
Double Wide	1 bit	0 = 32 bit data bus	
Message Type	4 bits	0 = data	
Destination Switch Address	24 bits	0	
Destination Address Type	4 bits	0	
Source Address Type	4 bits	0	
Source Switch Address	24 bits	0	

Table 1-10 (continued) D1 Data (HIPPI-LE Header) Created by IRIS HIPPI-LE

Field	Size	Value Used	Comments
Reserved	16 bits	0	
Destination IEEE Address	48 bits	0	
LE Locally Administered	16 bits	0	

The IRIS HIPPI-LE module creates the 802.2 headers (LLC and SNAP) with the values summarized in Table 1-11. This information occupies the initial bytes of the D2 data within the HIPPI packet.

Table 1-11 IEEE 802.2 Header (First Bytes of D2) Created by IRIS HIPPI-LE

Field	Size (bits)	Value Used	Comments
SSAP	8	170 decimal	As defined by IEEE 802.2 standard for Logical Link Control and restated in RFC 1374.
DSAP	8	170 decimal	Same as above.
CTL	8	3	Same as above.
Organization Code	24	0	Same as above.
EtherType	16	2048 decimal	Same as above.

On Reception

The IRIS HIPPI driver accepts all connection requests, and accepts all packets containing FP headers with known ULP-ids, thus supporting customer-developed, upper-layer applications. The I-field for the connection request is not interpreted. This process is summarized in Table 1-12.

Table 1-12 I-field Accepted by IRIS HIPPI Driver for HIPPI-LE ULP

Field	Recommended Values	Comments
L	0 = HIPPI-SC compliant	Content is ignored.
VU	any value	Content is ignored.
W	0 = 32 data bus	Content is ignored.
D	any value	Content is ignored.
PS	any value	Content is ignored.
C	any value	Content is ignored.
Routing Control	any value	Content is ignored.

Once a connection has been opened, the IRIS HIPPI driver places each incoming HIPPI packet on the input queue for the ULP-id indicated in the FP header. Incoming HIPPI packets must have the format illustrated in Figure 1-22. If the ULP-id is not known to the driver, the packet is dropped (that is, accepted then discarded). Packets with a ULP-id of 4 are enqueued for the HIPPI-LE module.

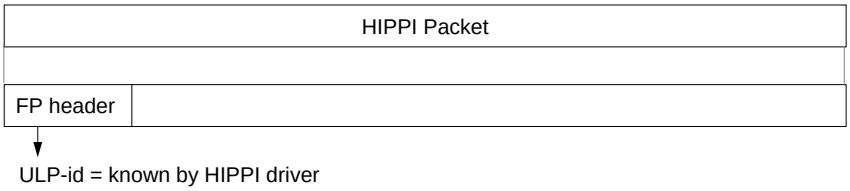


Figure 1-22 HIPPI Packets that IRIS HIPPI Driver Passes to HIPPI-LE

The HIPPI driver interprets only the FP header. All further processing of the HIPPI packet (including the various protocol headers) is done by the reader of the input queue (for example, HIPPI-LE).

On reception, the IRIS HIPPI driver handles HIPPI-FP headers as summarized in Table 1-13.

Table 1-13 FP Header Accepted by IRIS Driver for HIPPI-LE ULP

Field	Values Accepted Without Generating an Error	Comments
ULP-id	4 = HIPPI-LE	As defined by HIPPI-FP. For other ULP-ids, see the *IRIS HIPPI API Programmer's Guide* for details.
P bit	1	If set to 1, the driver interprets the D1 area as a HIPPI-LE header. If set to 0, the packet is discarded.
B bit	any value	For applications using the HIPPI-FP access method, the IRIS HIPPI driver passes the D1 data to the input queue reader as a separate item from the D2 data.
D1 Size	3 = three 64-bit words / 24 bytes	As defined by HIPPI-LE. If the value is different, the packet is discarded.
D2 Offset	any value	
D2 Size	up to 64 kilobytes	As defined by Internet Protocol. If size is greater than 64 kBytes, the packet is discarded.

The IRIS HIPPI-LE upper layer program handles received D1 data (the HIPPI-LE header) as summarized in Table 1-14.

Table 1-14 D1 Data Accepted by IRIS HIPPI-LE ULP

Field	Values Accepted Without Generating an Error	Comments
FC	0	As defined by HIPPI-LE. If the value is different, an error is generated.
DW	0 = 32 bit data bus	If set to 1, an error is generated.
MT	0 = data	If not 0 (data), an error is generated.
Dest_Sw_Addr	any value	
Dest_Addr_Type	any value	
Src_Addr_Type	any value	
Src_Sw_Addr	any value	
Dst_IEEE_Addr	any value	
LE_Locally_Adm	any value	
Src_IEEE_Addr	any value	

The IRIS HIPPI-LE module also handles the 802.2 headers as summarized in Table 1-15.

Table 1-15 IEEE 802.2 Headers Accepted by HIPPI-LE ULP

Field	Size (bits)	HIPPI-LE Default	Comments
SSAP	8	170 decimal	As defined by the IEEE 802.2 standard and restated by RFC 1374. If the received value is different, an error is generated.
DSAP	8	170 decimal	Same as above.
CTL	8	3	Same as above.
Organization Code	24	0	Same as above.
EtherType	16	2048 decimal	Same as above.

Configuring IRIS HIPPI

This chapter provides instructions and information about configuring the IRIS HIPPI software. The configuration tasks are listed, then described in detail, in the sections of this chapter.

Within this chapter, there is a section that describes how the physical HIPPI boards (*hippi0*, *hippi1*, *hippi2*, and *hippi3*) are matched to HIPPI network interfaces (*hip0*, *hip1*, *hip2*, and *hip3*).

Overview of Configuration Steps

Before configuring the IRIS HIPPI software, you need to decide whether or not you want the driver to include support for the IP network stack. The configuration steps are slightly different depending on this decision.

IRIS HIPPI Without IP Support

The following steps configure the IRIS HIPPI driver for use as a non-IP network connection. Complete details for each step are provided in separate sections of this chapter.

1. Use *inst* to install the IRIS HIPPI software from the CD-ROM, as explained in the *IRIS HIPPI Release Notes*. The *inst* command is documented in the *IRIS Software Installation Guide* that came with the system.
2. Optional:
   Edit the */usr/var/sysgen/system/hippi.sm* file to EXCLUDE the IP interface.

Note: If you exclude IP support from the driver, and later you want to use IP, you must redo the configuration steps. If you configure the software with IP, you only need to do the additional configuration steps to add IP functionality later. When the driver is built to support IP, but IP is not configured, some error messages are displayed each time the system is started. ♦

3. The system is ready to have its IRIS HIPPI hardware installed. When restarted (after the hardware installation), the system asks you to authorize rebuilding of the operating system. Answer **yes**, to build an operating system that includes the IRIS HIPPI driver. Then, reboot the system to start using the new operating system.

   Note: If the hardware is already installed, rebuild the operating system as described in "Building a New Driver Into the Operating System" on page 50. ♦

IRIS HIPPI With IP Support

The following steps configure the IRIS HIPPI driver with support for IP networking. Complete details for each step are provided in separate sections of this chapter.

1. Use *inst* to install the IRIS HIPPI software from the CD-ROM, as explained in the *IRIS HIPPI Release Notes*. The *inst* command is documented in the *IRIS Software Installation Guide* that came with the system.
2. Edit the */usr/var/sysgen/master.d/if_hip* file to configure IRIS HIPPI driver parameters.
3. Verify that the */usr/var/sysgen/system/hippi.sm* file has an INCLUDE statement for the IP interface.
4. Edit the */usr/etc/hippi.imap* file to include all the endpoint destination I-fields.
5. Edit the IP configuration files (*/etc/hosts* and */etc/config/netif.options*) to include IP network connection names and addresses.
6. Enable IP (that is, write `ON` into the */etc/config/network* file).

7. The system is ready to have its IRIS HIPPI hardware installed. When restarted (after the hardware installation), the system asks you to authorize rebuilding of the operating system. Answer **yes**, to build an operating system that includes the IRIS HIPPI driver. Then, reboot the system to start using the new operating system.

   Note: If the hardware is already installed, rebuild the operating system as described in "Building a New Driver Into the Operating System" on page 50. ♦

Checking If IRIS HIPPI Software Has Been Installed

Use the command below to check if the IRIS HIPPI software has been installed or to verify the version.

```
% versions hippi
I HIPPI date IRIS HIPPI, version
```

Editing the hippi.sm File

The */usr/var/sysgen/system/hippi.sm* file tells the system's software which IRIS HIPPI modules to include when it builds the IRIS HIPPI driver into the operating system. One line in this file can be edited to build an IRIS HIPPI driver that does not support IP networking.

- Original line that builds IP support into the driver:

  ```
  INCLUDE:  if_hip
  ```

- Changed line that builds an IRIS HIPPI driver without IP support:

  ```
  EXCLUDE:  if_hip
  ```

If you exclude IP functionality, then decide later that you want IP to function over HIPPI, you must undo this edit, then, rebuild the operating system.

Note: When the driver is built with IP support, but the IP protocol stack is not enabled, each time the system is started some error messages are displayed; the HIPPI functionality is all right. ♦

Editing the if_hip File

The */usr/var/sysgen/master.d/if_hip* file configures the IRIS HIPPI driver and board. Driver and board configuration is optional, because all parameters have default settings. The settings in this file affect all IRIS HIPPI boards installed in the system.

The IRIS HIPPI driver and board have very few configurable parameters (for example, the size for the maximum transmission unit and onboard IP checksumming). The specific items vary from release to release, so they are explained fully within the file.

Editing the hippi.imap File

The */usr/etc/hippi.imap* file maps hostnames (or IP addresses) to HIPPI I-fields. Optionally, a 6-byte universal IEEE address (called ULA, MAC or Ethernet address) can also be mapped. The file can contain up to 2048 lines.

Each time the HIPPI-LE upper layer protocol (ULP) module is about to program the HIPPI subsystem's source channel to issue a connection request, it obtains the I-field for its destination from a lookup table that has been loaded into memory at startup time. The lookup table maps IP addresses or hostnames to HIPPI I-fields. This table is generated (at startup time) from the *hippi.imap* file and can be modified in real time with the *hipmap* command.

Each I-field is a 32-bit value. Each line (entry) in the file can have any of the formats illustrated below:

- *hostname* `0x`*XXXXXXXX*

  where *hostname* is the name of a system as listed in the */etc/hosts* file and `0x`*XXXXXXXX* is the 32-bit I-field, in hexadecimal notation. The following line is an example of this format:

  ```
  hippi-goofy  0x01000001  #source adddress format for port1
  ```

- *x.x.x.x* 0x*XXXXXXXX*

  where *x.x.x.x* is the IP address in dotted decimal notation, and 0x*XXXXXXXX* is the 32-bit I-field in hexadecimal notation. The following line is an example of this format:

  ```
  223.9.1.18  0x07001002   #logical adddress format
  ```

- *hostname* 0x*XXXXXXXX* *XX*:*XX*:*XX*:*XX*:*XX*:*XX*

  where *hostname* is the name of a system as listed in the */etc/hosts* file, 0x*XXXXXXXX* is the 32-bit I-field in hexadecimal format, and *XX*:*XX*:*XX*:*XX*:*XX*:*XX* is the 48-bit ULA in hexadecimal notation. The following line is an example of this format:

  ```
  hippi-goofy  0x01000001  05:A6:70:9B:FF:8E
  ```

- *x.x.x.x* 0x*XXXXXXXX* *XX*:*XX*:*XX*:*XX*:*XX*:*XX*

  where *x.x.x.x* is the IP address in dotted decimal notation, 0x*XXXXXXXX* is the I-field (32-bits in hexadecimal format), and *XX*:*XX*:*XX*:*XX*:*XX*:*XX* is the ULA (48-bits in hexadecimal notation). The following line is an example of this format:

  ```
  223.9.1.18  0x01000001  05:A6:70:9B:FF:8E
  ```

The IRIS HIPPI software does not check or verify these values. It is the system administrator's responsibility to ensure that each entry is both valid and correct. The I-field value must be the exact I-field for use in the connection request; for example, it must contain the desired settings for the camp-on bit and the source's address bits.

Figure 2-1 is a template that can be used for developing each 32-bit I-field. The template shows the values that SGI recommends for use by the IRIS HIPPI-LE upper layer protocol.

Note: If the IRIS HIPPI source is connected directly to another HIPPI endpoint (no switch is involved), the single I-field value for the destination host can be 0x00000000.

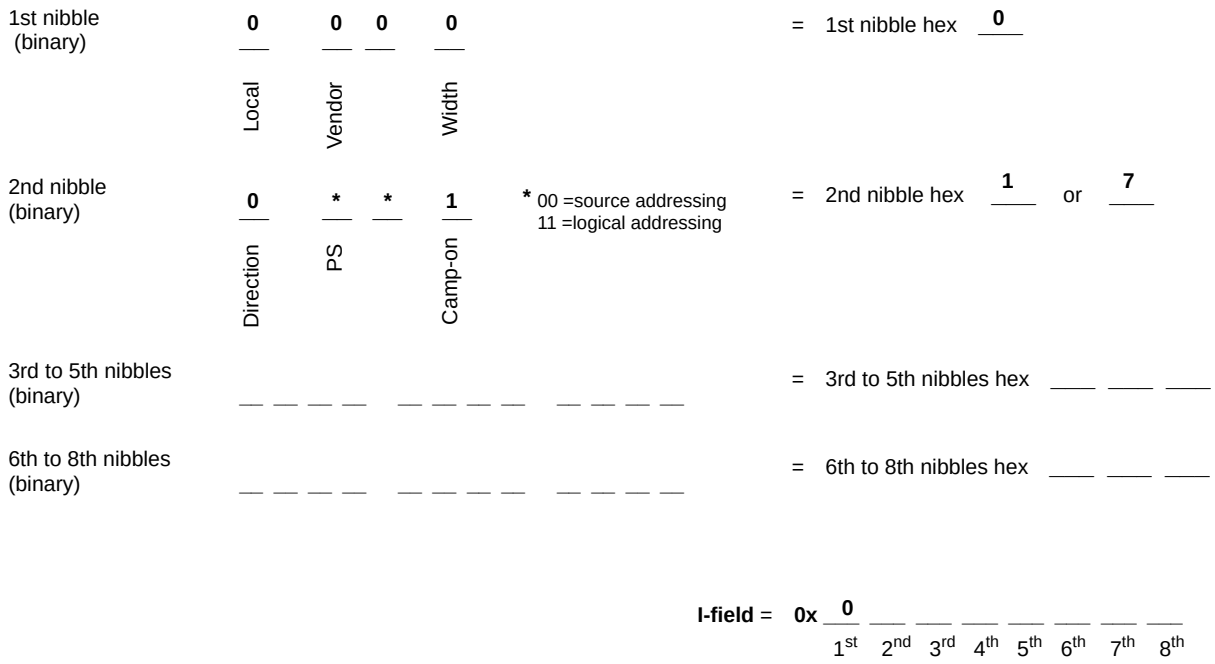


Figure 2-1 Template for Creating I-fields With Recommended Values

If IRIS HIPPI is already functioning, and you want to load a new *hippi.imap* file into memory, the following command can be invoked:

```
# /usr/etc/hipmap -f /usr/etc/hippi.imap
```

Editing the IP Configuration Files

To configure the IP networking interfaces, edit the */etc/hosts* and */etc/config/netif.options* files, as explained below.

Each time the IP software starts, it uses information from these two files to configure the IP interfaces.

Note: For additional details about enabling the IP networking software and configuring network interfaces, refer to the *IRIX Site Administrator's Guide*, which is available online through IRIS Insight™ or in hardcopy. ♦

The /etc/hosts File

The */etc/hosts* file maps hostnames to network-layer IP addresses. There must be one entry for each IRIS HIPPI board. The entries should be similar to the examples below, which illustrates four IRIS HIPPI interfaces for a system whose hostname is *goofy.*:

```
223.9.1.2     hippi-goofy.toons.com     hippi-goofy
223.9.2.4     hippi2-goofy.toons.com    hippi2-goofy
223.9.3.16    hippi3-goofy.toons.com    hippi3-goofy
223.9.4.32    hippi4-goofy.toons.com    hippi4-goofy
```

The /etc/config/netif.options File

The */etc/config/netif.options* file maps local hostnames (or IP addresses) to IRIS HIPPI boards (for example, *hip0*, *hip1*, etc.). There must be a two-line entry for each IRIS HIPPI board that services the IP network stack. The first entry (`if1addr` and `if1name`) defines the primary interface; in most situations, the primary interface should be Ethernet or FDDI. Each hostname or IP address in this file must also be in the */etc/hosts* file; the hostnames or IP addresses in the two files must be identical.

Systems that function as a client or server for *bootp* should configure Ethernet as their primary network interfaces. Any system that functions as a client or server for NFS, NIS, or other client / server program should configure the network interface over which the client / server functions occur as the primary network interface.

The example below illustrates a system with three IRIS HIPPI interfaces, an FDDI interface (that is, *ipg0*), and a primary Ethernet interface. If this system's hostname is *goofy*, these IRIS HIPPI entries interwork with the examples of */etc/hosts* file entries shown above.

```
if1name=et0
if1addr=$HOSTNAME
```

```
if2name=ipg0
if2addr=fddii-$HOSTNAME

if3name=hip0
if3addr=hippi-$HOSTNAME

if4name=hip1
if4addr=hippi2-$HOSTNAME

if5name=hip2
if5addr=hippi3-$HOSTNAME
```

Note: The use of the $HOSTNAME variable assumes that the system's hostname has been defined in the */etc/sys_id* file. ♦

Enabling IP Networking

To automatically enable the IP network stack each time the system is started, edit the */etc/config/network* file so that it contains the single word `ON` or `on`. If the file is missing, add the file.

Note: Enabling IP networking does not result in IP over HIPPI; it only enables the IP software to operate over whatever drivers are available to service it. (For example, the IP protocols may use Ethernet or FDDI drivers.) ♦

Building a New Driver Into the Operating System

For the IRIS HIPPI subsystem to be functional, the IRIX operating system (kernel) that is currently on the system must be built to include the IRIS HIPPI driver. When changes are made to the *hippi.sm* or *if_hip* files, or when new IRIS HIPPI software is installed, it is also necessary to rebuild the operating system to include the changes. This section describes how to rebuild the operating system. All the configuration steps listed in the "Overview of Configuration Steps" must be performed before the operating system is rebuilt.

When the IRIX operating system is rebuilt, it uses values from the */usr/var/sysgen/system/hippi.sm* file and the */usr/var/sysgen/master.d/if_hip* file to build various configurable parameters into the driver.

The three sets of instructions below each build a new operating system and start it running. It is not important which set of instructions are used.

Instruction Set 1

```
% su
Password: thepassword
# /etc/init.d/autoconfig
Automatically reconfigure the operating system (y or n)? y
<log on>
% su
Password: thepassword
# /etc/reboot
.....<various messages are displayed on console>...
configuring hip0 as hostname
configuring hip1 as hostname
```

Instruction Set 2

```
% su
Password: thepassword
# /etc/shutdown
```

When the system shuts down, restart it. When this question is displayed, answer with **yes** or **y**.

```
Automatically reconfigure the operating system (y or n)? y
<log on>
% su
Password: thepassword
# /etc/shutdown
```

Instruction Set 3

Use the same sequence as Set 2; however, instead of the */etc/shutdown* command, use any of the following:

- */etc/init 0*
- */etc/halt*
- */etc/reboot*

How HIPPI Boards Are Assigned to Interfaces

This section describes the manner in which IRIX assigns an IP network interface (for example, *hip0* and IP address 223.9.1.2) to a particular HIPPI board.

With each restart (for example, after a *reboot*, *shutdown*, *halt*, or *init* command, or a power off), the startup routine probes for hardware installed in the mezzanine I/O adapter slots, and makes a list of all the boards located. The slots are probed in the following order:

- main IO4 board: I/O adapter slot 5, then 6
- second IO4 board (if present): I/O adapter slot 2 (only when the FMezz board is long), slot 5, slot 3 (only when the FMezz board is long), slot 6
- third IO4 board (if present): I/O adapter slot 2 (only when the FMezz board is long), slot 5, slot 3 (only when the FMezz board is long), slot 6
- fourth IO4 board (if present): I/O adapter slot 2 (only when the FMezz board is long), slot 5, slot 3 (only when the FMezz board is long), slot 6

The list and order of IRIS HIPPI boards that were located by this process can be displayed with the */sbin/hinv* command, as shown below. The text `hippi#` indicates the order: `hippi0` is the first board located and `hippi1` is the second. In this example, the startup routine located two IRIS HIPPI boards attached to FMezz boards on two different IO4 boards.

```
%/sbin/hinv
...
HIPPI adapter: hippi0, slot 5 adap 6, firmware version ####
HIPPI adapter: hippi1, slot 3 adap 5, firmware version ####
```

As the startup routine begins to initialize HIPPI network interfaces, it does the following:

- Searches the *netif.options* file for IP over HIPPI interface names (for example, *hip0*, *hip1*, *hip2*, *hip3*). It orders the interfaces by the number in the associated string `if#name`. For example, the entries `if2name=hip0`, `if3name=hip1` and `if4name=hip2` would be ordered *hip0* (first), *hip1* (second), and *hip2* (third).
- For each HIPPI network interface name, the startup routine tries to assign a HIPPI board. The first HIPPI interface is assigned to the first board found; the second interface is assigned to the second board; and

so on, until the routine runs out of interfaces or boards. For example, using the ordering described above, board *hippi0* is assigned interface *hip0* and board *hippi1* is assigned *hip1*.

Note: If an installed board is not located due to a loose connection or malfunction, or if hardware is installed or removed, the assignment of HIPPI network interfaces to boards may change. For example, *hip0* (from the example above) could be assigned, at a later reboot of the machine, to the second HIPPI board (*hippi1*) in the system instead of the first, if the first board became loose or dysfunctional. ♦

Maintaining and Monitoring IRIS HIPPI

This chapter describes how to maintain, monitor, and verify the IRIS HIPPI subsystem.

Commands Available for IRIS HIPPI

IRIS HIPPI can be monitored and maintained with the commands summarized in Table 3-1.

Table 3-1 Utilities for Monitoring and Maintaining IRIS HIPPI

Command	Purpose
/usr/etc/hipmap	Adds and deletes entries from the lookup table (in memory) that maps HIPPI I-fields to IP addresses.
/usr/etc/hipcntl	Provides a variety of control and status functions for the IRIS HIPPI subsystem.
/usr/etc/hiptest	Verifies IRIS HIPPI subsystem through the character device interface, without going through the IP network interface.
/usr/etc/ping	Verifies IP network interfaces. Can be used to verify that a *hip#* network interface is functioning.
/usr/etc/ifconfig	All the normal IP configuration options work with IRIS HIPPI IP network interfaces (that is, *hip#*), except broadcast, arp, and the specification of a destination IP address for setting up a point-to-point connection.
/usr/etc/netstat	All the normal network status information is available for IP interfaces to IRIS HIPPI. Non-IP interfaces are not displayed; however, if the IRIS HIPPI driver has been built with IP support, a disabled *hip0* interface with no IP address is shown.

Step-by-Step Instructions for Common Procedures

This section describes some of procedures commonly used to monitor and maintain the IRIS HIPPI subsystem. All of the IRIS HIPPI utilities (*hipmap*, *hipcntl*, and *hiptest*) require the user to have superuser (root) privileges.

Disable or Enable IRIS HIPPI Board

To shut down or disable the IRIS HIPPI board, use the command below. This resets the board; all data (incoming or outgoing) that is on the board is lost.

```
# hipcntl [hippi#] shutdown
```

To start or enable the IRIS HIPPI board, use the command below. This command verifies that the versions of the firmware on the board and the driver in the operating system match. If they do not match, the driver loads a compatible version of firmware onto the board.

```
# hipcntl [hippi#] startup
```

Configure Board to Reject or Accept Connection Requests

To configure the IRIS HIPPI subsystem so that the transmit channel does not generate any connection **REQUEST** signals and so that the receive channel does not generate any **CONNECT** (accept) signals, use the command below:

```
# hipcntl [hippi#] reject
```

To configure the IRIS HIPPI subsystem so that both the transmit and receive channels open connections, use the command below. This command results in the transmit channel generating connection **REQUEST**s when host applications send data, and in the receive channel generating **CONNECT** signals in response to connection **REQUEST**s.

```
# hipcntl [hippi#] accept
```

Check Status

To display status information for an IRIS HIPPI board, use the command below. Each counted item is initialized to zero upon reset of the board. The

counters roll over to zero upon reaching 2^{32}. The displayed information is described in Table 3-2.

```
# hipcntl [hippi#] status
```

Table 3-2 IRIS HIPPI Status Information

Status Item	Description
FLAGS:	
DSIC	SRC sees the **INTERCONNECT** input signal.
SDIC	DST sees the **INTERCONNECT** input signal.
ACCEPTING	DST is accepting connections. When this flag is not listed, the DST is rejecting connections.
DST.PKT	DST sees that the **PACKET** input signal is asserted
DST.REQ	DST sees that the **REQUEST** input signal is asserted
SRC.REQ	SRC channel's **REQUEST** output signal is asserted
SRC.CON	SRC sees that the **CONNECT** input signal is high
SRC connections:	Count of total connection **REQUEST** signals issued by source.
SRC packets:	Count of total packets sent by source.
SRC rejects:	Count of connection attempts that were rejected by the destination.
SRC seq errors (dm):	Count of data state machine's sequence errors.
SRC seq errors (cd):	Count of connection state machine's sequence errors for which the destination is believed to be at fault.
SRC seq errors (cs):	Count of connection state machine's sequence errors for which the source is believed to be at fault.
SRC dsic lost:	Count of connections dropped due to lost **DSIC** signal.
SRC time outs:	Count of connection attempts that timed out so that the source withdrew the request.

Table 3-2 (continued) IRIS HIPPI Status Information

Status Item	Description
SRC connects lost:	Count of connections that were dropped by the destination.
SRC parity errs:	Count of source parity errors.
DST connections:	Count of connections accepted.
DST packets:	Count of total packets received
DST rcv on bad ulp:	Count of packets discarded due to unknown ULP-id.
DST hippi-le drop:	Count of HIPPI-LE packets discarded.
DST llrc:	Count of connections dropped due to LLRC errors.
DST parity:	Count of connections dropped due to parity errors.
DST sequence err:	Count of connections dropped due to sequence errors.
DST sync err:	Count of synchronization errors.
DST illegal burst:	Count of packets with illegal burst sizes.
DST sdic lost:	Count of connections dropped due to lost **SDIC** signal.
DST null connections	Count of connections with zero packets.

Disable or Enable an IP Interface

To enable / disable the IP network interface to the IRIS HIPPI board, use the standard */usr/etc/ifconfig* command, as shown below.

```
# ifconfig [hip#] down
# ifconfig [hip#] up
```

Configure IP Network Interface Over IRIS HIPPI

Dynamic configuration of the IP network interfaces is done with the */usr/etc/ifconfig* command, which is explained in detail in the man page. The command lines listed below are available for use with IRIS HIPPI:

```
# ifconfig  [hip#] IPaddr
# ifconfig  [hip#] up
# ifconfig  [hip#] down
# ifconfig  [hip#] netmask ########
# ifconfig  [hip#] metric #
```

Note: Some of the standard *ifconfig* arguments are not supported for IRIS HIPPI. ♦

Change the Lookup Table That Maps IP Hosts to I-fields

The */usr/etc/hipmap* command makes changes to the lookup table that is currently in memory.

To add an entry to the look up table, use this command line:

```
# hipmap hostname  I-field_value  [ULA_value]
```

To delete one entry from the lookup table, use this command line:

```
# hipmap -d hostname
```

To delete all the entries from the lookup table, use this command line:

```
# hipmap -D
```

To concatenate entries from a file onto the lookup table, use this command line:

```
# hipmap -f filename
```

To clear the lookup table, then add entries from a file, use this command line:

```
# hipmap -D -f filename
```

Display the Lookup Table That Is Currently in Memory

Use the command line below to display the table of IP addresses mapped to I-fields that is currently loaded into memory:

hipmap -a

Set Timeout for Source Channel Connections

To dynamically change the timeout value used by the IRIS HIPPI source channel, use the command line below. The source timeout is the amount of time that the source channel waits for a **CONNECT** or **READY** signal from the destination before it aborts the request.

In this command line, the *timeout* is expressed in milliseconds. The granularity for this timeout is a quarter of a second (that is, 250 milliseconds). A *timeout* value that are not divisible by 250 is rounded up to the next quarter-second.

hipcntl [**hippi***#*] **stimeo** *timeout_in_milliseconds*

Verifying the HIPPI Subsystem

The most reliable method for verifying an IRIS HIPPI subsystem is to install a loopback link between the destination and source on the same system, then run the *hiptest* verification test described under the heading “Verify the Interface to HIPPI-FP” in this section. After the HIPPI subsystem has been verified, further upper-layer verification tests can be run (for example, the test described under the heading “Verify an IP Interface” in this section).

Install a Loopback Link

To install a loopback link, use any of the procedures illustrated below:

- Use any standard HIPPI cable to connect the IRIS HIPPI DST and SRC ports on the system’s I/O panel plate to each other, as illustrated in Figure 3-1.

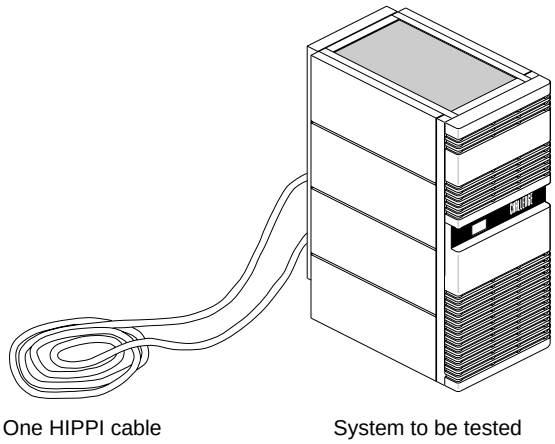


Figure 3-1 Installing a Loopback Link Using a HIPPI Cable

- Use a special loopback (female-to-female) cable to connect the other end of the HIPPI destination and source cables as illustrated in Figure 3-2.

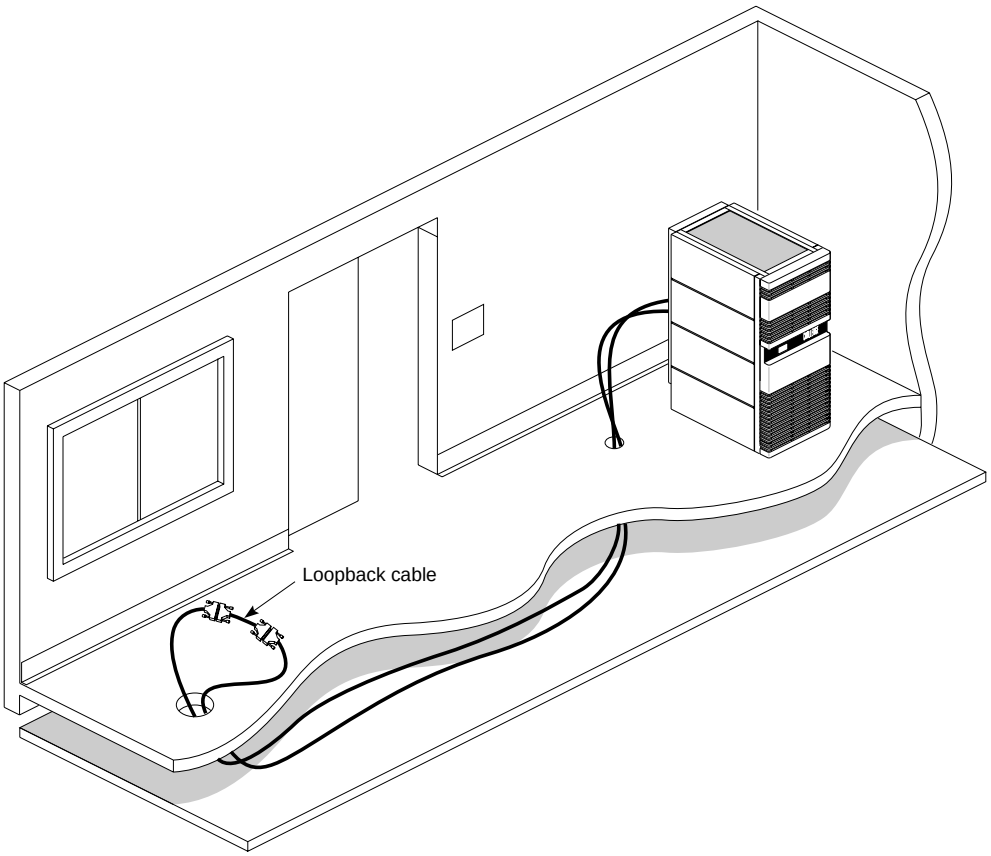


Figure 3-2 Installing a Loopback Link Using a Loopback Cable

- At a switch, connect the cables coming from the IRIS HIPPI board to the same port or to two different ports. For example, the cable from the IRIS HIPPI destination could be attached to the OUT of port 1, while the cable from the IRIS HIPPI source could be attached to the IN of port 1 (or of port 2).

  Note: This loopback configuration requires you to use a valid I-field during the test and assumes that the switch is functional. ♦

Verify the Interface to HIPPI-FP

To verify the IRIS HIPPI-FP subsystem (without going through the IP stack), use the */usr/etc/hiptest* command. The test only works for an IRIS HIPPI board that has a loopback link installed between its source and destination channels. The command is available only to superuser (root).

This test sends randomly-sized HIPPI-FP packets that contain randomly generated data as the D2 data set. The test then reads the received packets and verifies that the received data matches the data that was sent. The following items from the received packet are compared to those items from the transmitted packet: length of the header (FP header and D1 data area), length of the D2 area, and data integrity (word-by-word comparison) for the D2 data set.

The command creates HIPPI-FP packets with the following non-configurable characteristics:

FP header	8 bytes of FP header where all fields contain valid values for the packet. The ULP-id used is 0x89 (hexadecimal).
D1 area size	24 bytes.
D1 data set	Zero.
D2 area size	Randomly generated size, up to the constraining bytecount specified by *maxsize*. The first words of D2 area are included in the first burst of the packet.
D2 data set	Randomly generated data.

The command allows you to specify the following packet characteristics:

-I The I-field value (in hexadecimal format) to use for the connection request. If not specified on the command line, the command uses 0x00000001.

```
#/usr/etc/hiptest -I 0x07001002
```

-D The IRIS HIPPI board to test: */dev/hippi0*, */dev/hippi1*, etc.. If not specified on the command line, the command uses */dev/hippi0*.

```
#/usr/etc/hiptest -D /dev/hippi3
```

maxsize The maximum bytesize (in decimal format) for the packets. The value specified is rounded down to a number that is divisible by 8. The minimum is 32 (8 bytes of FP header and 24 bytes of D1 data). The maximum is 2 megabytes (that is, 2097152 bytes). When *maxsize* is specified, *#packets* must also be specified. If *maxsize* is not specified on the command line, the command uses 2 megabytes.

For an example, see the command line shown for #packets.

#packets The number of packets (in decimal format) to send before dropping the connection and ending the test. The minimum is 1. There is no maximum. When *#packets* is specified, *maxsize* must also be specified.If *#packets* is not specified on the command line, the command uses 100.

#**/usr/etc/hiptest 1500 10**

The command line usage for *hiptest* is summarized below. After the command is invoked, each successfully sent packet is indicated with a dot. To terminate the test at any point, press the <**Ctrl**> and <**C**> keys simultaneously.

`hiptest [-I 0x<`*Ifieldvalue*`>] [-D /dev/hippi[0-3]] [`*maxsize* `[`*#pckts*`] ]`

Examples:

To run the test using the default settings, use the commands below:

```
%cd /usr/etc
%su
Password: thepassword
#hiptest
/dev/hippi0 sending 100 packets of size [0..2097152] to I-field 0x00000001
..................... <up to 100 dots>
```

To send one minimum-sized packet, use the command line below:

```
#hiptest 32 1
/dev/hippi0 sending 1 packets of size [0..32] to I-field 0x00000001
.
```

To send 25 packets of up to 2 megabytes on a loopback test, use the command line below:

```
#hiptest 2097152 25
/dev/hippi0 sending 25 packets of size [0..2097152] to I-field 0x00000001
.........................
```

To run the test when there is a switch located along the loopback link, specify a valid I-field, as illustrated in the command below. You must replace the I-field shown in the example with one that is appropriate; however, it is recommended that the I-field (as shown in this example) have the camp-on bit set to one, the PS bits set to zero for source addressing, and the rightmost bits set to the port identification number for the IRIS HIPPI destination.

```
#hiptest -I 0x01000002
/dev/hippi0 sending 100 packets of size [0..2097152] to I-field 0x01000002
..................... <up to 100 dots>
#
```

To test four different IRIS HIPPI boards, invoke the command in four separate shell windows or execute it four times in the background, as in the example below:

```
#hiptest -D/dev/hippi0 &
#hiptest -D/dev/hippi1 &
#hiptest -D/dev/hippi2 &
#hiptest -D/dev/hippi3 &
```

If the *hiptest* utility fails with an error message, locate the error message in the section “Alphabetical Error Message Listing” in Chapter 4 and follow the instructions.

Verify an IP Interface

To verify that each HIPPI IP network interface is functional, follow the instructions in this section. This test requires the local system to be attached either to another HIPPI endpoint that supports IP or to a HIPPI LAN that has functioning IP hosts. This test assumes that the HIPPI subsystem has passed the *hiptest* verification, as described under the heading “Verify the Interface to HIPPI-FP” in this section.

To accomplish this verification, use */usr/etc/ping -r* (lower case -r, not -R) to make this station communicate with another HIPPI IP station over the HIPPI subsystem.

1. Obtain the IP network addresses for all the local area networks attached to IRIS HIPPI boards on this system. This information can be displayed with the command shown below. The network address is listed in the column labelled `Network`, as illustrated in Figure 3-3.

   `%`**`/usr/etc/netstat -ina`**

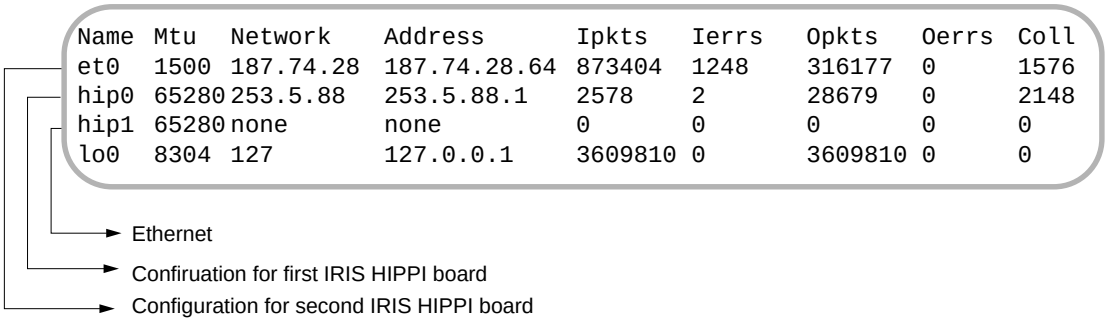


Figure 3-3 The */usr/etc/netstat -ina* Display

2. Obtain the name or IP address of at least one station on each of these networks. Two methods for obtaining station names are described below. The *hip#_networkaddress* variables used are the values in the `Network` column of the *netstat* display (illustrated in Figure 3-3).

   - For sites with NIS, use the commands below to create a file for each network connection that contains the names and addresses of stations located on that local area network:

     `%`**`ypcat hosts | grep`** ***`hip0_networkaddress`*** **`> hip0.s`**
     `%`**`ypcat hosts | grep`** ***`hip1_networkaddress`*** **`> hip1.s`**
     `<do this for each HIPPI IP network address>`

     Example:
     `%`**`ypcat hosts | grep 253.5.28 > hip0.s`**

   - For distributed (non-NIS) sites, use these commands to create a file for each network connection that contains the names and addresses of stations located on that local area network:

```
%grep hip0_networkaddress /etc/hosts > hip0.s
%grep hip1_networkaddress /etc/hosts > hip1.s
<do this for each HIPPI IP network address>
```

Example:

```
%grep 253.5.88 /etc/hosts > hip1.s
```

3. Communicate with one station on the LAN attached to the *hip0* connection. For the variable *hip0_station*, you can use any of the names or IP addresses from the *hip0.s* file, except the station's own.

```
%ping -r hip0_station
PING stationname (IPaddress): 56 data bytes
64 bytes from . . . time=x ms
. . .
<Ctrl><c>
----stationname PING Statistics----
# packets trans,# pckts rcvd, x% packet loss
```

4. If *netstat* lists other IRIS HIPPI (*hip#*) network interfaces, communicate with one station on each of those networks. For the variable *hip#_station*, you can use any of the names from the *hip#.s* file, except the station's own.

```
%ping -r hip#_station
PING stationname (IPaddress): 56 data bytes
64 bytes from . . . time=x ms
. . .
<Ctrl><c>
----stationname PING Statistics----
# packets trans, # pckts rcvd, x% packet loss
```

5. If one *ping* on each network succeeds, you have completed the verification procedure. All the network connections are functioning. Use the commands below to remove the files with the lists of stations:

```
%rm hip0.s
%rm hip1.s
<do this command line for each .s file created>
```

If the *ping* on a network fails, follow the instructions in the section "Troubleshoot an IP Interface."

Troubleshooting

Troubleshoot the Interface to HIPPI-FP

If the *hiptest* utility fails with an error message, locate the error message in the section "Alphabetical Error Message Listing" in Chapter 4 and follow the instructions.

Troubleshoot an IP Interface

If the *ping* verification tests fail for all the HIPPI network connections, your system probably has been configured incorrectly. Verify the configure with the commands below.

1. Verify that IP networking is enabled with the following command line:

   ```
   %/sbin/chkconfig | grep network
   network    on
   ```

2. Verify that the */usr/etc/hippi.imap* file has entries for the local system's network connection names (or IP addresses) and for the remote system interface names or IP addresses that failed.

3. For each HIPPI I/O panel, verify that the HIPPI cables connect to a LAN with a network address that matches the HIPPI network interface address for that board. You can display the local network address with the */usr/etc/netstat -ina* command. The address shown for network interface *hip0* belongs to board *hippi0*; the address for *hip1* belongs to board *hippi1*. To verify the network address being used on the cable at the I/O panel, you have to perform the equivalent action on other systems on that LAN.

   If the network address is correct, continue to the next step. If the address is wrong, either change the local system's address to match or connect the correct the cables to the I/O panel.

4. Refer to the *IRIX Advanced Site and Server Administration Guide* for information about configuring and troubleshooting IP.

If one of the *ping* verification tests succeeds, but one or more fails, the IP stack is functioning, but the specific interface has a problem. Follow the

instructions in this section for each problematic network connection to resolve the problem.

1. Verify that the */usr/etc/hippi.imap* file has an entry for the problematic local interface and for the remote hostnames or IP addresses that failed.
2. If the system is connected to a switch, verify that the switch is operational.
3. Verify that the other system (endpoint host) is operational. If it is, continue to the next step.

   Or, as an alternate, select two different stations on the LAN for which *ping* failed. Try to *ping -r* each station using the numerical address (instead of the name). If a *ping* works, the network connection is functional. If the *ping*s fail, proceed to the next step.
4. Check that the HIPPI cables between the I/O panel and the other system (switch or endpoint) are tightly connected at both ends.
5. Again try to *ping* a station on that network. If the *ping* still fails, proceed to the next step.
6. Verify that the network portion (leftmost digits) of the addresses you are attempting to *ping* match the network addresses displayed by the */usr/etc/netstat -in* command.

   If the network addresses do not match, repeat the steps in "Verify an IP Interface" to obtain and test a new selection of stations. Take extra precautions to ensure that you use the correct network addresses. If the network addresses match, continue.
7. Verify that the HIPPI cables at the I/O panel attach (at their other ends) to a system using the same network address. For example, verify that the network address for *hip0* matches the network address for the system at the other end of the cables attached to the *hippi0* board. (If the other system is a switch, verify the network addresses at the endpoints attached to that switch.)

Chapter 4

IRIS HIPPI Error Messages

This chapter lists the error messages that the IRIS HIPPI utilities can display.

Overview of the Error Message Listing

This section is a reference section containing an alphabetical list of all the error messages that can be displayed by IRIS HIPPI software.

With each error message is a discussion of the problems the message may indicate. The list contains only messages that indicate an error or problem; it does not contain informational messages that occur during normal operation.

Messages are alphabetized according to the following rules:

- Each message is alphabetized by the numerals (0–9) and letters (a–z) of the message's text. Numerals precede letters. (Figure 4-1 illustrates the text of an error message.)
- Nonletters (for example, - or %) and blank spaces are shown in the text of the message, but are ignored in alphabetization. For example, the message `hip_open` appears between `hipnet` and `hippi`.
- When an error message includes an item that the software specifies differently (fills in) for each instance of the message, this item is displayed in italic font and labeled with a generic name (for example, *`filename`*). The generic names are skipped for alphabetization purposes. For example, the error message `goofy not responding` is located under *`hostname`* `not responding` among the "n" listings. Common generic names used in this listing include *`hostname`*, *`interfacename`*, *`packet#`*, *`version#`*, *`userentry`*, *`reason`*, *`digit`*, *`filename`*, and *`hexnumeral`*.

Note: If you cannot find an error message in the listing, identify potential fill-in words, then look up the message without those words.♦

- Capitalization is not considered in alphabetization.
- The creator of each message is listed, in angled brackets, below the text of the message: (<*creator*>).

IRIS HIPPI error messages are written into the file */usr/var/adm/SYSLOG* or displayed at the terminal; some messages appear in both places. Within the *SYSLOG* file, each message is preceded by the date, time, host name, name of the process that created the message, and process ID number, as illustrated in Figure 4-1. Only the text of the error message (as illustrated in Figure 4-1) is included in the alphabetic list that follows.

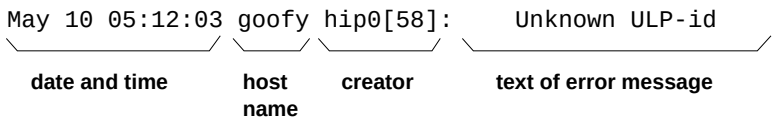


Figure 4-1 Error Message Format in */usr/var/adm/SYSLOG* File

Note: The list of error messages in this chapter covers only those unique to IRIS HIPPI. Standard system error messages, even when caused by the IRIS HIPPI code, are not covered. ♦

Alphabetical Error Message Listing

This section lists the error messages displayed on the console by the IRIS HIPPI utilities and driver. Many of these messages are also written to the *SYSLOG* file.

`#: bad HIPPI unit number`

The entry used on the command line with *hipcntl* to identify the IRIS HIPPI board (*hippi#*) contains an invalid unit number. Valid command line entries are: **`hippi0`**, **`hippi1`**, **`hippi2`**, and so on, for as many of the installed boards as were located during the last restart.

`harpioctl: unknown cmd:` *command*

While attempting to resolve an address, the driver encountered the indicated unknown *ioctl* command. This indicates that an upper layer application (for example, the *ifconfig* utility) is passing the unknown ARP command to the driver. Valid ARP commands include SIOCSHARP, SIOCGHARP, SIOCDHARP, and SIOCGHARPTBL.

`hip#: ifhip_output: Unsupported addr. family:0x`*hexnumeral*

While processing a packet for transmission, the driver found that the specified destination address does not belong to a supported address family. The packet's address family is indicated by *hexnumeral*. The packet was not sent.

`hipcntl: couldn't get HIPPI statistics:` *reason*

The *ioctl* call for the command HIPIOC_GET_STATS failed. The *reason* is any of those described by the *intro*(2) man page.

`hipcntl: couldn't open HIPPI device /dev/hippi#`

The *open* system call for the indicated IRIS HIPPI file device failed. This may indicate that the IRIS HIPPI software has not been installed properly or that it has been partially removed. Use *inst* to reinstall the IRIS HIPPI software from the CD-ROM or distribution directory.

`hipcntl: couldn't set HIPPI accept flag:` *reason*

The *ioctl* call for the command HIPIOC_ACCEPT_FLAG failed to changed the destination channel's accept / reject setting. The *reason* is any of those described by the *intro*(2) man page.

`hipcntl: couldn't set src timeout:` *reason*

The *ioctl* call for the command HIPIOC_STIMEO failed to set a new timeout for the source channel. The *reason* is any of those described by the *intro*(2) man page.

```
hipcntl: Double Warning: may be firmware driver mismatch if
you force download.
```

As *hipcntl* prepared to download firmware (startup) and discovered that the driver is a debug version, it also discovered that the driver and the firmware that is about to be loaded do not match. This probably indicates a mismatch between the *hipcntl* utility and the driver. Erase this *hipcntl* utility and install a copy that matches the driver.

```
hipcntl: Error: board is already up.
```

When *hipcntl* attempted to start the board, it found the board already started.

```
hipcntl: Error: couldn't get version numbers.
Possible hipcntl/kernel-driver mismatch.
Please autoconfig your system and reboot.
```

As *hipcntl* prepared to download firmware (startup), it discovered that it either could not retrieve a version number for the current driver or for the firmware currently loaded into the PROM on the IRIS HIPPI board. This indicates that the IRIS HIPPI driver has not been built into the operating system.

`hipcntl: Error: mismatch between kernel driver and hipcntl. Cannot startup adapter.`
`You probably need to autoconfig and reboot your system and/or remove any old copies of hipcntl(1m) on your system.`

As *hipcntl* prepared to download firmware (startup), it discovered that the driver and the firmware that is about to be loaded do not match. This probably indicates a mismatch between the *hipcntl* utility and the driver. Erase this *hipcntl* utility, reinstall the IRIS HIPPI software, and build a new operating system.

`hipcntl: HIPPI Board is down`

The *ioctl* call for the command HIPIOC_GET_STATS failed because the IRIS HIPPI board is not available (that is,it is shutdown or not responding). To remedy this problem,use command **`hipcntl startup`**. If it does not solve the problem, you may need to have the IRIS HIPPI board checked.

`hipcntl: problem programming flash:` *reason*

The *ioctl* call for the command HIPPI_PGM_FLASH failed to download new firmware into the IRIS HIPPI board's PROM. The *reason* is any of those described by the *intro*(2) man page. The new firmware has not been loaded into the IRIS HIPPI board's PROM. This message should be preceeded by other error messages indicating problems with the board's FLASH EEPROM. Contact the Silicon Graphics Technical Assistance Center.

`hipcntl: trouble bringing up HIPPI:` *reason*

The *ioctl* call for the command HIPPI_SETONOFF failed to start the IRIS HIPPI board. The *reason* is any of those described by the *intro*(2) man page. This message probably indicates that the board is dysfunctional. Invoke *hipcntl* to shut down the board; then, try to start the board. If this does not succeed, contact the Silicon Graphics Technical Assistance Center.

`hipcntl: trouble shutting down HIPPI:` *reason*

The *ioctl* call for the command HIPPI_SETONOFF failed to shutdown the IRIS HIPPI board. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: couldn't bind socket: reason`

The utility was unable to bind to the raw socket. This indicates a problem with the operating system, not with the IRIS HIPPI software or hardware. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: couldn't get raw socket:` *reason*

The utility was unable to obtain a raw socket. This indicates a problem with the operating system, not with the IRIS HIPPI software or hardware. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: couldn't open input file:` *reason*

The file supplied on the command line (for example, */usr/etc/hippi.imap*) could not be opened. This can indicate that the file does not exist, or that the permissions are not set correctly. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: couldn't SIOCDHARP:` *reason*

The SIOCDHARP command within an *ioctl* system call failed. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: couldn't SIOCSHARP:` *reason*

The SIOCSHARP command within an *ioctl* system call failed. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: malformed address name:` *IPaddress* `or` *hostname*

The *hostname* or *IPaddress* indicated is not valid. The *hostname* or *IPaddress* is a user entry from a file (for example, the */usr/etc/hippi.imap* file) or a command line entry.

`hipmap: malformed I-field in line:` *line#*

The second entry on the indicated line does not conform to a valid I-field. To be valid, the I-field entry must be a 32-bit value in hexadecimal format (for example, 0x00100003).

`hipmap: malformed line:` *line#*

The indicated line in the file being read (for example, */usr/etc/hippi.imap*) is not correctly formatted.

`hipmap: malformed switch address.`

The I-field entered on the command line is not valid. To be valid, the I-field entry must be a 32-bit value in hexadecimal format (for example, 0x0100000C or 0100000C).

`hipmap: malformed ULA in line:` *line#*

On the indicated line, there is an optional third entry that does not conform to a valid IEEE universal LAN MAC address (ULA) address. To be valid, the ULA entry must be a 48-bit value in hexadecimal format (for example, 0x7A385CF9028D).

`hipmap: trouble flushing harp entry:` *reason*

The SIOCDHARP command failed within an *ioctl* system call.The *reason* is any of those described by the *intro*(2) man page.

`hipmap: trouble reading harptable:` *reason*

The SIOCGHARPTBL command failed within an *ioctl* system call. The *reason* is any of those described by the *intro*(2) man page.

`hipmap: warning: couldn't resolve name:` *hostname*

The system call, *gethostbyname*, failed for the indicated *IPaddress* or *hostname*. This probably means that the indicated entry does not exist in the host name database (the */etc/hosts* file on the local filesystem or on the NIS server).

`hippi#: board asleep at` *iofile*`:`*line#* `with` *cmd_addr* `not` *cmd_addr* `after` *cmd_addr* `at` *line#*

The indicated IRIS HIPPI board (hippi*#*) controled by the indicated *iofile* is not responding to commands from the driver. The *line#* and *cmd_addr* variables indicate the expected and actual locations in the command queues. Use *hipcntl* to shut down then startup the IRIS HIPPI board. If this does not resolve the problem, the board is probably dysfunctional. Contact the Silicon Graphics Technical Assistance Center.

`hippi#: EEPROM erase FAILED!`

While attempting to erase the FLASH EEPROM on the IRIS HIPPI board, the driver encountered an error. Contact the Silicon Graphics Technical Assistance Center.

`hippi#: erase FAILED while zeroing flash`

While attempting to zero out the FLASH EEPROM on the IRIS HIPPI board, the driver encountered an error. Contact the Silicon Graphics Technical Assistance Center.

`hippi#: flash write failed!`

While attempting to download new firmware into the FLASH EEPROM on the IRIS HIPPI board, the driver encountered an error. Contact the Silicon Graphics Technical Assistance Center.

`hippi#: no board signature!`

While the startup software was attempting to initialize the host-to-board interface, the board's initialization firmware did not respond. Contact the Silicon Graphics Technical Assistance Center.

`hippi_b2h: unknown op:` *command*

The driver received an unknown command from the IRIS HIPPI board. This may indicate a mismatch between the driver and firmware versions. Contact the Silicon Graphics Technical Assistance Center.

`hiptest(DST): couldn't bind fd_i to ULP:` *reason*
`hiptest(SRC): couldn't bind fd_o to ULP:` *reason*

The test's `HIPIOC_BIND_ULP` *ioctl()* call failed. For the source (SRC), the output (writing) call failed; for the destination (DST), the input (reading) call failed. The *reason* is any of those described by the *intro*(2) man page.

This indicates either a problem with the software or too many applications trying to use the ULP-id. Perhaps the driver has not been built into the operating system or the IRIS HIPPI software has not been installed properly. This error message also appears if more than four applications (for example, instances of *hiptest*) try to use ULP-id 0x89.

`hiptest(DST): couldn't open hippi device:` *reason*
`hiptest(SRC): couldn't open hippi device:` *reason*

The IRIS HIPPI board (for example, */dev/hippi#*) was not found. The *reason* is any of those described by the *intro*(2) man page.

For example, this message can indicate that the device file was not found (perhaps the software was not installed properly) or that the board was not located at startup time. To verify the latter, use the */sbin/hinv* command.

`hiptest(DST): couldn't open hippi device: Permission denied`
`hiptest(SRC): couldn't open hippi device: Permission denied`

You must be superuser to use *hiptest*.

`hiptest(SRC): couldn't set D1_SIZE hdr:` *reason*

The test's `HIPIOC_D1_SIZE` *ioctl()* call failed. The *reason* is any of those described by the *intro*(2) man page.

This indicates a problem with the software. Perhaps the driver has not been built into the operating system or the IRIS HIPPI software has not been installed properly.

`hiptest(SRC): couldn't set I-field:` *reason*

The test's `HIPIOC_I` *ioctl()* call failed. The *reason* is any of those described by the *intro*(2) man page.

This indicates a problem with the software. Perhaps the driver has not been built into the operating system or the IRIS HIPPI software has not been installed properly.

`hiptest(DST): data integrity error at offset` *byte_offset*
`hiptest(DST):` *packet#*`: expecting` *tx_data* `got` *rcv_data*
`hiptest(DST): virtual address =` *ptr_rcv_data*

The D2 data in the received packet does not match the D2 data that *hiptest* sent. The *byte_offset* variable indicates the word within the packet where the error was detected. The *tx_data* variable indicates what was sent as compared to *rcv_data*, which was received. The problematic word of received data is located at *ptr_rcv_data*.

`hiptest(DST):` *packet#*`: length error: retv=`*rcv_bytecount* `len2=`*tx_bytecount*

The D2 data set from the received packet is not the same size as that sent. The two bytecounts are displayed: *rcv_bytecount* is for the received packet and *tx_bytecount* is for the packet that was sent.

`hiptest(DST):` *packet#*`: trouble reading header:` *reason*

The *read()* call for the header of the packet specified by *packet#* failed. The *reason* is any of those described by the *intro*(2) man page.

`hiptest(DST):` *packet#*`: header is` *bytecount* `long!?`

The header (that is, FP header and D1 data) for the packet specified by packet# was longer than the header that *hiptest* sent. The length of the received header is indicated by *bytecount*. The test always sends 32 bytes.

`hiptest(DST):` *packet#*`: trouble reading body:` *reason*

The *read()* call for the body of a packet failed. The *reason* is any of those described by the *intro*(2) man page.

`hiptest(SRC):` *packet#*`: write return value:` *returnvalue*
`hiptest(SRC): trouble writing:` *reason*

The *write()* call for the packet failed, or the connection request that was triggered by this *write()* failed to open a connection. For example, if there is a switch between the source and destination endpoints, the I-field may be invalid.

The *packet#* indicates which packet in the series failed, where 0 is the first packet. The *returnvalue* indicates the number of bytes that were successfully sent; when *returnvalue* is -1, the *write()* call failed to send any data. The *reason* is any of those described by the *intro*(2) man page.

`if_hip#: can't output checksum proto` *headertype*

While processing a packet for transmission, the driver found that the header was not TCP nor UDP, and because of this could not calculate a checksum for the packet. The packet was not sent.

`Usage: hipcntl stimeo <value>`

The *hipcntl* command line for setting the source's connection timeout did not contain a valid setting. Valid settings are milliseconds entered in decimal format (for example, **`hipcntl stimeo 1000`** sets the timeout to 1 second).

Index

F

H

I

L

M

N

P

R

S

T

U

V

W

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-2229-002.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389